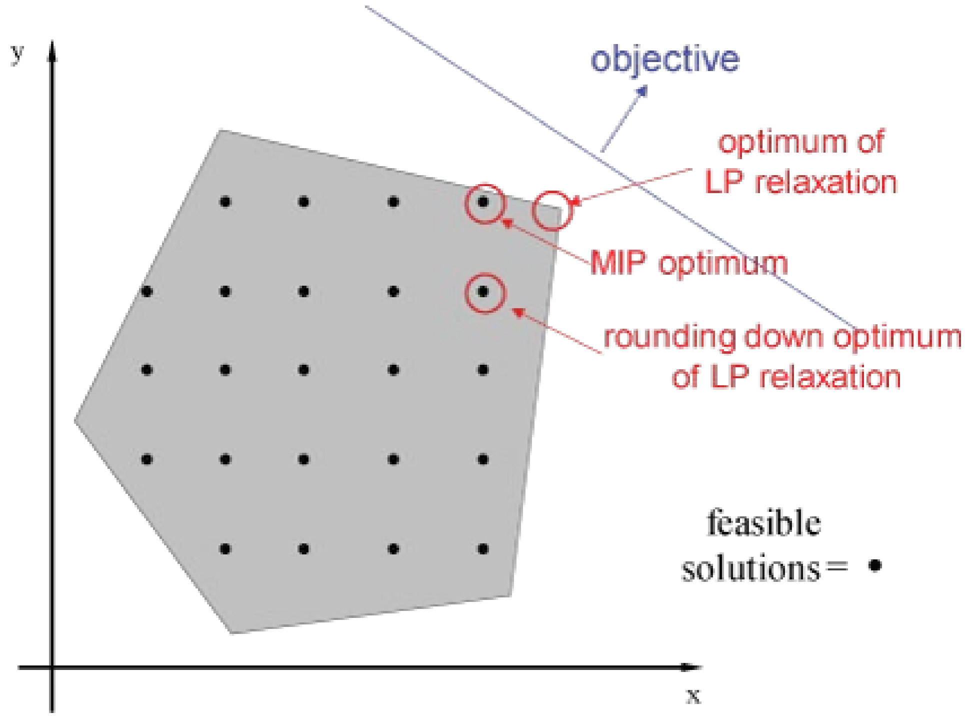


Integer Programming: An Introduction

Martin W. P. Savelsbergh
University of Newcastle

George Dantzig

If I were asked to summarize my early and perhaps my most important contributions to linear programming, I would say they are three:

(1) Recognizing (as a result of my wartime years as a practical program planner) that most practical planning relations could be reformulated as a system of linear inequalities.

(2) Replacing the set of ground rules for selecting good plans by an objective function. (Ground rules at best are only a means for carrying out the objective, not the objective itself.)

(3) Inventing the simplex method which transformed the rather unsophisticated linear-programming model of the economy into a basic tool for practical planning of large complex systems.

The tremendous power of the simplex method is a constant surprise to me. To solve by brute force the assignment problem (which I mentioned earlier) would require a solar system full of nano-second electronic computers running from the time of the big bang until the time the universe grows cold to scan all the permutations in order to select the one which is best. Yet it takes only a moment to find the optimum solution using a personal computer and standard simplex or interior method software.

Ralph Gomory

RESEARCH ANNOUNCEMENTS

The purpose of this department is to provide early announcement of significant new results, with some indications of proof. Although ordinarily a research announcement should be a brief summary of a paper to be published in full elsewhere, papers giving complete proofs of results of exceptional interest are also solicited.

OUTLINE OF AN ALGORITHM FOR INTEGER SOLUTIONS TO LINEAR PROGRAMS

BY RALPH E. GOMORY[1]

Communicated by A. W. Tucker, May 3, 1958

The problem of obtaining the best integer solution to a linear program comes up in several contexts. The connection with combinatorial problems is given by Dantzig in [1], the connection with problems involving economies of scale is given by Markowitz and Manne [3] in a paper which also contains an interesting example of the effect of discrete variables on a scheduling problem. Also Dreyfus [4] has discussed the role played by the requirement of discreteness of variables in limiting the range of problems amenable to linear programming techniques.

It is the purpose of this note to outline a finite algorithm for obtaining integer solutions to linear programs. The algorithm has been programmed successfully on an E101 computer and used to run off the integer solution to small (seven or less variables) linear programs completely automatically.

Ailsa Land

Alison Doig

ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

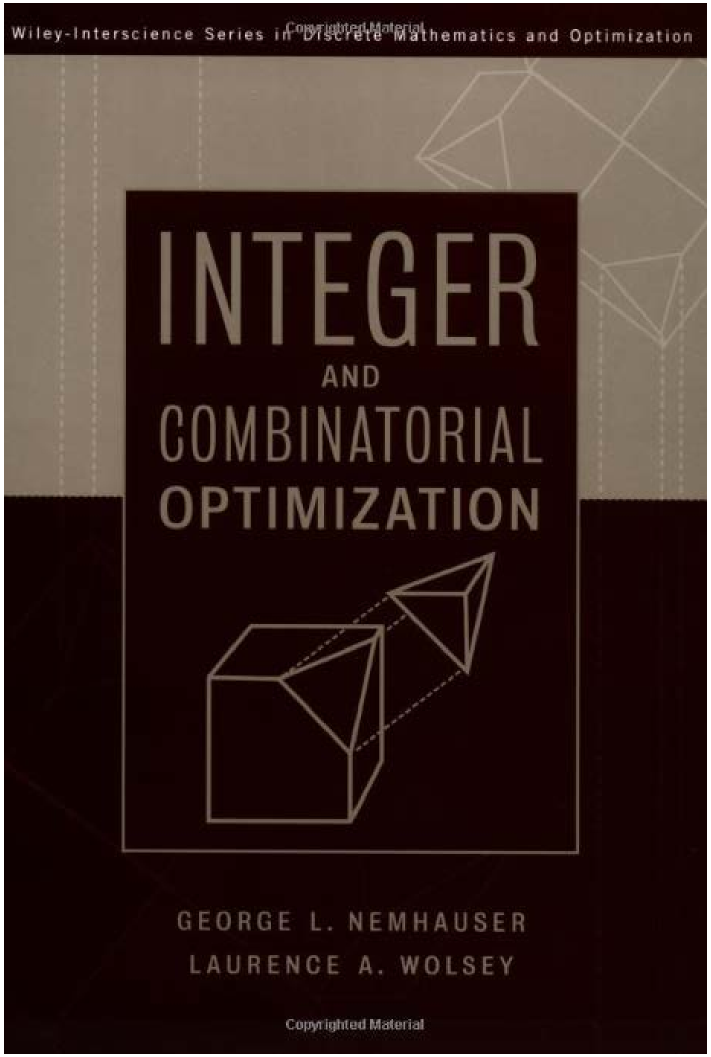
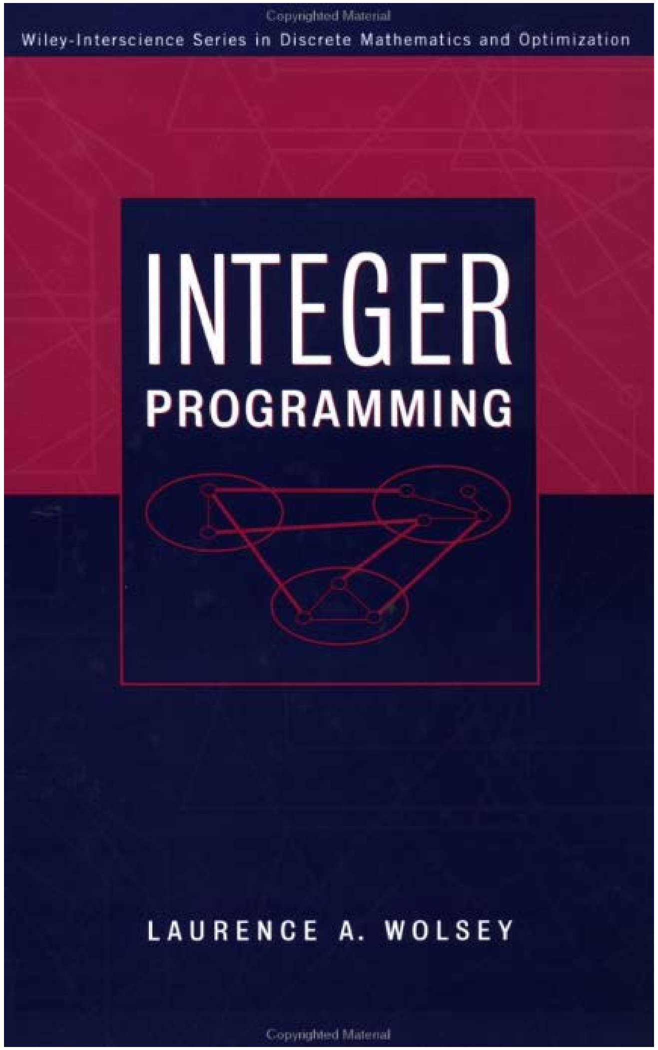
AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

BY A. H. LAND AND A. G. DOIG

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

George Nemhauser

Laurence Wolsey


Wiley-Interscience Series in Discrete Mathematics and Optimization
Copyrighted Material
INTEGER
AND
COMBINATORIAL
OPTIMIZATION
GEORGE L. NEMHAUSER
LAURENCE A. WOLSEY
Copyrighted Material

Copyrighted Material
Wiley-Interscience Series in Discrete Mathematics and Optimization
INTEGER
PROGRAMMING
LAURENCE A. WOLSEY
Copyrighted Material

Zonghau Gu

CPLEX – IP Solver

GuRoBi – IP Solver

Natashia Boland

Will answer all questions about my presentation

Outline

- Some integer programs
- Solving integer programs
  - Branch-and-bound
  - Preprocessing
  - Primal heuristics
  - Valid inequalities
- Reformulation
  - Extended formulations
  - Column generation formulations

Some Integer Programs

Knapsack Problem

Parameters

c_j : value of item j

a_j : weight of item j

b : capacity

Formulation

$$\max \sum_j c_j x_j$$

$$\sum_j a_j x_j \leq b$$

Variables

x_j : whether or not to take item j

Node/Vertex Packing

Parameters

$G = (V, E)$: graph with vertex set V and edge set E

w_v : weight of vertex v

Formulation

$$\max \sum_{v \in V} w_v x_v$$

$$x_u + x_v \leq 1 \text{ for } e = \{u, v\} \in E$$

Variables

x_v : whether or not vertex v is in the packing

Economic Lotsizing

Parameters

c_t : unit production cost in period t

f_t : set up cost in period t

d_t : demand in period t

Formulation

$$\min \sum_{t=1}^{T} c_t x_t + \sum_{t=1}^{T} f_t y_t$$

$$\sum_{s=1}^{t} x_s \geq d_{1t} \text{ for } t = 1, ..., T$$

$$x_t \leq d_{tT} y_t \text{ for } t = 1, ..., T$$

Variables

x_t : production in period t

y_t : whether or not a set up occurs in period t

Generalized Assignment Problem

Parameters

p_{ij} : profit when assigning task j to machine i

w_{ij} : capacity consumption of task j on machine i

d_i : capacity of machine i

Formulation

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \text{ for } j = 1, \ldots, n$$

$$\sum_{j=1}^{n} w_{ij} x_{ij} \leq d_i \text{ for } i = 1, \ldots, m$$

Variables

x_{ij} : whether or not task j is assigned to machine i

Solving Integer Programs

- Three basic concepts
  - **Divide and conquer**
    - Solve a problem by solving two smaller problems
  - **Relax**
    - Remove a constraint to make the problem easier
    - Optimal value may improve
  - **Restrict**
    - Add a constraint to make the problem smaller
    - Optimal value may worsen

Solving Integer Programs

Integer Program

$$z^{IP} = \min cx$$
$$Ax \leq b$$
$$x_j \in \{0,1\} \text{ for } j = 1, ..., n$$

$$z^{IP} = \min\{cx \mid x \in S_{IP}\}$$
$$S_{IP} = \{x \in R^n \mid Ax \leq b, x \in \{0,1\}^n\}$$

Solving Integer Programs

Linear Programming Relaxation

$$z^{LP} = \min cx$$

$$Ax \leq b$$

$$0 \leq x_j \leq 1 \text{ for } j = 1, ..., n$$

$$z^{LP} = \min\{cx \mid x \in S_{LP}\}$$

$$S_{LP} = \{x \in R^n \mid Ax \leq b, 0 \leq x \leq 1\}$$

Solving Integer Programs

Restricted Integer Program

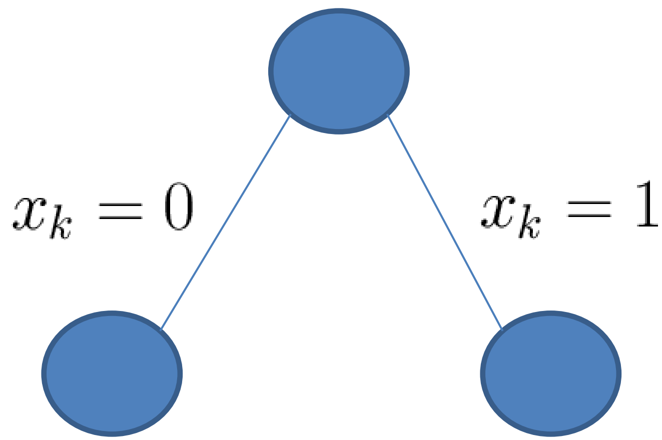
$$z_{(k,1)}^{IP} = \min cx$$
$$Ax \leq b$$
$$x_k = 1$$
$$x_j \in \{0,1\} \text{ for } j = 1, ..., n$$

$$z_{(k,1)}^{IP} = \min\{cx \mid x \in S_{IP}^{(k,1)}\}$$
$$S_{IP}^{(k,1)} = \{x \in R^n \mid Ax \leq b, x_k = 1, x \in \{0,1\}^n\}$$

Solving Integer Programs

- Divide and conquer

$$z^{IP} = \min\{cx \mid x \in S_{IP}\}$$



$$z^{IP}_{(k,0)} = \min\{cx \mid x \in S^{(k,0)}_{IP}\} \qquad z^{IP}_{(k,1)} = \min\{cx \mid x \in S^{(k,1)}_{IP}\}$$

$$\boxed{z^{IP} = \min\{z^{IP}_{(k,0)}, z^{IP}_{(k,1)}\}}$$

Divide and conquer: Solve original IP by solving two smaller restricted IPs

Solving Integer Programs

- Relaxation

$$S_{LP} \supseteq S_{IP} \qquad\qquad z^{LP} \leq z^{IP}$$

- Restriction

$$S_{IP}^{(k,1)} \subseteq S_{IP} \qquad\qquad z_{(k,1)}^{IP} \geq z^{IP}$$

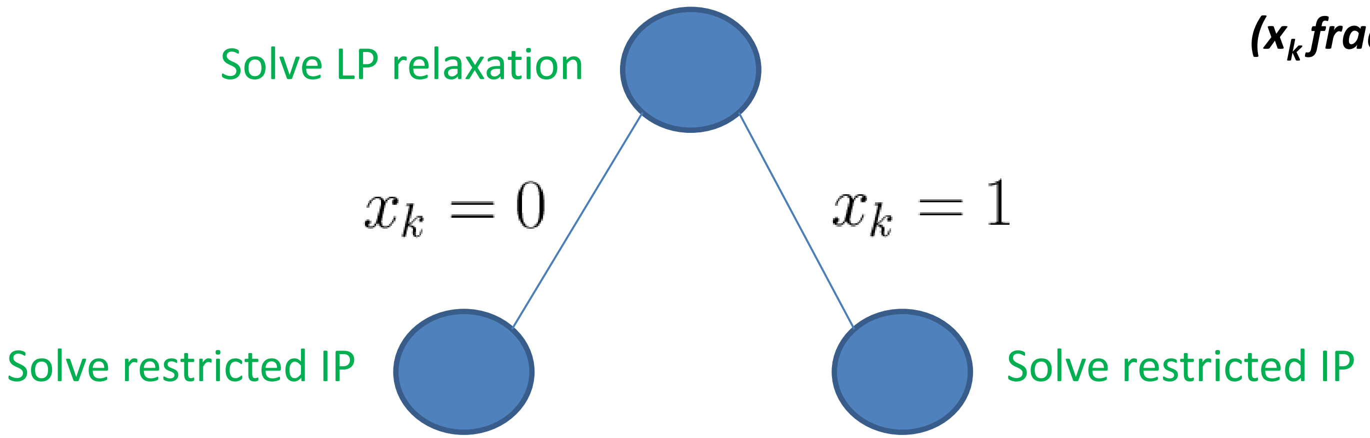
Solving Integer Programs

- Recall: Solving linear programs is easy
- What can happen if we solve the LP relaxation of our integer program?
  - Infeasible
    - IP is infeasible - stop
  - Feasible & integer
    - Optimal IP solution – record & stop
  - Feasible & fractional
    - Divide & conquer

Solving Integer Programs

$$z^{LP} = \min\{cx \mid x \in S_{LP}\}$$

solution
feasible & fractional
(x_k fractional)



$$z^{IP}_{(k,0)} = \min\{cx \mid x \in P^{(k,0)}_{IP}\}$$

$$z^{IP}_{(k,1)} = \min\{cx \mid x \in P^{(k,1)}_{IP}\}$$

Branching

Solving Integer Programs

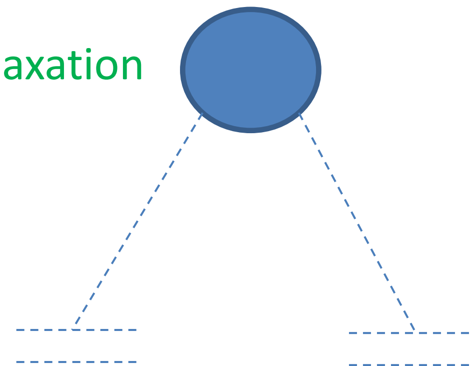
- Observation: When we solve the LP, we relax the problem. Therefore, the IP solution value will never be better
- Observation: When we branch, we restrict the problem. Therefore, the solution value will never improve
- *Observation: If the value of the LP relaxation is worse than the value of the best known IP solution, then the restricted IP cannot provide a better solution*

Solving Integer Programs

$$z^{LP} = \min\{cx \mid x \in S_{LP}\}$$

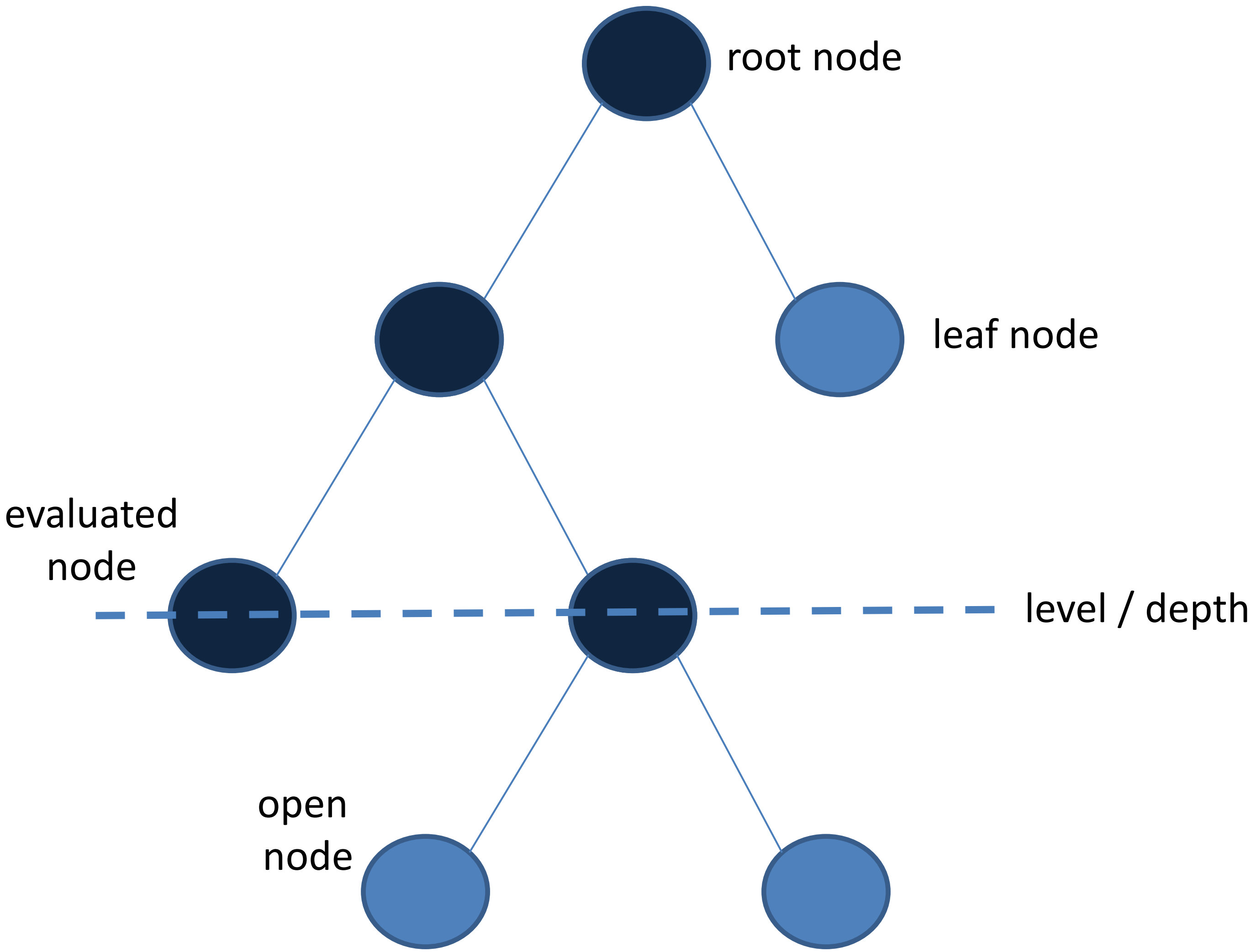
$$\boxed{z^{LP} \geq z^{IP}_{best}}$$

Solve LP relaxation

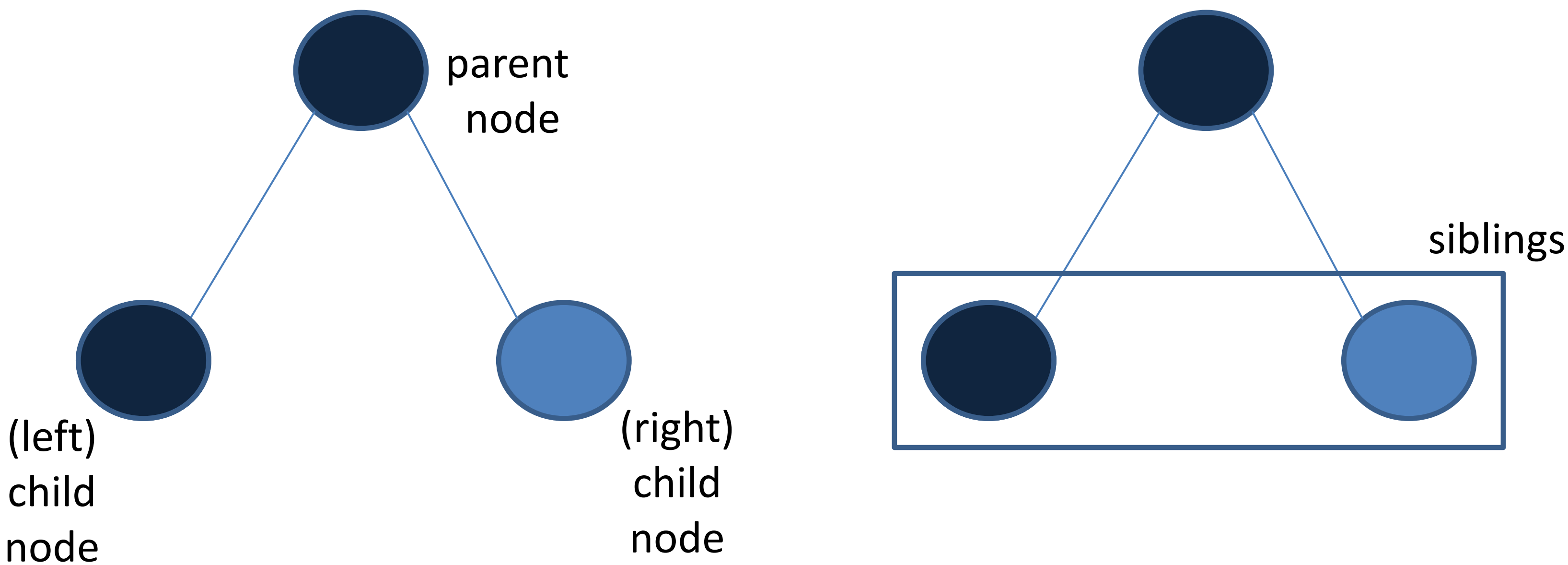


Bounding

Branch-and-Bound Tree/Search Tree



Branch-and-Bound Tree/Search Tree



How Can We Improve Basic Branch-and-Bound?

- Solve smaller linear programs
- Improve linear programming bounds
- Find feasible solutions quickly
- Branch intelligently

Branch Intelligently

Branch Intelligently

- Key questions
  - Which variable to branch on?
  - What node to evaluate next?

Variable Selection

- *Focus: improving global bound*

- Algorithm
  - For all candidate variables calculate a score
  - Select the candidate with the highest score
- Score function

$$score(q^-, q^+) = (1 - \mu)\min\{q^-, q^+\} \; + \; \mu \max\{q^-, q^+\}$$

left child

right child

Most Infeasible Branching

$$f_i^+ = \lceil x_i^{LP} \rceil - x_i^{LP}$$

$$f_i^- = x_i - \lfloor x_i^{LP} \rfloor$$

$$s_i = score(q_i^-, q_i^+) = 1 \cdot \min\{f_i^-, f_i^+\} + 0 \cdot \max\{f_i^-, f_i^+\}$$

If the goal is to improve the global bound, then quality of a branching has to be measured by the change in objective function value at the child nodes

Pseudocost Branching

$$\sigma^+ = \Delta^+ / f^+$$

observed per unit change in objective function at right child

$$\Sigma^+$$

sum of observed per unit change in objective function at right child over all branchings on the variable

$$\eta^+$$

number of branchings on the variable

$$\Phi^+ = \Sigma^+ / \eta^+$$

average observed per unit change at right child for the variable

$$s_i = score(f_i^+ \Phi_i^+, f_i^- \Phi_i^-)$$

Pseudocost Branching

- Initialization?
  - Average over all known pseudocosts

Strong Branching

- Calculate change in objective function value at children of all candidate variables (i.e., solve two LPs)

$$s_i = score(\Delta_i^+, \Delta_i^-)$$

- Efficiency
  - Restrict number of pivots
  - Restrict candidate set
    - Dynamic
      - order by pseudocost
      - no improved candidate for *k* iterations

Hybrid Pseudocost/Strong Branching

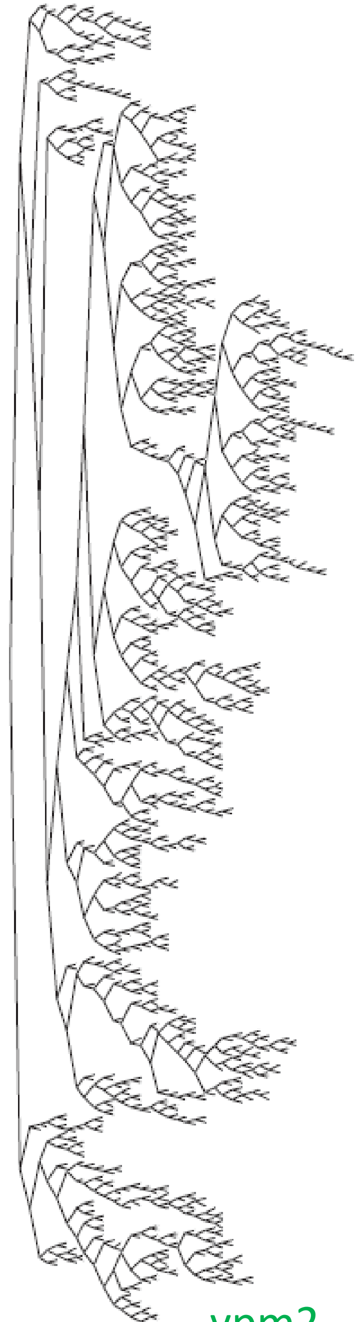
- Strong branching up to level *d* in the tree and pseudocost branching for levels larger than *d*
- Pseudocost branching with strong branching initialization

Reliability Branching

- Generalizes pseudocost branching with strong branching initialization
- Use strong branching when pseudocosts are unreliable

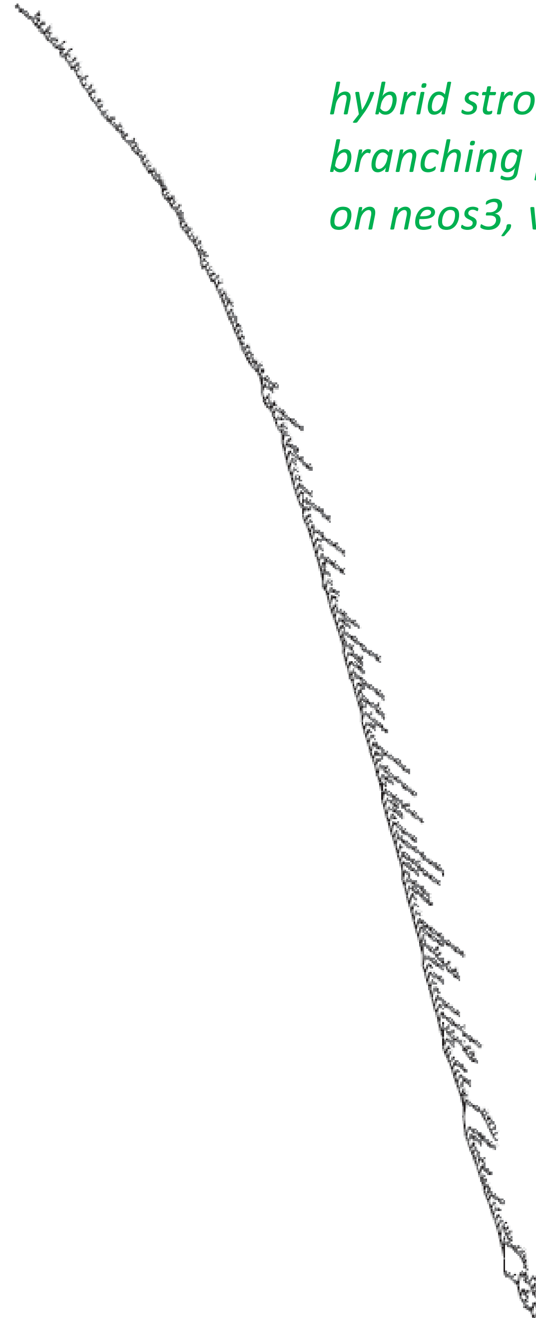
$$\min\{\eta_i^+, \eta_i^-\} < \eta_{rel}$$

strong branching



vpm2

hybrid strong/pseudocost branching performs poorly on neos3, we can see why…



neos3

Recent Ideas

Perform one or more partial tree searches to determine “key” variables to branch on

- Information-based branching
- Backdoor branching

Restarts and backdoor are concept “borrowed” from constraint programming

Node selection

- *Focus: finding (good) feasible solutions*

- Static
- Estimate-based
- Backtracking

Static

- Best-bound
  - Guarantees that only nodes that have to be evaluated are evaluated
  - May not be efficient since consecutive LPs differ substantially
- Depth-first
  - Requires very little memory
  - Extremely efficient because consecutive LPs differ very little
  - May evaluate many superfluous nodes

Estimate-based

- Best projection

$$s = \sum_j \min\{f_j^+, f_j^-\}$$

per unit
infeasibility cost

$$E = z^{LP} - \left(\frac{z_{root}^{LP} - z_{best}^{IP}}{s_{root}}\right)s$$

- Best estimate

$$E = z^{LP} - \sum \min\{f_j^+ \Phi_j^+, f_j^- \Phi_j^-\}$$

Backtrack

- Goal: exploit advantage of depth-first search
- Implementation: depth-first until the objective value reaches a threshold, then jump (e.g, best-estimate)

$$T = \min_{i \in open\ nodes} E_i$$

Preprocessing

Preprocessing

- Infeasibility detection
- Redundancy detection
- Improving bounds
- Fixing variables
- Improving coefficients

Preprocessing

integer program

$$\min cx$$
$$Ax \leq b$$
$$x \in \{0,1\}^n$$

solve

row *i*

$$z = \min \sum_{j \in B^+} a_j^i x_j - \sum_{j \in B^-} a_j^i x_j$$
$$A^i x \leq b^i$$
$$x \in \{0,1\}^n$$

constraints without constraint *i*

analysis

$$z > b_i \Rightarrow \text{infeasible}$$

Detecting redundancy

$$z = \max \sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j$$

$$z < b_i \Rightarrow \text{redundant}$$

Improving bounds

$$z = \min \sum_{j \in B^+ \setminus \{k\}} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j$$

$$x_k \leq \lfloor \frac{(b_i - z)}{a^i_k} \rfloor$$

Improving bounds

$$z = \min \sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^- \setminus \{k\}} a^i_j x_j$$

$$x_k \geq \lceil \frac{(z - b_i)}{a^i_k} \rceil$$

$$z = \min \sum_{j \in B^+} a_j^i x_j - \sum_{j \in B^-} a_j^i x_j$$

Fixing variables

$$x_k = 1 \quad (k \in B^+)$$

$$z > b_i \Rightarrow x_k = 0$$

$$z = \min \sum_{j \in B^+} a_j^i x_j - \sum_{j \in B^-} a_j^i x_j$$

Fixing variables

$$x_k = 0 \quad (k \in B^-)$$

$$z > b_i \Rightarrow x_k = 1$$

$$z = \max \sum_{j \in B^+} a_j^i x_j - \sum_{j \in B^-} a_j^i x_j$$

Improving coefficient

$$x_k = 0 \quad (k \in B^+)$$

$$z \leq b_i \Rightarrow \text{reduce coefficients } a_k^i \text{ and } b_i \text{ by } b_i - z$$

$$z = \max \sum_{j \in B^+} a_j^i x_j - \sum_{j \in B^-} a_j^i x_j$$

Improving coefficient

$$x_k = 1 \quad (k \in B^-)$$

$$z \leq b_i \Rightarrow \text{reduce coefficients } a_k^i \text{ and } b_i \text{ by } b_i - z$$

Preprocessing

- Implementation

$$\min cx$$
$$Ax \leq b$$
$$x \in \{0,1\}^n$$

$$z = \min \sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j$$
$$A^i x \leq b^i$$
$$x \in \{0,1\}^n$$

ignore completely or
ignore most

Probing

Tentatively fix a variable and explore the consequences

$$7x_1 + 3x_2 - 4x_3 - 2x_4 \leq 1$$
$$-2x_1 + 7x_2 + 3x_3 + x_4 \leq 6$$

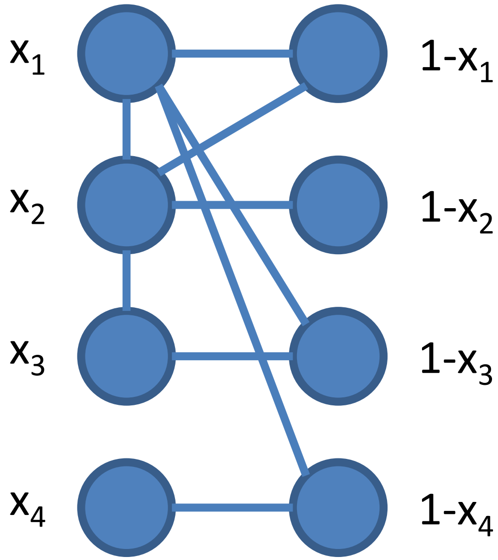
fix consequences

$$x_1 = 1 \;\Rightarrow\; x_3 = x_4 = 1 \text{ and } x_2 = 0$$ (first constraint)

$$x_2 = 1 \;\Rightarrow\; x_1 = 1 \text{ and } x_3 = 0$$ (second constraint)

Probing

Conflict graph: representation of the implications

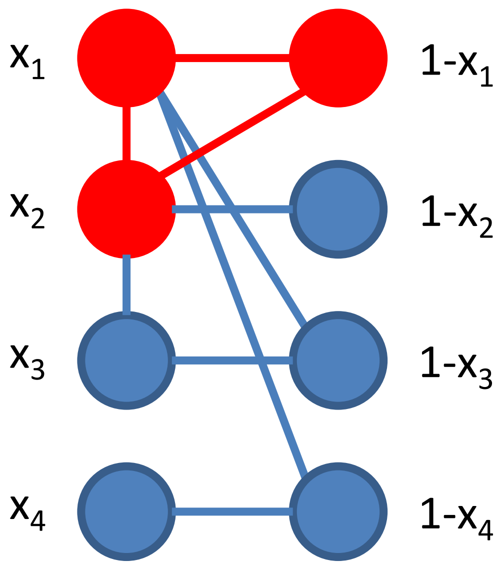


An edge represents variables that cannot be one at the same time

e.g., $x_1 + x_2 \leq 1$

Probing

Conflict graph: representation of the implications



clique inequality:

$x_1 + x_2 + 1\text{-}x_1 \leq 1$

or

$x_2 \leq 0$

implementation: requires solving a maximum clique problem

Reduced Cost Fixing

Integer program $$\min\{c^T x | Ax \le b, x \text{ integer}\}$$

Optimal solution linear programming relaxation $$z^{LP} = \min\{c^T x | Ax \le b, x \ge 0\}$$

optimal dual variables

Reduced cost: $$\bar{c} = c - A^T y$$

What if? $$x_j = 0 \text{ and } z^{LP} + \bar{c}_j > z^{IP}_{best}$$

Permanently set $x_j = 0$

Primal Heuristics

Primal Heuristics

- Relaxation Induced Neighborhood Search (RINS)
- Local branching
- Feasibility pump

Relaxation Induced Neighborhood Search

- Suppose a known feasible solution x_{IP} exists
- Consider a linear programming solution x_{LP} at a node of the search tree
- Attempt to find a better feasible solution by solving a restricted IP in which the variables where x_{IP} and x_{LP} agree are fixed

Relaxation Induced Neighborhood Search

- Implementation question
  - When to apply?
- Variation
  - Rather than using a known IP solution and an LP solution, use two known IP solutions to determine restricted IP (path relinking)

Local Branching

- Search in the neighborhood of a known feasible solution $\bar{x}$ by limiting the possible changes

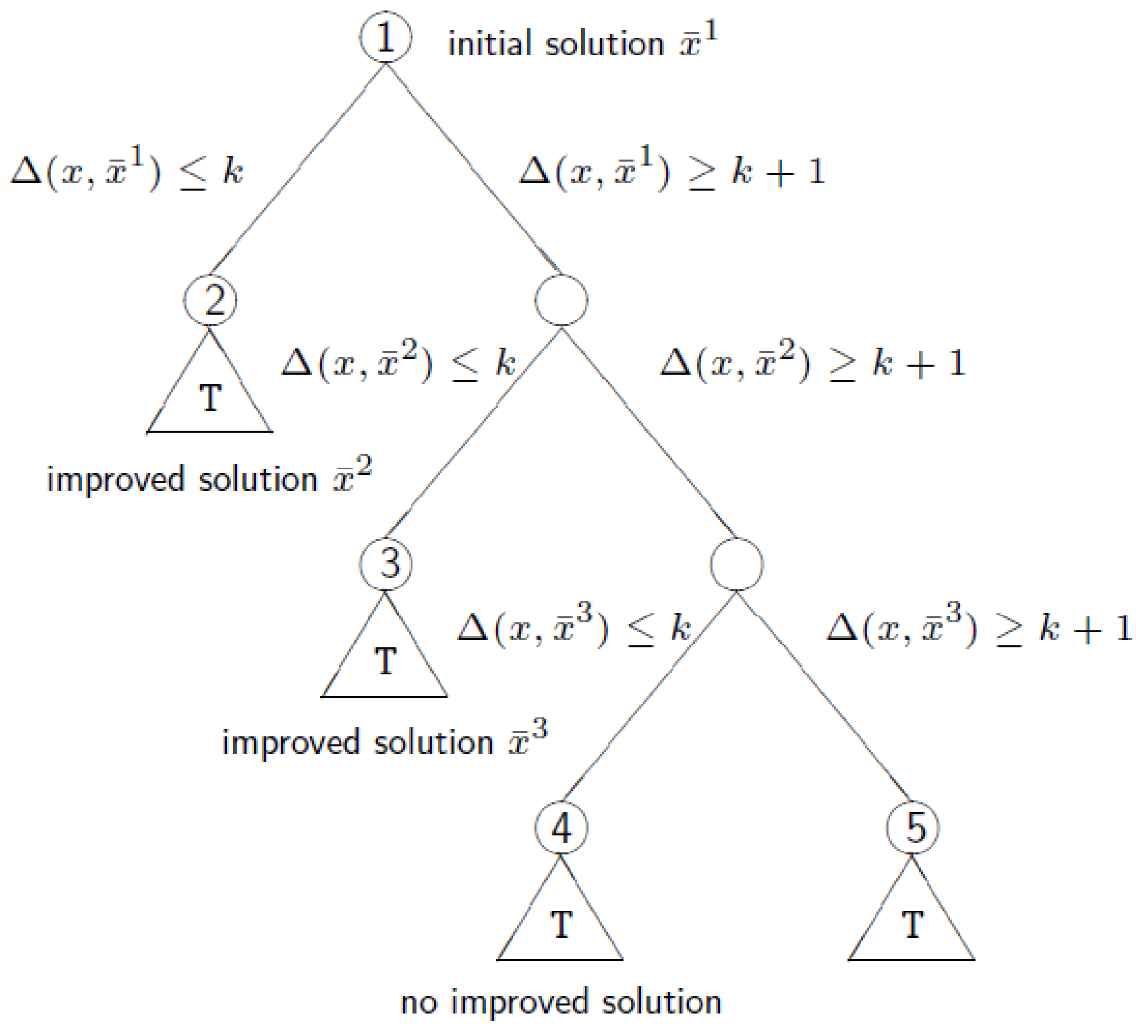
Known feasible solution

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k$$

Set of variables at 1 in known feasible solution

Set of variables at 0 in known feasible solution

Local Branching (Cont.)



The Feasibility Pump (FP)

$x \in R^n$ and $J \subseteq \{1,\ldots,n\}$

$$\begin{array}{ll} \min & cx \\ \text{s.t.} & Ax \le b \\ & x_j \text{ is integer } \forall j \in J \end{array}$$

Feasibility Pump: The general idea

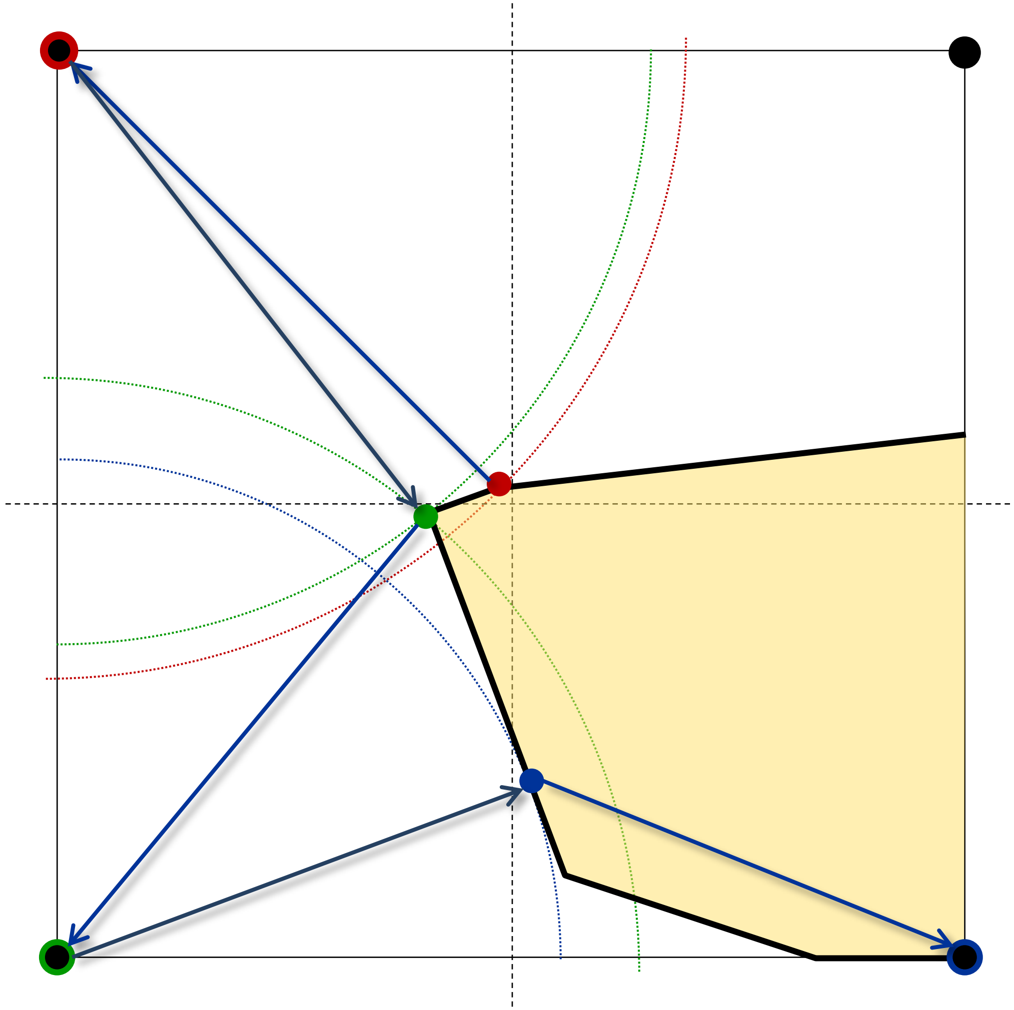
Start with **LP feasible** ***x***

z ← **closest integer** point to ***x*** ← **rounding** ***x***, ***i***.e. [***x***]

x ← **closest LP feasible** point to ***z*** ← **projecting** ***z*** onto LP feasible region

Repeat until ***z*** is **feasible**

The Feasibility Pump (FP)



Feasibility Pump: The general idea

Start with **LP feasible** $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow$ **closest integer** point to $\boldsymbol{x}$

$\boldsymbol{x} \leftarrow$ **closest LP feasible** point to $\boldsymbol{z}$

Repeat until $\boldsymbol{z}$ is **feasible**

● $\boldsymbol{x}_1$ ● $\boldsymbol{z}_1$

● $\boldsymbol{x}_2$ ● $\boldsymbol{z}_2$

● $\boldsymbol{x}_3$ ● $\boldsymbol{z}_3$

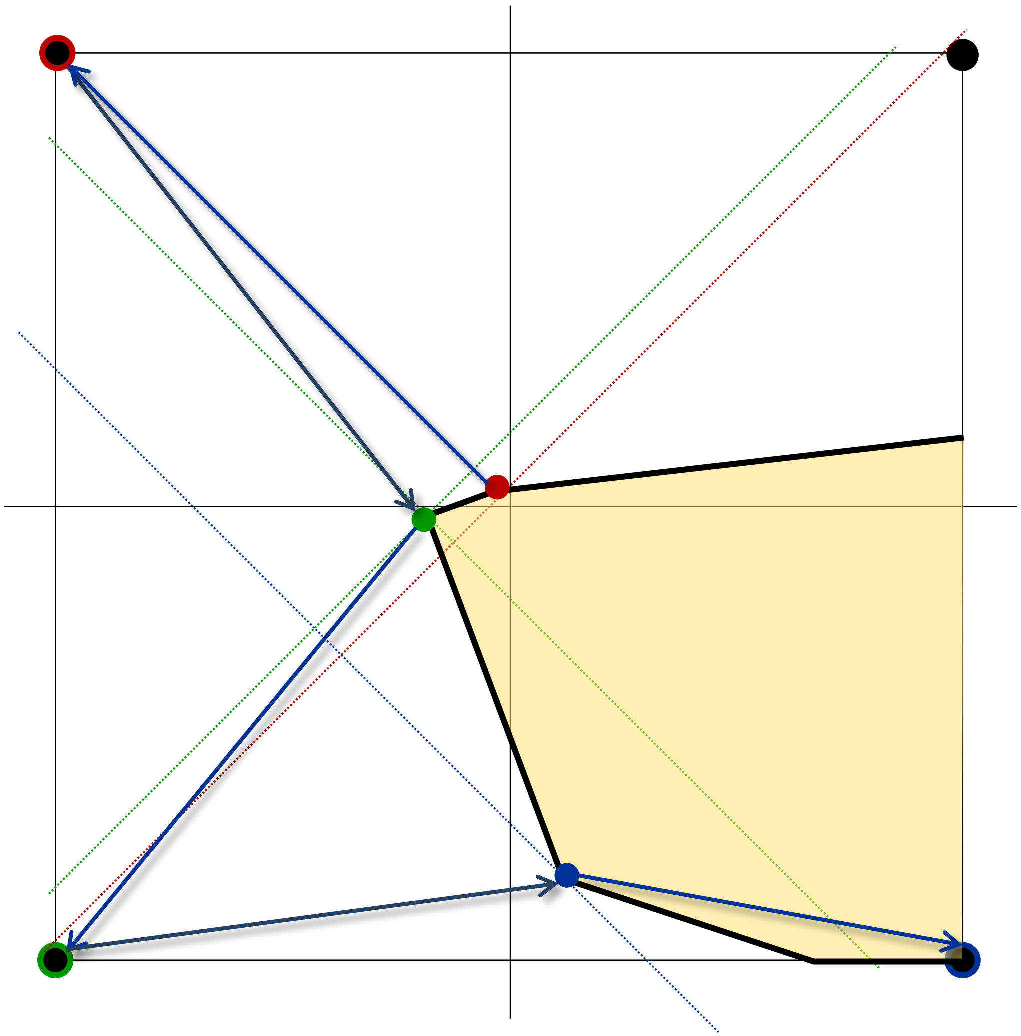
Two scenarios:

1. Feasible $\boldsymbol{z}$
2. Cycling (i.e., $[\boldsymbol{x}] = \boldsymbol{z}$ and $\mathbf{proj}_{LP}(\boldsymbol{z}) = \boldsymbol{x}$)

$\mathrm{d}(\boldsymbol{x}_1,\boldsymbol{z}_1) \leq \mathrm{d}(\boldsymbol{z}_1,\boldsymbol{x}_2) \leq \mathrm{d}(\boldsymbol{x}_2,\boldsymbol{z}_2) \leq \mathrm{d}(\boldsymbol{z}_2,\boldsymbol{x}_3) \leq \mathrm{d}(\boldsymbol{x}_3,\boldsymbol{z}_3)$

$\boldsymbol{x}$ □integer $\boldsymbol{z}$ □feasible

The Feasibility Pump (FP)



Feasibility Pump: The general idea

Start with **LP feasible** ***x***

z ← **closest integer** point to ***x***

x ← **closest LP feasible** point to ***z***

Repeat until ***z*** is **feasible**

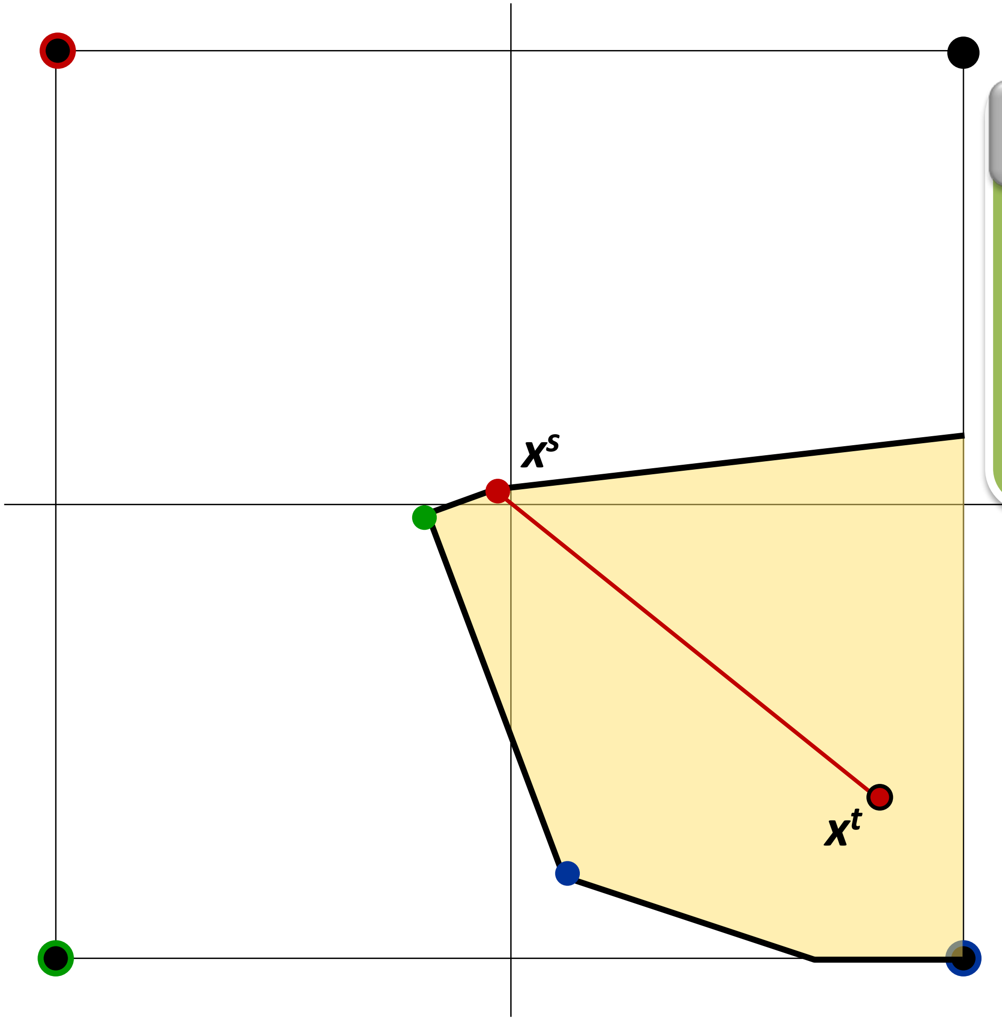
Spends most time in projection procedure:

- May overlook good integer solutions close to ***x***

Fix:

- Spend more time around FP iterates ***x*** to find feasible integer solutions rather than relying on naïve rounding
- Make search more balanced

Line Search within FP



A substitute for rounding

For each FP iterate $\boldsymbol{x}$, round all points along a line segment passing through $\boldsymbol{x}$ and a point ***deep within the feasible region***.

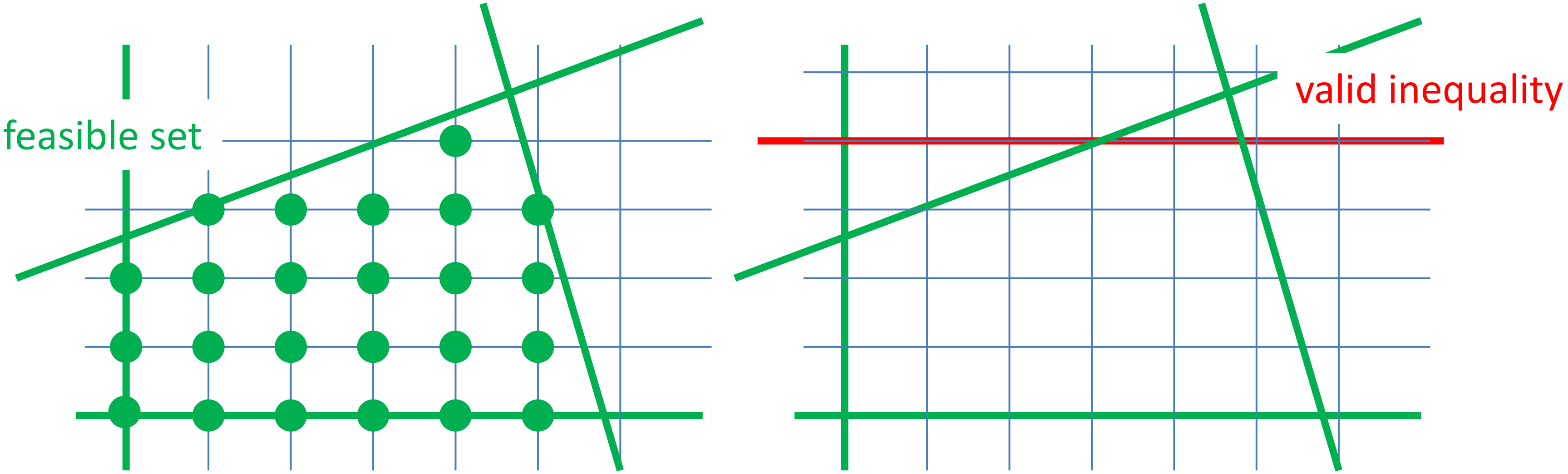
Q. How to find suitable $\boldsymbol{x}^t$?

Q. How to find all rounded solutions along the shooting line ***efficiently***?

The chosen line segment is called the shooting line with starting point $\boldsymbol{x}^s$ and end point $\boldsymbol{x}^t$.

Valid Inequalities

Valid Inequalities



valid inequality: an inequality such that all feasible solutions satisfy the inequality

Valid Inequalities

Polyhedron $$P = \{x \in R^n | Ax \leq b\}$$

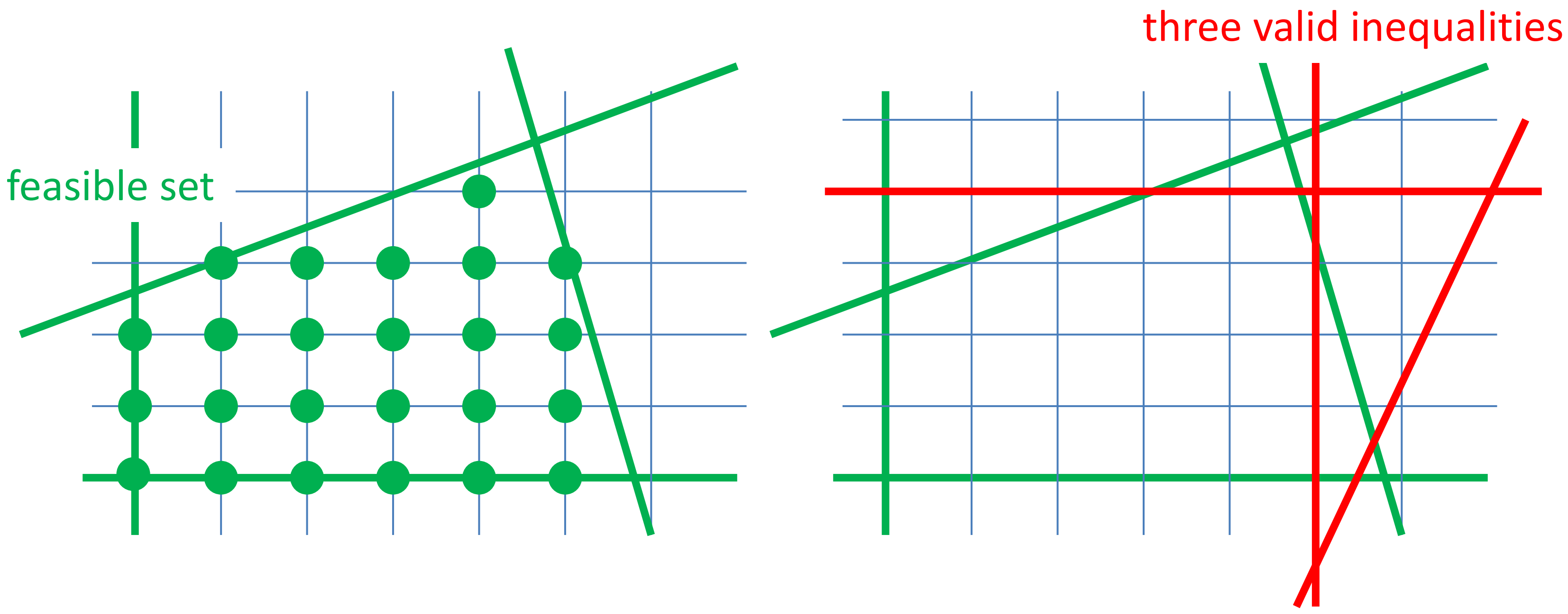
Feasible set $$S = P \cap \{0, 1\}^n$$

Valid inequality $$\pi x \leq \pi_0 \text{ for all } x \in S$$

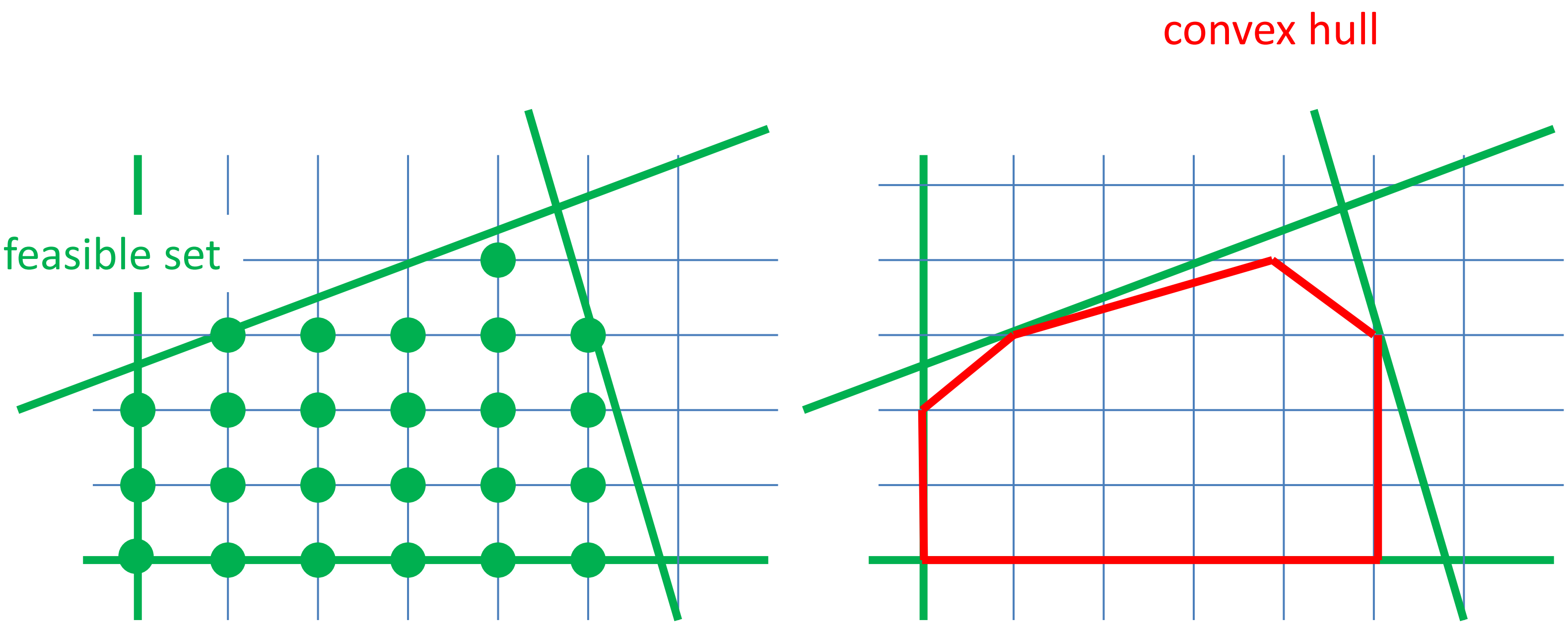
Valid Inequalities / Cuts

- Are some cuts better than others?
- Can we characterize the strongest cut?

Valid Inequalities

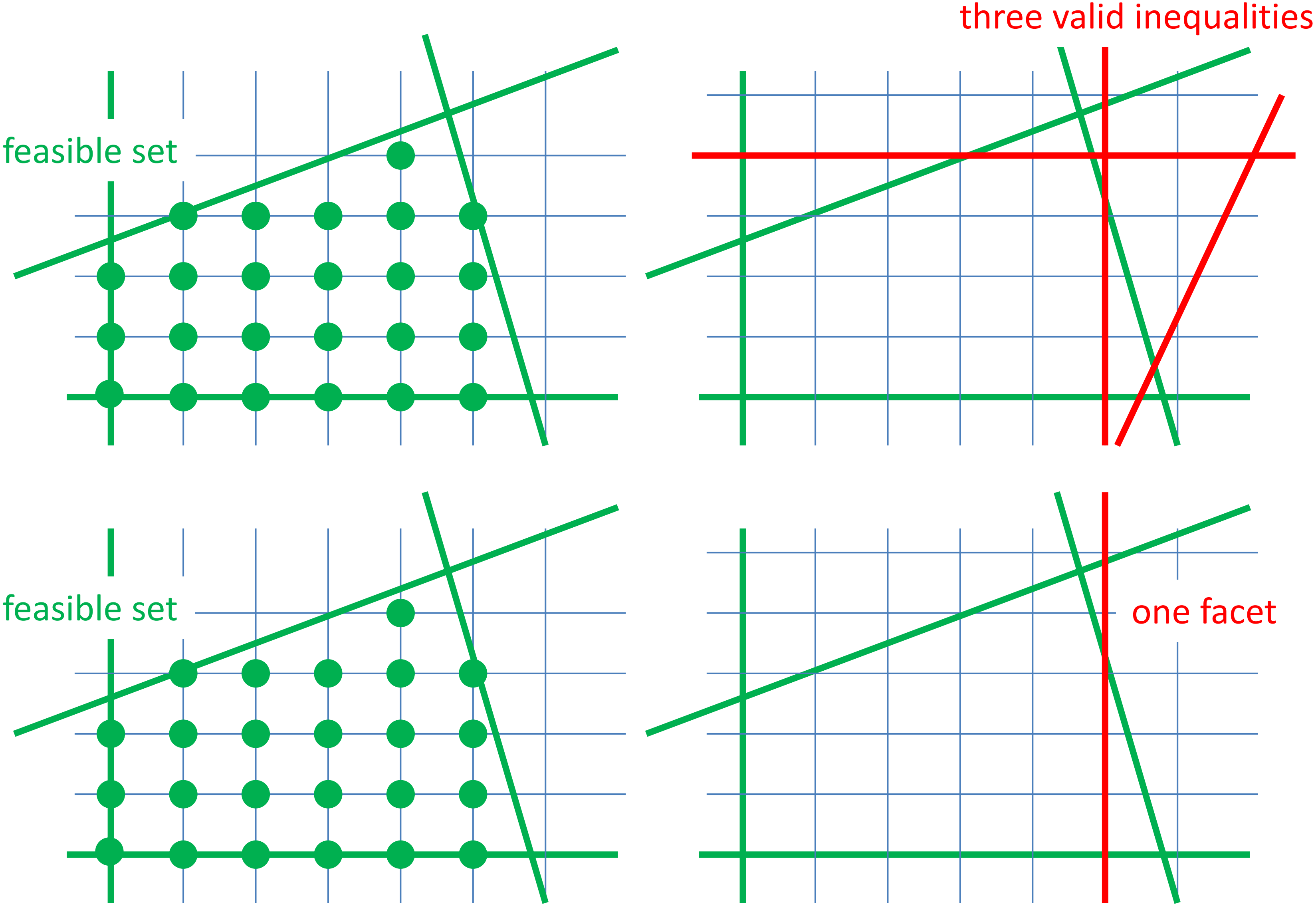


Valid Inequalities



a valid inequality that is necessary in the description of the convex hull of feasible solutions is the strongest possible valid inequality and called a facet inducing inequality (a facet)

Valid Inequalities



Valid Inequalities / Cuts

- Problem specific cuts
  - Node / vertex packing
  - Economic lotsizing
- Substructure cuts
  - Knapsack problem
- General cuts
  - Gomory cuts

Cuts for a relaxation of the problem

Node/Vertex Packing

Parameters

$G = (V, E):$ graph with vertex set V and edge set E

$w_v:$ weight of vertex v

Formulation

$$\max \sum_{v \in V} w_v x_v$$

$$x_u + x_v \leq 1 \text{ for } e = \{u, v\} \in E$$

Variables

x_v : whether or not vertex v is in the packing

Node / Vertex Packing

Odd hole inequality: $$\sum_{v \in O} x_v \leq \lfloor \frac{|H|}{2} \rfloor \quad O \subseteq V; O \text{ odd}$$

Economic Lotsizing

Parameters

c_t : unit production cost in period t

f_t : set up cost in period t

d_t : demand in period t

Formulation

$$\min \sum_{t=1}^{T} c_t x_t + \sum_{t=1}^{T} f_t y_t$$

$$\sum_{s=1}^{t} x_s \geq d_{1t} \text{ for } t = 1, ..., T$$

$$x_t \leq d_{tT} y_t \text{ for } t = 1, ..., T$$

Variables

x_t : production in period t

y_t : whether or not a set up occurs in period t

Economic Lotsizing

Valid inequality

$$S \subseteq L := \{1, ..., l\} \text{ for } l = 1, ..., T$$

$$\sum_{t \in L \setminus S} x_t + \sum_{t \in S} d_{tl} y_t \geq d_{1l}$$

Knapsack Problem

Parameters

c_j : value of item j

a_j : weight of item j

b : capacity

Formulation

$$\max \sum_j c_j x_j$$

$$\sum_j a_j x_j \leq b$$

Variables

x_j : whether or not to take item j

Cover Cuts

Cover: $$C \subseteq \{1, ..., n\} : \sum_{j \in C} a_j > b$$

Cover inequality: $$\sum_{j \in C} x_j \leq |C| - 1$$

General Cuts

- Gomory cut $\sum_j \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor \quad (u \geq 0, Ax \leq b)$
- Proof of validity

$$
\begin{aligned}
x \text{ feasible} \quad &\Rightarrow \quad Ax \leq b \\
&\underset{(u \geq 0)}{\Rightarrow} \quad uAx \leq ub \\
&\Rightarrow \quad \sum_j (ua_j) x_j \leq ub \\
&\underset{(x \geq 0)}{\Rightarrow} \quad \sum_j \lfloor ua_j \rfloor x_j \leq ub \\
&\underset{(x \text{ feasible, } \lfloor ua_j \rfloor \text{ integer})}{\Rightarrow} \quad \sum_j \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor
\end{aligned}
$$

How to determine valid inequalities?

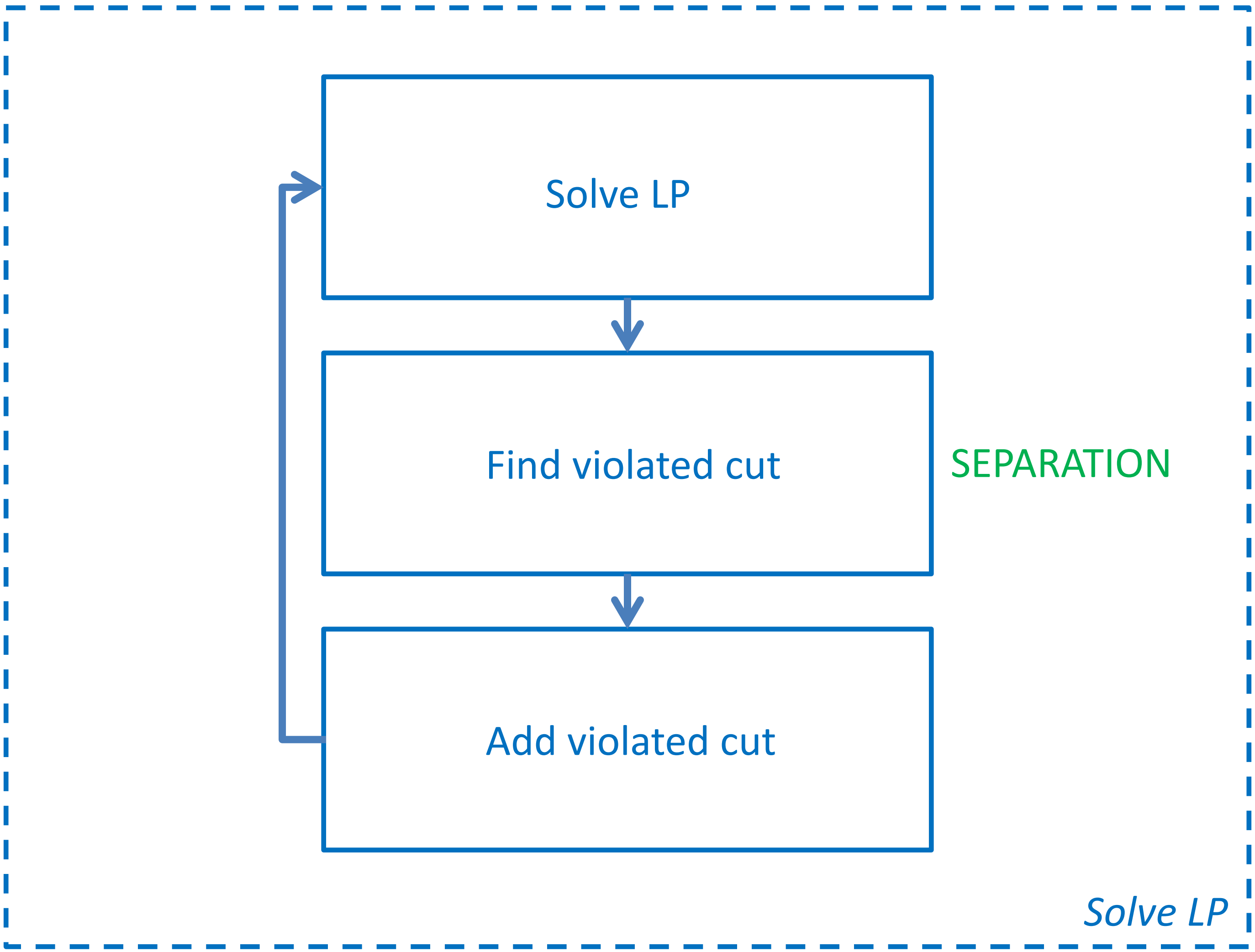
- Exploit problem knowledge
- Study LP solutions
- Use PORTA

How to use valid inequalities?

- How to handle an exponential number of valid inequalities?

- Are valid inequalities useful if there not facets?
- How to determine whether a valid inequality is a facet?

Cut Generation



Separation

- Odd-hole inequalities
- (l, S) inequalities
- Cover inequalities

Odd-hole inequalities

(l, S)-Inequalities

Separation

$$\sum_{t=1}^{l} \min\{x_t^*, d_{tl} y_t^*\} \leq d_{1l}$$

Cover Inequalities

Separation

$$\min \sum_j (1 - x_j^*) z_j$$

$$\sum_j a_j z_j > b$$

$$z_j \in \{0, 1\}$$

Up Lifting (Cover Inequality)

valid inequality
(when $x_1 = 0$)

$$\pi_2 x_2 + \ldots + \pi_n x_n \leq \pi_0$$

consider

$$\alpha x_1 + \pi_2 x_2 + \ldots + \pi_n x_n \leq \pi_0$$

valid when

$$\alpha \leq \pi_0 - \zeta$$

where

$$\zeta = \max \pi_2 x_2 + \ldots + \pi_n x_n$$
$$a_2 x_2 + \ldots + a_n x_n \leq b - a_1$$

Down Lifting (Cover Inequality)

valid inequality
(when $x_1 = 1$)

$$\pi_2 x_2 + \ldots + \pi_n x_n \leq \pi_0$$

consider

$$\gamma x_1 + \pi_2 x_2 + \ldots + \pi_n x_n \leq \pi_0 + \gamma$$

valid when

$$\gamma \leq \zeta - \pi_0$$

where

$$\begin{aligned} \zeta = \max\ & \pi_2 x_2 + \ldots + \pi_n x_n \\ & a_2 x_2 + \ldots + a_n x_n \leq b \end{aligned}$$

Cover Inequalities

- Should the separation problem be solved using an exact method?
- Should the lifting problem be solved using an exact method?
- In what order should variables be lifted?

Up Lifting

Theorem. Suppose $S \subseteq B^n$, $S^\delta = S \cap \{x \in B^n : x_1 = \delta\}$ for $\delta \in \{0,1\}$, and

$$\pi_2 x_2 + \pi_3 x_3 + \ldots \pi_n x_n \leq \pi_0 \qquad (*)$$

is valid for S^0.

If $S^1 = \emptyset$ then $x_1 \leq 0$ is valid for S.

If $S^1 \neq \emptyset$ then

$$\alpha x_1 + \pi_2 x_2 + \pi_3 x_3 + \ldots \pi_n x_n \leq \pi_0 \qquad (**)$$

is valid for S for any $\alpha \leq \pi_0 - \zeta$ where

$$\zeta = \max\{\pi_2 x_2 + \pi_3 x_3 + \ldots \pi_n x_n : x \in S^1\}.$$

Moreover, if (*) defines a facet of conv(S^0) and $\alpha = \pi_0 - \zeta$ then (**) gives a facet of conv(S).

Down Lifting

Theorem. Suppose $S \subseteq B^n$, $S^\delta = S \cap \{x \in B^n : x_1 = \delta\}$ for $\delta \in \{0,1\}$, and

$$\pi_2 x_2 + \pi_3 x_3 + \ldots \pi_n x_n \leq \pi_0 \qquad (*)$$

is valid for S^1.

If $S^0 = \emptyset$ then $x_1 \geq 1$ is valid for S.

If $S^0 \neq \emptyset$ then

$$\gamma x_1 + \pi_2 x_2 + \pi_3 x_3 + \ldots \pi_n x_n \leq \pi_0 + \gamma \qquad (***)$$

is valid for S for any $\gamma \geq \zeta - \pi_0$ where

$$\zeta = \max\{\pi_2 x_2 + \pi_3 x_3 + \ldots \pi_n x_n : x \in S^0\}.$$

Moreover, if (*) defines a facet of $\text{conv}(S^1)$ and $\gamma = \zeta - \pi_0$ then (***) gives a facet of $\text{conv}(S)$.

Reformulation

Reformulation

- Disaggregation
- Extended formulations
- Column generation formulations

Disaggregation

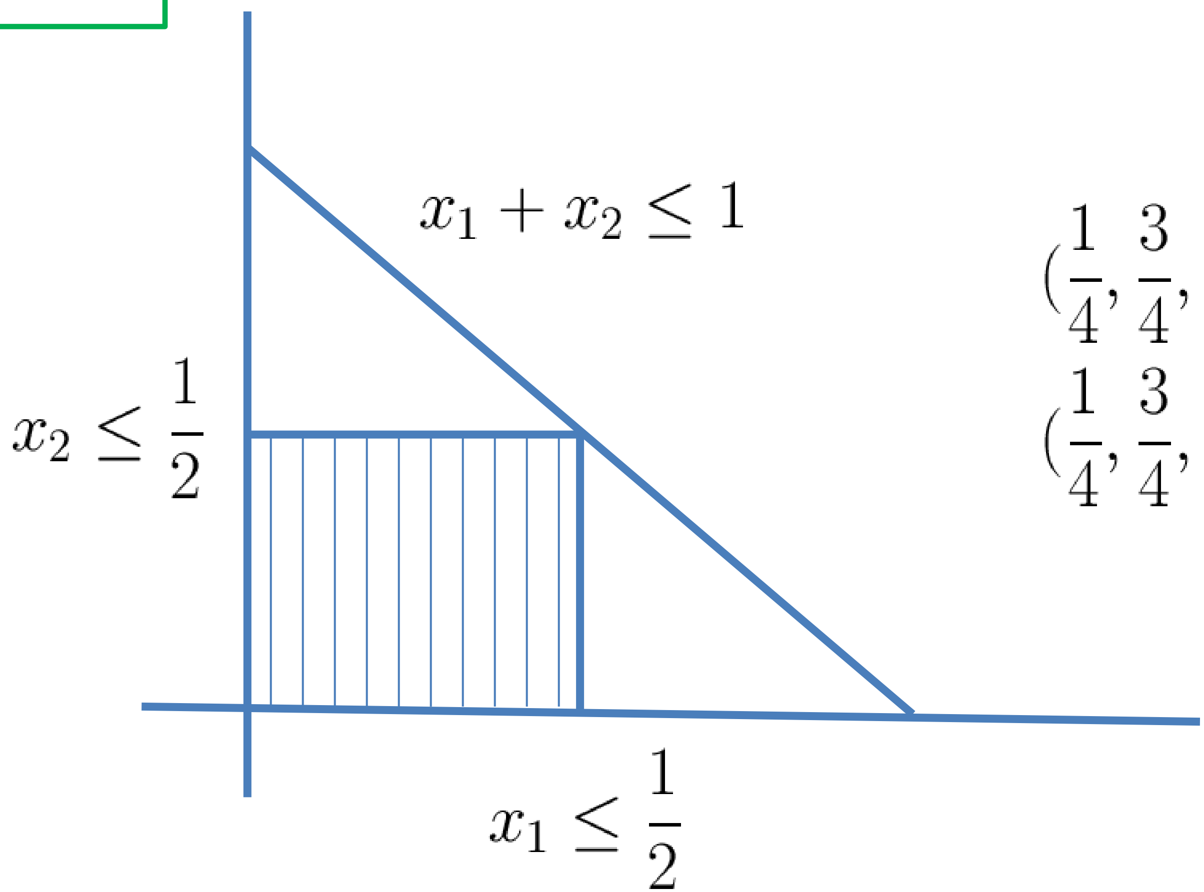
$$\sum_{j=1}^{n} x_{ij} \leq m y_i \qquad (P_1)$$

ALTERNATIVELY $$x_{ij} \leq y_i \qquad j = 1, \ldots, m \qquad (P_2)$$

Is one formulation better than the other?
Better in what sense?

Disaggregation

Consider $y = \frac{1}{2}$



$$(\frac{1}{4}, \frac{3}{4}, \frac{1}{2}) \in P_1$$
$$(\frac{1}{4}, \frac{3}{4}, \frac{1}{2}) \notin P_2$$

Economic Lotsizing

Parameters

c_t : unit production cost in period t

f_t : set up cost in period t

d_t : demand in period t

Formulation

$$\min \sum_{t=1}^{T} c_t x_t + \sum_{t=1}^{T} f_t y_t$$

$$\sum_{s=1}^{t} x_s \geq d_{1t} \text{ for } t = 1, ..., T$$

$$x_t \leq d_{tT} y_t \text{ for } t = 1, ..., T$$

Variables

x_t : production in period t

y_t : whether or not a set up occurs in period t

Economic Lotsizing

c_t : unit production cost in period t

f_t : set up cost in period t

d_t : demand in period t

Formulation

$$\min \sum_{t=1}^{T} \sum_{s=t}^{T} c_t w_{ts} + \sum_{t=1}^{T} f_t y_t$$

$$\sum_{s=1}^{t} w_{st} \geq d_t \text{ for } t = 1, ..., T$$

$$w_{st} \leq d_t y_s \text{ for } t = 1, ..., T, s = 1, ..., t$$

Variables

w_{st} : production in period s for period t

y_t : set up in period t

$$\boxed{\text{Note} \quad x_s = \sum_{t \geq s} w_{st}}$$

Extended formulations

$Q = \{(x,w): (x,w) \in R^n \times R^p\}$ is an extended formulation for a pure integer program with formulation $P \subseteq R^n$ if

$$\mathrm{Proj}_x Q \cap Z^n = P \cap Z^n$$

where the projection of Q onto x is defined to be

$$\mathrm{Proj}_x Q = \{x : (x,w) \in Q, \exists w \in R^p\}$$

How can we compare the formulations in this case?

A: if $\mathrm{Proj}_x Q \subset P$, we say Q is a better formulation than P

Generalized Assignment Problem

Parameters

p_{ij} : profit when assigning task j to machine i

w_{ij} : capacity consumption of task j on machine i

d_i : capacity of machine i

Formulation

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \text{ for } j = 1, \ldots, n$$

$$\sum_{j=1}^{n} w_{ij} x_{ij} \leq d_i \text{ for } i = 1, \ldots, m$$

Variables

x_{ij} : whether or not task j is assigned to machine i

Generalized Assignment Problem

Parameters

$$y_k^i = (y_{1k}^i, y_{2k}^i, \ldots, y_{nk}^i)$$

Formulation

$$\max \sum_{i=1}^{m} \sum_{k=1}^{K_i} \left(\sum_{j=1}^{n} p_{ij} y_{jk}^i\right) \lambda_k^i$$

$$\sum_{i=1}^{m} \sum_{k=1}^{K_i} y_{jk}^i \lambda_k^i = 1 \qquad j = 1, \ldots, n$$

$$\sum_{k=1}^{K_i} \lambda_k^i = 1 \qquad i = 1, \ldots, m$$

Variables $\lambda_k^i \in \{0, 1\} \qquad i = 1, \ldots, m, \; k = 1, \ldots, K_i$

Generalized Assignment Problem

Parameters

$$y_k^i = (y_{1k}^i, y_{2k}^i, \ldots, y_{nk}^i)$$

Satisfy

$$\sum_{j=1}^{n} w_{ij} x_j \leq d_i$$

$$x_j \in \{0, 1\} \qquad j = 1, \ldots, n$$

Generalized Assignment Problem

Variables

Coefficient matrix

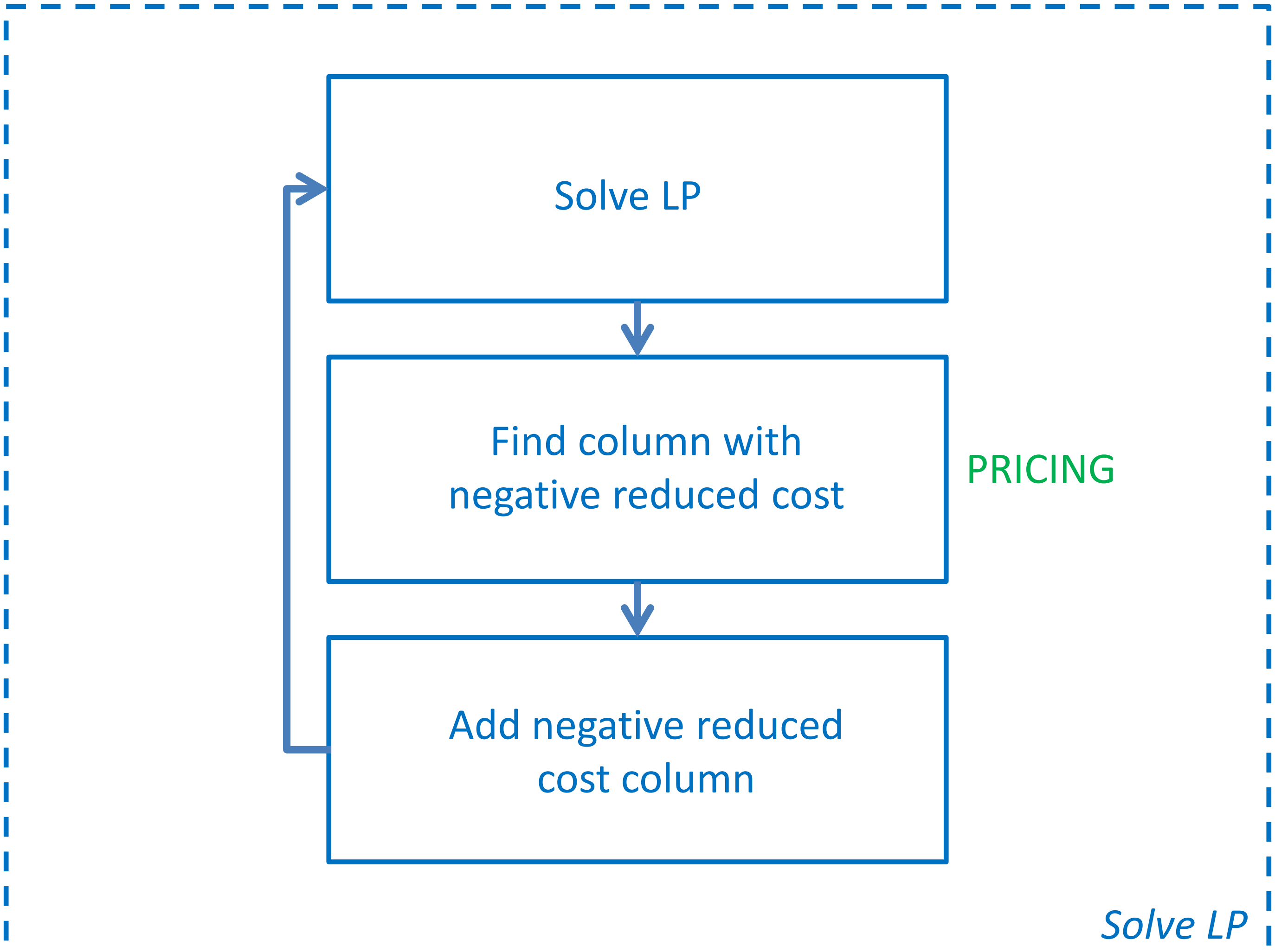
RHS

$$
\begin{array}{cc}
 & \begin{matrix} p_1^1 & p_2^1 & \cdots & p_{K_1}^1 & p_1^2 & p_2^2 & \cdots & p_{K_2}^2 & \cdots & p_1^m & p_2^m & \cdots & p_{K_m}^m \end{matrix} \\
\begin{matrix} 1 \\ 2 \\ \vdots \\ n \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{matrix} &
\begin{pmatrix}
y_{11}^1 & y_{12}^1 & \cdots & y_{1K_1}^1 & y_{11}^2 & y_{12}^2 & \cdots & y_{1K_2}^1 & \cdots & y_{11}^m & y_{12}^m & \cdots & y_{1K_m}^m \\
y_{21}^1 & y_{22}^1 & \cdots & y_{2K_1}^1 & y_{21}^2 & y_{22}^2 & \cdots & y_{2K_2}^1 & \cdots & y_{21}^m & y_{22}^m & \cdots & y_{2K_m}^m \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
y_{n1}^1 & y_{n2}^1 & \ddots & y_{nK_1}^1 & y_{n1}^2 & y_{n2}^2 & \ddots & y_{nK_2}^1 & \cdots & y_{n1}^m & y_{n2}^m & \cdots & y_{nK_m}^m \\
1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 1 & 1 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 1 & \cdots & 1
\end{pmatrix}
\end{array}
\begin{pmatrix} \lambda_1^1 \\ \lambda_2^1 \\ \vdots \\ \lambda_{K_1}^1 \\ \lambda_1^2 \\ \lambda_1^2 \\ \vdots \\ \lambda_{K_2}^2 \\ \vdots \\ \lambda_1^m \\ \lambda_2^m \\ \vdots \\ \lambda_{K_m}^m \end{pmatrix}
=
\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}
$$

How to use column generation formulations

- How to handle exponential number of columns?
- How to find “missing” columns?

Branch-and-Price



Pricing Problem

v_i dual associated with machine *i*

u_j dual associated with task *j*

$$\max_{1 \leq i \leq m} \{z(KP_i) - v_i\}$$

where

$$z(KP_i) = \max \sum_{1 \leq j \leq n} (p_{ij} - u_j)x_j$$

$$\sum_{1 \leq i \leq n} w_{ij}x_j \leq d_i$$

$$x_j \in \{0,1\} \qquad i \in \{1, ..., n\}$$

Branching

- Branching on selection variables
  - Fixing to 1, i.e., selecting a particular schedule for a machine
  - Fixing to 0, i.e., forbidding a particular schedule for a machine

- How to prevent the forbidden schedule to be generated again?

Branching

- Branching on original variables $$x_{ij} = \sum_{k=1}^{K_i} y^i_{jk} \lambda^i_k$$

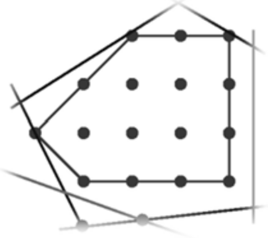
- Fixing to 1
  - Force task *j* in schedule for machine *i*
  - Forbid task *j* in schedule for other machines
- Fixing to 0
  - Forbid task *j* in schedule for machine *i*

What Next?

- Parallel Integer Programming
  - *Topics in Parallel Integer Optimization (Jeff Linderoth, 1998)*
  - *PARINO: PARallel INteger Optimizer (Kalyan Permulla, 1997)*

- Multi-objective Integer Programming
  - *MINTO: Multi-objective INTeger Optimizer*

Restrict and Relax Search



Outline

- Motivation
- Restrict-and-Relax Search
  - Initial restriction
  - Fixing and unfixing
- Computational experiments
  - 0-1 integer programs
  - Multi-commodity fixed charge network flow

Motivation

- Why use restricted integer programs?
  - Desire to find good feasible solutions quickly
    - Crucial for real-life applications
    - Beneficial for many integer programming techniques (e.g., reduced cost fixing)

- Why use restricted integer programs?
  - Integer program of interest very large (e.g., too large to fit in memory)

Motivation

- Success stories
  - Relaxation induced neighborhood search (general)
  - Local branching (general)
  - IP-based neighborhood search (problem-specific)

IP-based Neighborhood Search

Algorithm Neighborhood Search

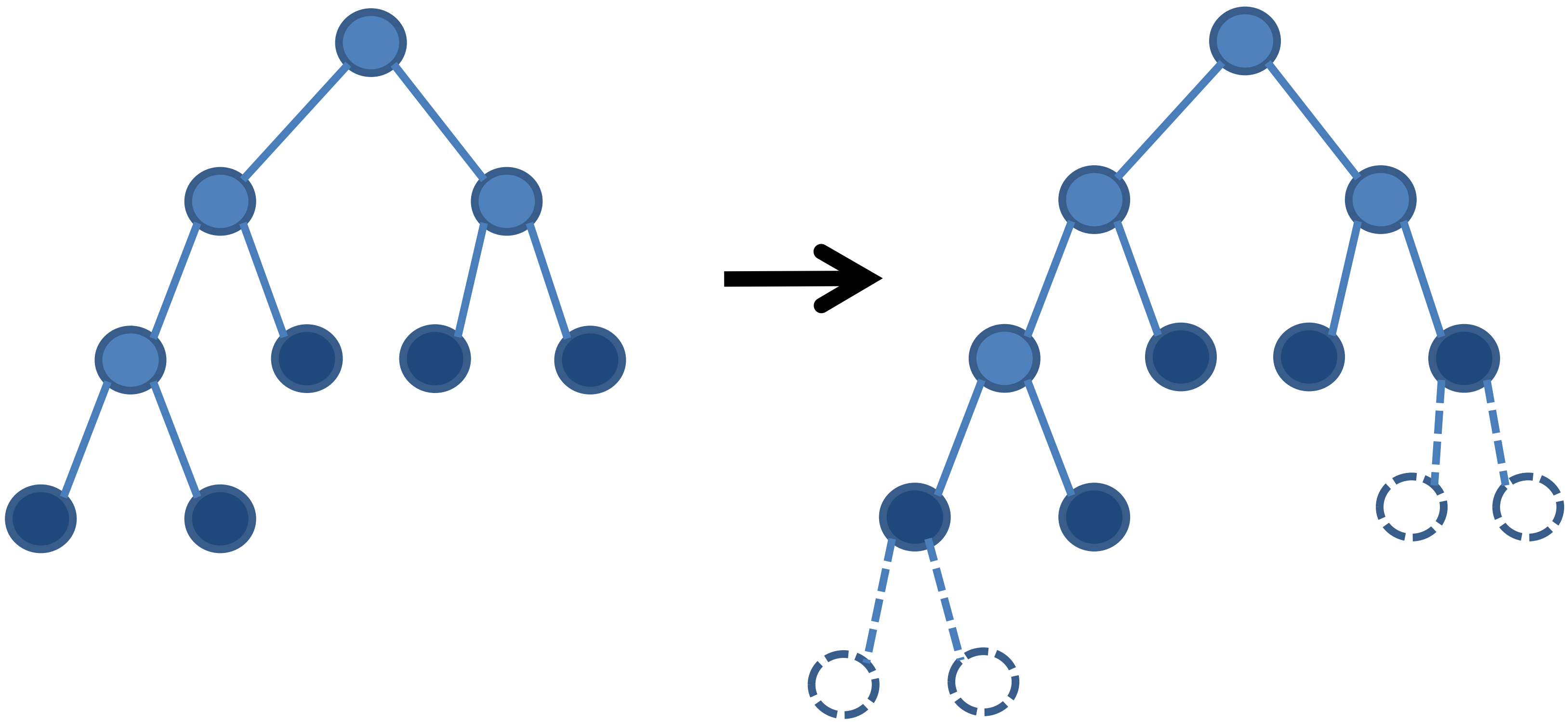
Require: Integer program P
  while continue search **do**
    Choose a subset of variables V
    Fix value of variables in V
    Solve an IP to assign the remaining variables
    if an improved solution is found **then**
      Update global solution
    end if
  end while

IP-based search heuristic

- Key decisions:
  - What variables to fix?
  - What values to fix them to?

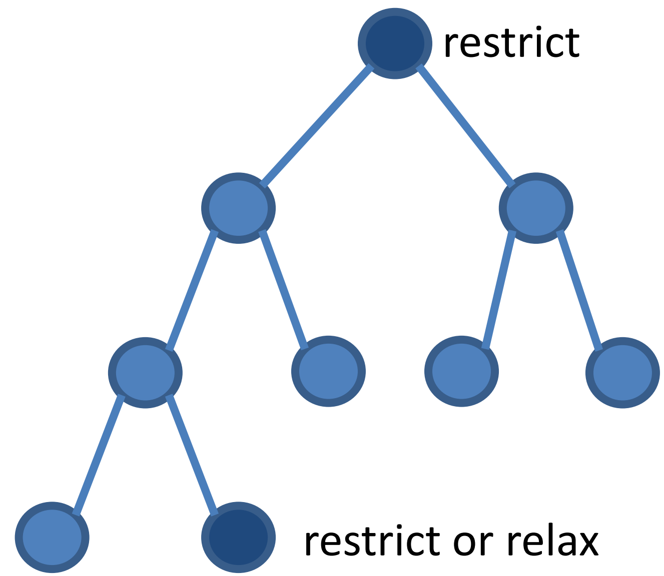
IP-based search heuristic

If we solve a number of related restricted integer programs, can we re-use the search tree?



Restrict-and-Relax Search

- Main ideas:
  - Branch-and-bound algorithm that **always** works on a restricted integer program
  - Branch-and-bound algorithm that uses **local information** to decide whether to relax (unfix variables) or restrict (fix variables)



Restrict-and-Relax Search

- Main ideas:
  - Restrict: Efficiency
  - Relax: Quality

Restrict-and-Relax Search

Original IP

$$
\begin{aligned}
z = \quad & \min cx \\
& Ax = b \\
& x \in \mathbb{B}^r \times \mathbb{R}^{n-r}
\end{aligned}
$$

Restricted IP

$$
\begin{aligned}
z_F = \quad & \min cx \\
& Ax = b \\
& x_i = \bar{x}_i \ , \ i \in F \\
& x \in \mathbb{B}^r \times \mathbb{R}^{n-r}
\end{aligned}
$$

Restricted IP at node of the search tree

$$
\begin{aligned}
v_t = \quad & \min cx \\
& Ax = b \\
& x_i = \bar{x}_i \ , \ i \in F \cup B_t \\
& x \in \mathbb{B}^r \times \mathbb{R}^{n-r}
\end{aligned}
$$

Restrict-and-Relax Search

Restricted IP at node of the search tree

$$
\begin{aligned}
v_t = \ & \min cx \\
& Ax = b \\
& x_i = \bar{x}_i \ , \ i \in F \cup B_t \\
& x \in \mathbb{B}^r \times \mathbb{R}^{n-r}
\end{aligned}
$$

Modified restricted IP at node of the search tree

$$
\begin{aligned}
\bar{v}_t = \ & \min cx \\
& Ax = b \\
& x_i = \bar{x}_i \ , \ i \in \bar{F} \cup B_t \\
& x \in \mathbb{B}^r \times \mathbb{R}^{n-r}
\end{aligned}
$$

$$\boxed{\text{Goal: Choose } \bar{F} \text{ in such a way that } \bar{v}_t < v_t}$$

Restrict-and-Relax Search

- Key decisions
  - How to define the initial restricted integer program?
  - How to determine the variables to fix or unfix?
  - At which nodes in the search tree to relax or restrict?

How to define the initial restricted integer program?

- Based on a known feasible solution
- Based on the solution to the LP relaxation
- Based on the Phase I solution to the LP relaxation

How to define the initial restricted integer program?

Scheme:

- $x_{LP} = 0$ for variable: Score: c
- $x_{LP} = 1$ for variable: Score: -c
- Fix variables from large to small
- Fix at most 90% of variables

How to determine the variables to fix and unfix?

- Fixing variables (LP feasible node):

$$\boxed{\text{if } x_i^* = 0, \text{ then } r_i^* \geq 0 \text{ and if } x_i^* = 1, \text{ then } r_i^* \leq 0}$$

Choose variables to fix in nondecreasing order of absolute value of reduced costs

Fix variables in the current primal solution that are not likely to be part of an optimal solution in the future

- Unfixing variables (LP feasible node):

$$\boxed{\text{if } x_i^* = 0 \text{ and } r_i^* < 0 \text{ or if } x_i^* = 1 \text{ and } r_i^* > 0}$$

Choose variables to unfix in nondecreasing order of absolute value of reduced costs

Unfixing variables in the current primal solution that are likely to result in an optimal solution with lower value in the future

How to determine the variables to fix and unfix?

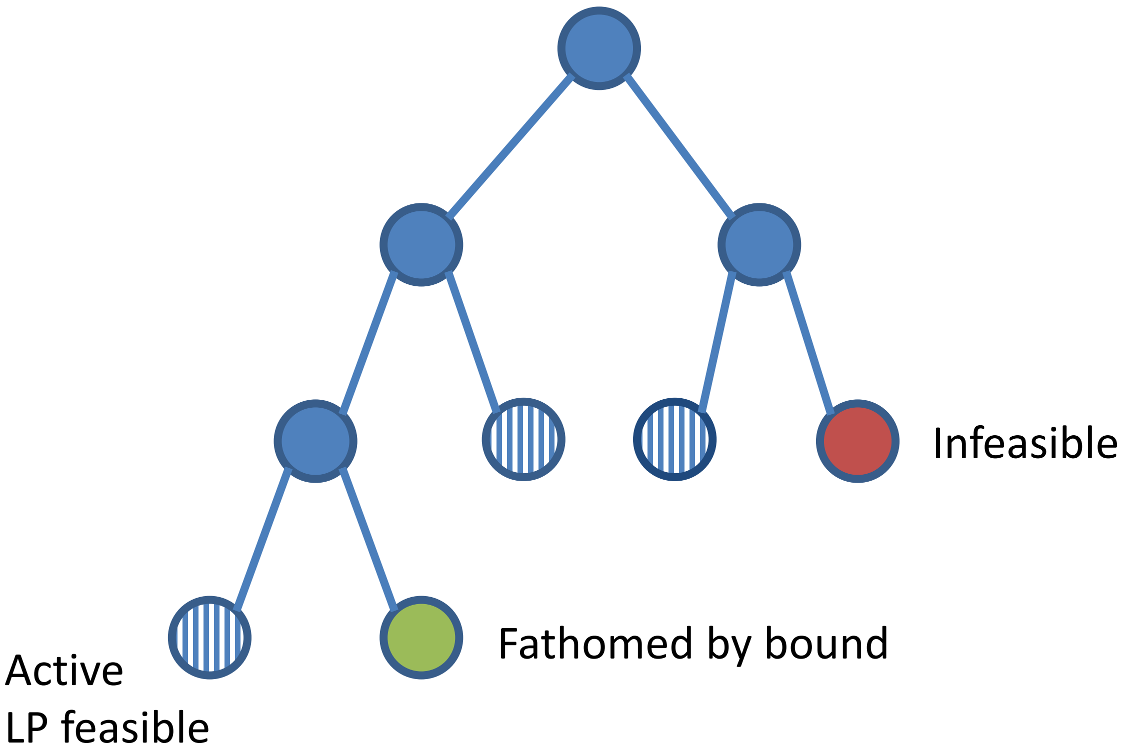
- Implementation - Gradual transition:

$$\min cx$$
$$Ax = b$$
$$x_i = \bar{x}_i \ , \ i \in F_t^j \cup B_t$$
$$x \in \mathbb{B}^r \times \mathbb{R}^{n-r}$$

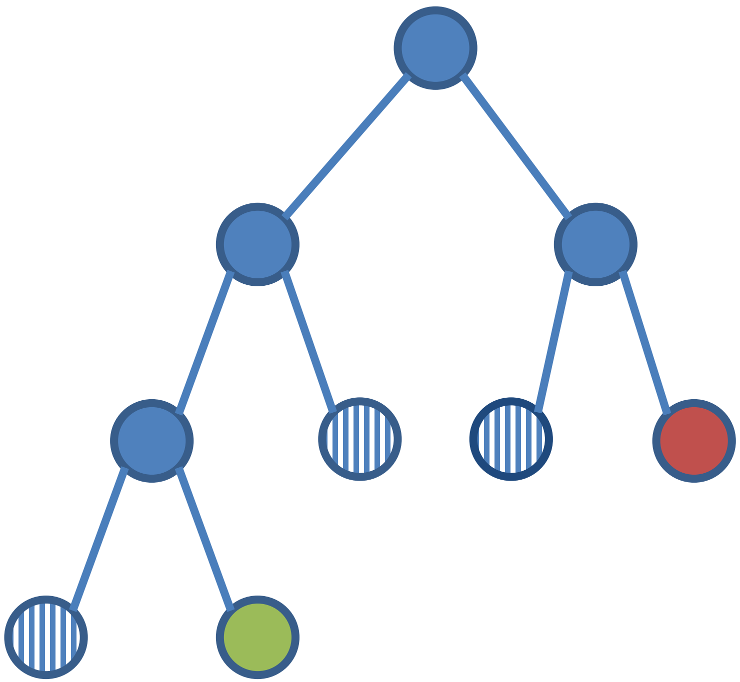
$$F_t^0 = F \text{ and } |F_t^j \setminus F_t^{j-1}| \text{ small}$$

  – Fast linear programming solves
  – Up to date dual information

Unfixing variables

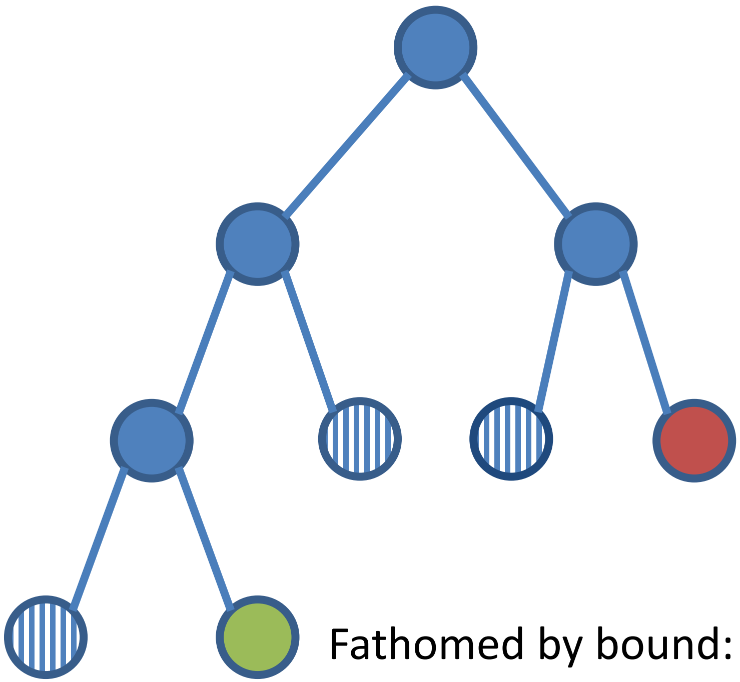


Unfixing variables



Opportunistic relaxing:
Unfix previously fixed variables

Unfixing variables



Infeasible:

(1) Resolve LP with all variables unfixed
(2) Unfix variables that change values

Fathomed by bound:

(1) Remove cut-off
(2) Resolve LP with all variables unfixed
(3) If value < best known, unfix based on reduced costs

Resolve LP with all variables unfixed guarantees that no nodes are discarded that shouldn't

At which nodes in the search tree to relax or restrict?

- Parameters:
  - level-frequency (l-f) : Relax/restrict at node t if node level is a multiple of l-f
  - unfix-ratio (u-r) : At each trial, unfix at most u-r % of the fixed variables
  - fix-ratio (f-r) : At each trial, fix at most f-r % of the free variables
  - node-trial-limit (t-l) : At each node, fix/relax at most t-l times
  - max-depth (max-d) : Fix/unfix only at nodes below tree level max-d
  - min-depth (min-d) : Fix/unfix only at nodes above tree level min-d
  - Pruned-by-bound (p-b) : If enabled, fix/unfix at nodes pruned by bound regardless of node level
  - Pruned-by-infeasibility (p-i) : If enabled, fix/unfix at nodes pruned by infeasibility regardless of node level

At which nodes in the search tree to relax or restrict?

- Default values:
  - level-frequency (l-f) : 4
  - unfix-ratio (u-r) : 5%
  - fix-ratio (f-r) : 2.5%
  - node-trial-limit (t-l) : 5
  - max-depth (max-d) : ∞
  - min-depth (min-d) : 0
  - Pruned-by-bound (p-b) : enabled
  - Pruned-by-infeasibility (p-i) : enabled

Computational Study

- Instances:
  - COR@L
  - MIPLIB

- Restrict-and-Relax
  - Initial restricted IP: based on LP relaxation
  - Default settings for parameters
  - Time limit: 500 seconds

- Implementation: SYMPHONY + CLP

Computational Study

- Default solver: Original IP
- Default solver: Restricted IP
- Restrict-and-relax Search

Results

	Original IP	Restricted IP
RR $<$	51	49
RR $=$	13	5
RR $>$	20	15
RR feas	13	28
RR no feas	1	2

By varying control parameters we can obtained improved solutions for all instances!

Results (sample)

	Original IP	Restricted IP	Restrict-and-Relax Search	Optimal
neos-693347	360	-	241	234
neos808444	-	-	0	0
m100n500k4r1	-22	-22	-24	-25

Results (sample)

	%fixed	#nodes	#unfix	avg.	#fix	avg.	#solutions
neos-693347	0.74	593	266	18	243	27	1
neos808444	0.9	633	129	24	1	1	1
m100n500k4r1	0.7	38075	11095	6	1620	12	4

Multi-commodity Fixed Charge Network Flow

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}(d^k x_{ij}^k) + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

Variable flow cost (>= 0)

Fixed cost of installing arc (>= 0)

Commodity flow balance

$$\sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(j,i) \in A} x_{j,i}^k = \delta_i^k \quad \forall i \in N, \ \forall k \in K,$$

Arc capacity and coupling

$$\sum_{k \in K} d^k x_{ij}^k \leq u_{ij} y_{ij} \ \ \forall (i,j) \in A,$$

$$y_{ij} \in \{0,1\} \ \ \forall (i,j) \in A.$$ Do we install arc (i,j)?

$$x_{ij}^k \in \{0,1\} \quad \forall k \in K, \ \forall (i,j) \in A.$$ Does commodity k flow on arc (i,j)?

Computational Study

- Instances
  - Notation: T - #nodes(100x) - #arcs(1000x) - #commodities
  - Smallest (T-5-3-50)
    - 150,000 variables, 180,000 constraints, 750,000 nonzeroes
  - Largest (T-5-3-200)
    - 600,000 variables, 700,000 constraints, 3,000,000 nonzeroes
- Restrict-and-relax settings
  - Initial restriction:
    - **Phase I of simplex algorithm**, fix up to 90% of variables
  - Parameters:
    - unfix ratio: 6%, fix ratio: 5%
  - Time limit: 2 hours

Results

	CPLEX		RR		
	z	LP solved?	z	# Sols	% Gap
T-5-2-150	∞	no	9,660,795	15	-
T-5-2-200	∞	no	8,452,576	17	-
T-5-2.5-100	6,238,900	yes	3,963,321	24	24.01
T-5-2.5-150	∞	no	8,921,333	11	-
T-5-2.5-200	∞	no	15,439,381	11	-
T-5-3-50	5,014,455	yes	2,448,390	35	12.37
T-5-3-75	∞	yes	4,698,771	17	24.64
T-5-3-100	∞	no	7,777,745	14	-
T-5-3-125	∞	no	6,169,464	17	-
T-5-3-150	∞	no	8,081,616	16	-
T-5-3-200	∞	no	14,691,367	9	-