Bound set based branch-and-cut algorithms for bi-objective combinatorial optimization problems

Sune Lauth Gadegaard[1]
Matthias Ehrgott [2] and Lars Relund Nielsen[1]

[1]Department of Economics and Business Economics, Aarhus University

[2]Department of Management Science, Lancaster University

June 24, 2016

CORAL
AARHUS UNIVERSITY

Outline

- Notation and definitions
- Branch and cut
- Bound sets
- Cutting plane algorithm
- Pruning
- Bound set update
- Branching
- Conclusions

Notation

We want to solve a BOCO of the following form

$$\begin{aligned} &\min Cx \\ \text{s.t.: } &Ax \leq b \\ &x \in \{0,1\}^n \end{aligned}$$

For simplicity

- $\mathcal{X} := \{x \in \{0,1\}^n \ : \ Ax \leq b\}$.
- $\underline{\mathcal{X}} := \{x \in [0,1]^n : Ax \leq b\} \leftarrow$ The LP relaxation

Orderings

Definition

For $z^1, z^2 \in \mathbb{R}^2$ we say that

$$z^1 \leqq z^2 \Leftrightarrow z_i^1 \leq z_i^2, \text{ for } i = 1, 2$$
$$z^1 \leq z^2 \Leftrightarrow z^1 \leqq z^2 \text{ and } z^1 \neq z^2$$
$$z^1 < z^2 \Leftrightarrow z_i^1 < z_i^2, \text{ for } i = 1, 2$$

Cones

Definition

By $\mathbb{R}^2_{\geqq}$ we define the set

$$\mathbb{R}^2_{\geqq} = \{z \in \mathbb{R}^2 \ : \ z \geqq 0\}$$

Similarly for $R^2_{\geq}$ and $\mathbb{R}^2_{>}$.

Efficiency

Definition

$\hat{x} \in \mathcal{X}$ is called *efficient* if there does not exist another $x \in \mathcal{X}$ such that

$$Cx \leq C\hat{x}$$

The corresponding outcome vector, $\hat{z} := C\hat{x}$, is called *non–dominated*.

Notation

$\mathcal{X}_E$ set of all efficient solutions.

$\mathcal{Z}_N$ $C\mathcal{X}_E = \{z \in \mathbb{R}^2 \; : \; z = Cx, x \in \mathcal{X}_E\}$

$\bar{\mathcal{Z}}$ upper bound set, $\bar{\mathcal{Z}} \subseteq C\mathcal{X}$.

L lower bound set.

η an active branching node.

$\mathcal{X}(\eta)$ feasible set of node η.

$\underline{\mathcal{X}}(\eta)$ LP relaxed version of $\mathcal{X}(\eta)$.

Branch & cut

Single objective branch and bound:

1. Pick an active node
2. If node is infeasible $\rightarrow$ prune it.
3. Add cuts if necessary
4. Solve relaxation.
   - If solution is integral, update incumbent and prune.
   - If subproblem contains no improving solutions, prune. Else branch.

Bi–objective branch and bound:

Branch & cut

Single objective branch and bound:

1. Pick an active node
2. If node is infeasible $\rightarrow$ prune it.
3. Add cuts if necessary
4. Solve relaxation.
   - If solution is integral, update incumbent and prune.
   - If subproblem contains no improving solutions, prune. Else branch.

Bi–objective branch and bound:

1. Pick an active node

Branch & cut

Single objective branch and bound:

1. Pick an active node
2. If node is infeasible $\rightarrow$ prune it.
3. Add cuts if necessary
4. Solve relaxation.
   - If solution is integral, update incumbent and prune.
   - If subproblem contains no improving solutions, prune. Else branch.

Bi–objective branch and bound:

1. Pick an active node
2. If node is infeasible $\rightarrow$ prune it.

Branch & cut

Single objective branch and bound:

1. Pick an active node
2. If node is infeasible → prune it.
3. Add cuts if necessary
4. Solve relaxation.
   - If solution is integral, update incumbent and prune.
   - If subproblem contains no improving solutions, prune. Else branch.

Bi–objective branch and bound:

1. Pick an active node
2. If node is infeasible → prune it.
3. Add cuts if necessary

Branch & cut

Single objective branch and bound:

1. Pick an active node
2. If node is infeasible → prune it.
3. Add cuts if necessary
4. Solve relaxation.
   - If solution is integral, update incumbent and prune.
   - If subproblem contains no improving solutions, prune. Else branch.

Bi–objective branch and bound:

1. Pick an active node
2. If node is infeasible → prune it.
3. Add cuts if necessary
4. Solve (bi–objective) relaxation
   - If solution(s) integral, update incumbent *set* and *branch*.

Branch & cut

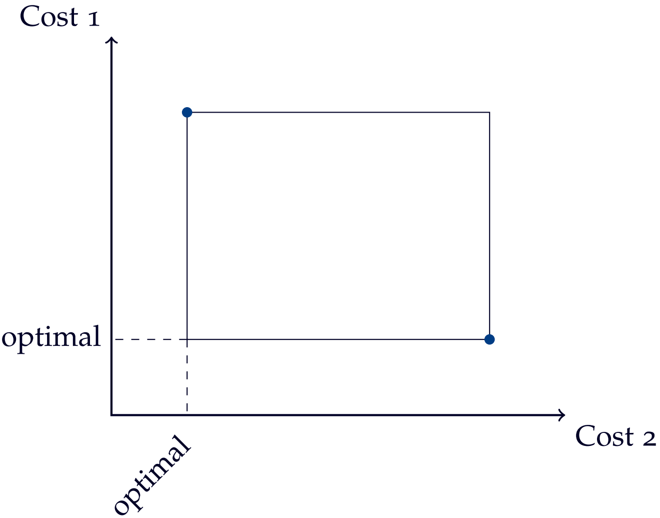
Single objective branch and bound:

1. Pick an active node
2. If node is infeasible → prune it.
3. Add cuts if necessary
4. Solve relaxation.
   - If solution is integral, update incumbent and prune.
   - If subproblem contains no improving solutions, prune. Else branch.

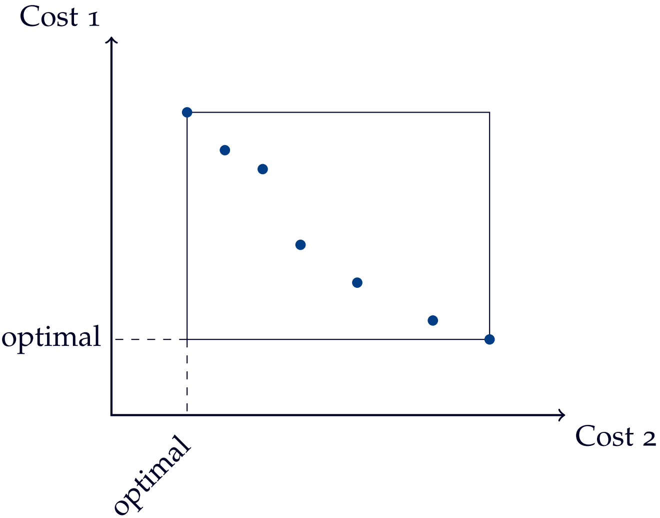
Bi–objective branch and bound:

1. Pick an active node
2. If node is infeasible → prune it.
3. Add cuts if necessary
4. Solve (bi–objective) relaxation
   - If solution(s) integral, update incumbent *set* and *branch*.
   - If subproblem contains no *efficient solutions*, prune. Else branch.

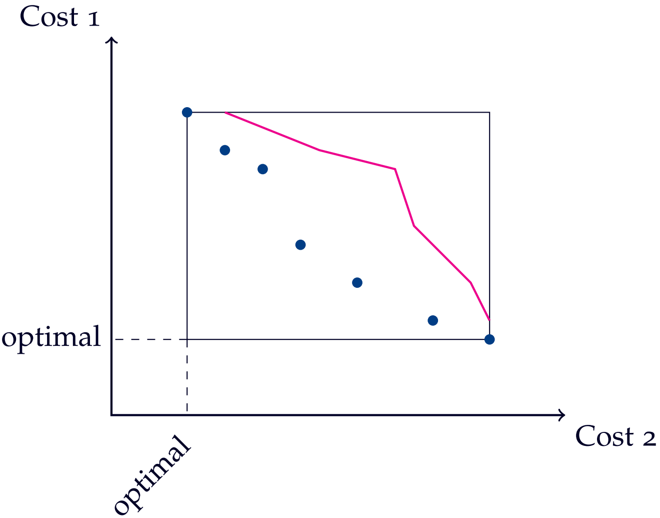
Bound sets



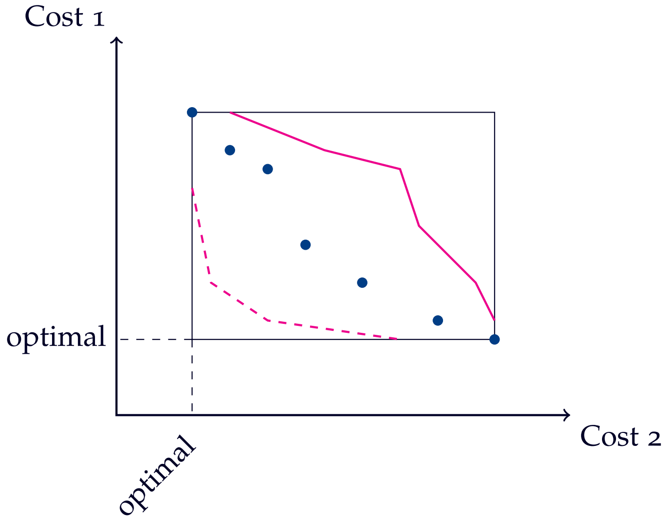
Bound sets



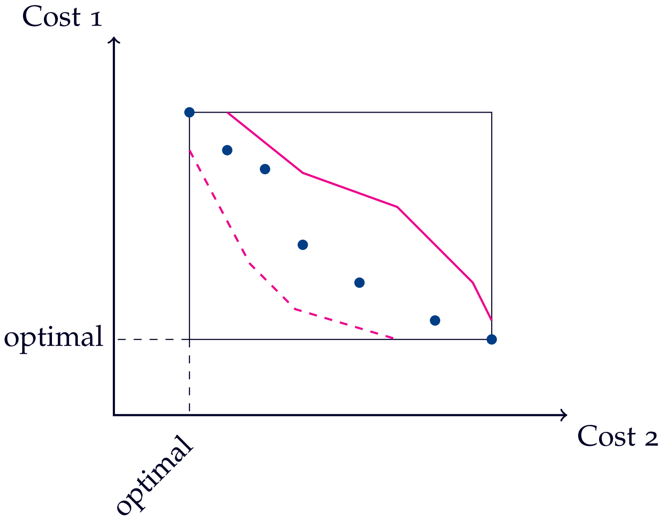
Bound sets



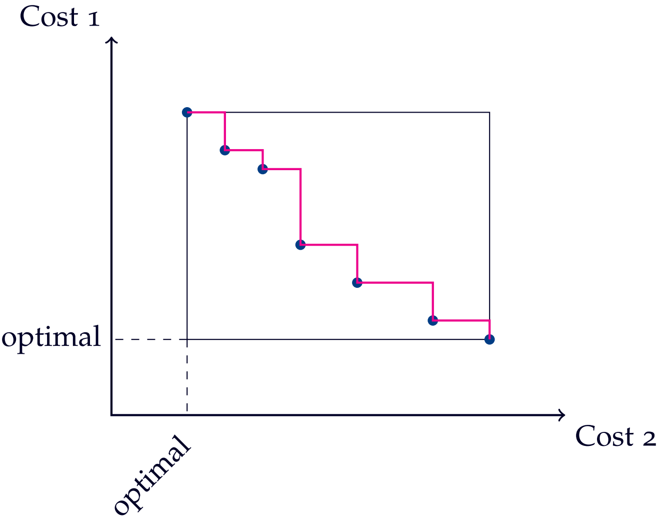
Bound sets



Bound sets



Bound sets



Bound sets

Lower bound set

- Lower bound on the non–dominated frontier

Bound sets

Lower bound set

- Lower bound on the non–dominated frontier
- A set of points ensuring, that all solutions lie *above*!

Bound sets

Lower bound set

- Lower bound on the non–dominated frontier
- A set of points ensuring, that all solutions lie *above*!
- Relax integrality constraints

Bound sets

Lower bound set

- Lower bound on the non–dominated frontier
- A set of points ensuring, that all solutions lie *above*!
- Relax integrality constraints

Upper bound set

- Upper bound on the non–dominated frontier

Bound sets

Lower bound set

- Lower bound on the non–dominated frontier
- A set of points ensuring, that all solutions lie *above*!
- Relax integrality constraints

Upper bound set

- Upper bound on the non–dominated frontier
- We only need to look for Pareto solutions *below* the upper bound set

Bound sets

Lower bound set

- Lower bound on the non–dominated frontier
- A set of points ensuring, that all solutions lie *above*!
- Relax integrality constraints

Upper bound set

- Upper bound on the non–dominated frontier
- We only need to look for Pareto solutions *below* the upper bound set
- Outcome vectors of feasible solutions

Bound sets

Lower bound set

- Lower bound on the non–dominated frontier
- A set of points ensuring, that all solutions lie *above*!
- Relax integrality constraints

Upper bound set

- Upper bound on the non–dominated frontier
- We only need to look for Pareto solutions *below* the upper bound set
- Outcome vectors of feasible solutions

We only need to search *between* the upper and lower bound sets

Cutting planes

Single objective

- Strengthen the lower bound
- Approximate the integer hull of solutions in the direction of the objective function

Multi objective

- Strengthen the lower bound *set*
- Approximate the integer hull of solutions in the direction of the objective function*s*

Modified NISE–algorithm

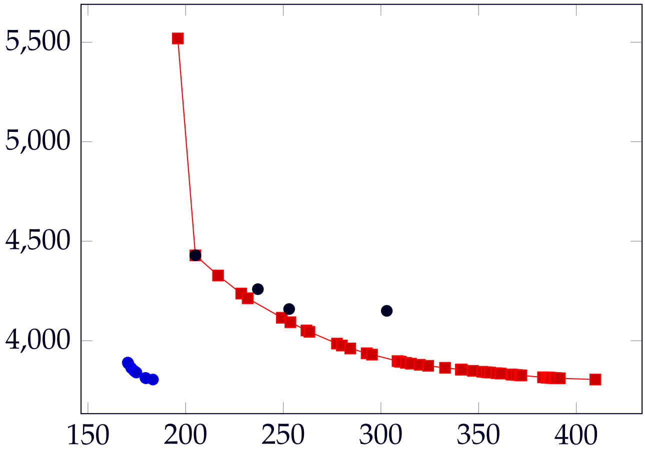
The NISE algorithm works by solving a series of problems of the form

$$\begin{aligned} \min\ & \lambda c^1 x + (1-\lambda) c^2 x \\ \text{s.t.: } & Ax \leqq b \\ & 0 \leqq x \leqq 1 \end{aligned}$$

Modified NISE–algorithm

1. Update λ according to the NISE scheme
2. Solve the weighted sum LP and obtain optimal solution x^*
3. If a cut $\pi^T x \leqq \pi_0$ exists add it, and go back to 2.
4. Else record $c^1 x^*$ and $c^2 x^*$, and go to 1.

An example of cut effect



Bound fathoming

- We assume $\bar{\mathcal{Z}}$ is initialized with the lexicographic minimizers.

Bound fathoming

- We assume $\bar{\mathcal{Z}}$ is initialized with the lexicographic minimizers.

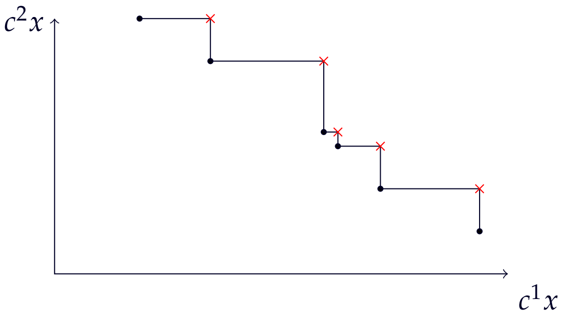
Theorem

A subproblem corresponding to branching node η contains no efficient solutions, if the set

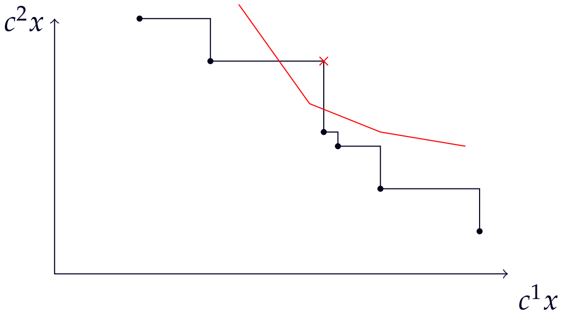
$$L(\eta) + \mathbb{R}^2_{\geqq}$$

contains no local Nadir points.

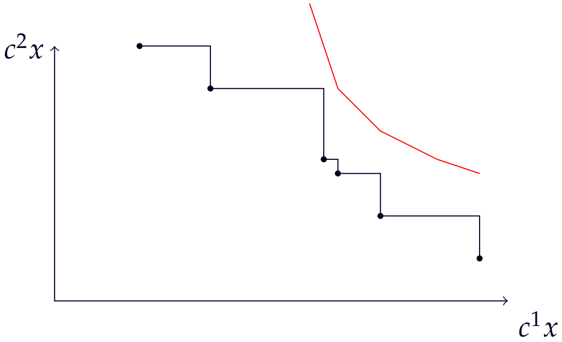
Bound fathoming – An illustration



Bound fathoming – An illustration



Bound fathoming – An illustration



Bound fathoming: Explicit PIP–test

How to check this?

Bound fathoming: Explicit PIP–test

How to check this?

- Solve the Bi–objective LP-relaxation of the node using NISE algorithm

Bound fathoming: Explicit PIP–test

How to check this?

- Solve the Bi–objective LP-relaxation of the node using NISE algorithm
  - Get the extreme points of the frontier

Bound fathoming: Explicit PIP–test

How to check this?

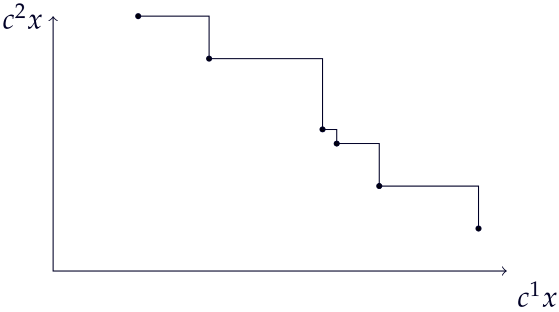
- Solve the Bi–objective LP-relaxation of the node using NISE algorithm
  - Get the extreme points of the frontier
- Intersect with the bounding box from lex-min solutions

Bound fathoming: Explicit PIP–test

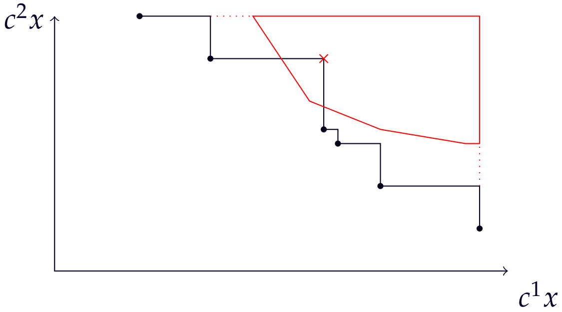
How to check this?

- Solve the Bi–objective LP-relaxation of the node using NISE algorithm
  - Get the extreme points of the frontier
- Intersect with the bounding box from lex-min solutions
- Use a PIP algorithm

Bound fathoming: Explicit PIP–test



Bound fathoming: Explicit PIP–test



Bound fathoming: Explicit LP–test

Perform the test using linear programming.

- z^N is a local Nadir point
- $\{\underline{z}^1, \ldots, \underline{z}^L\}$ extreme points of $(C\underline{\mathcal{X}}(\eta))_N$.

$$\min s_1 + s_2 \tag{1}$$

$$\text{s.t.:} \sum_{l=1}^{L} \underline{z}_1^l \lambda_l - s_1 \leq z_1^N, \tag{2}$$

$$\sum_{l=1}^{L} \underline{z}_2^l \lambda_l - s_2 \leq z_2^N, \tag{3}$$

$$\sum_{l=1}^{L} \lambda_l = 1. \qquad s_1, s_2 \geq 0. \tag{4}$$

Implicit LP–test

Implicit test using linear programming

- z^N is a local Nadir point
- $\underline{\mathcal{X}}$ LP relaxation

$$\begin{aligned} \min\ & s_1 + s_2 \\ \text{s.t.: } & c^1 x \leqq z_1^N, \\ & c^2 x \leqq z_2^N, \\ & x \in \underline{\mathcal{X}}. \\ & s_1, s_2 \geq 0. \end{aligned}$$

Nodes are *not* dominated

Simple test to check if a node is *not* dominated:

Theorem

A branching node η *cannot be pruned by previous theorem, if there exits* $\lambda \in (0,1)$ *and* $z \in \mathcal{N}(\bar{\mathcal{Z}})$ *such that*

$$Cx^{\lambda} \leq z$$

where $x^{\lambda} \in \arg\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}(\eta)\}$.

Bound set updating

- Solve scalarized LP–relaxation

$$\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}(\eta)\}$$

before solving bi–objective LP–relaxation.

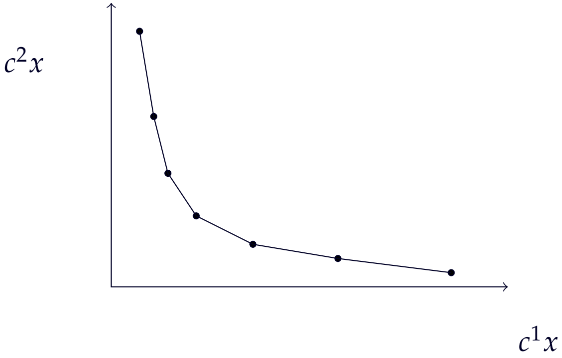
Bound set updating

- Solve scalarized LP–relaxation

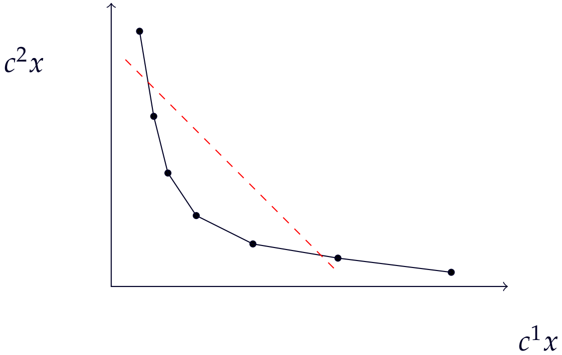
$$\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}(\eta)\}$$

  before solving bi–objective LP–relaxation.
- Inherit lower bound set of parent node, and update!

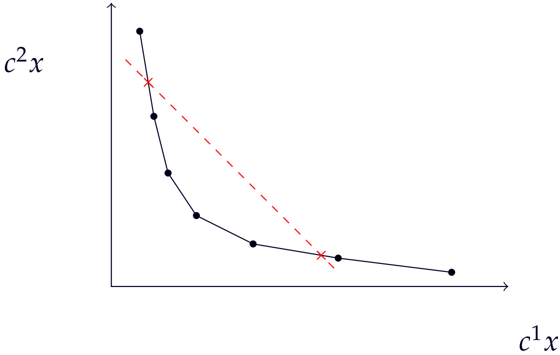
Bound set updating – Illustration



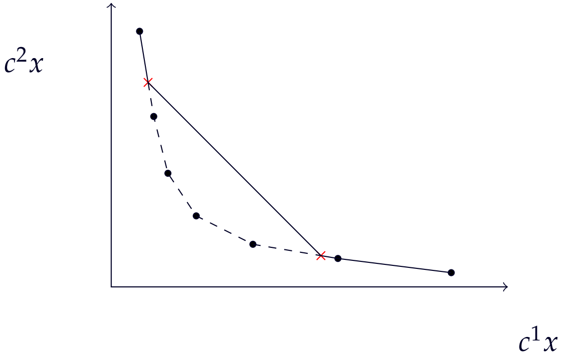
Bound set updating – Illustration



Bound set updating – Illustration

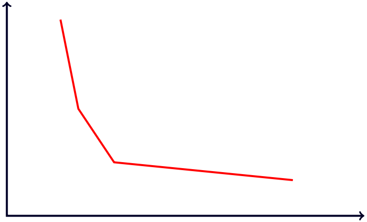


Bound set updating – Illustration



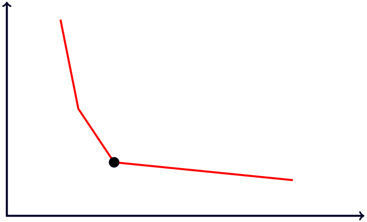
When should we update/resolve

- If we branch in objective space, child nodes should be resolved (more on branching in a minute).
- If the lower bound set from scalarization strictly dominates that of parent node, then we resolve



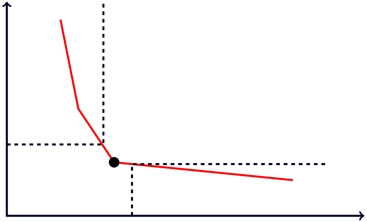
When should we update/resolve

- If we branch in objective space, child nodes should be resolved (more on branching in a minute).
- If the lower bound set from scalarization strictly dominates that of parent node, then we resolve



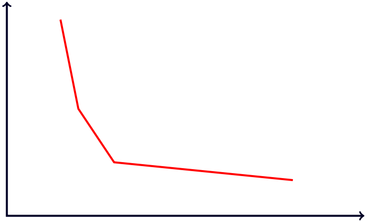
When should we update/resolve

- If we branch in objective space, child nodes should be resolved (more on branching in a minute).
- If the lower bound set from scalarization strictly dominates that of parent node, then we resolve



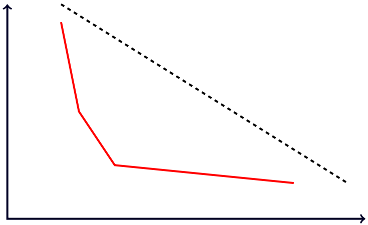
When should we update/resolve

- If we branch in objective space, child nodes should be resolved (more on branching in a minute).
- If the lower bound set from scalarization strictly dominates that of parent node, then we resolve



When should we update/resolve

- If we branch in objective space, child nodes should be resolved (more on branching in a minute).
- If the lower bound set from scalarization strictly dominates that of parent node, then we resolve



When should we branch?

Single objective branch and bound:

- Pick an active node
- If node is infeasible → prune it.
- Solve relaxation.
  - If solution is integral, update incumbent and prune.
  - If lower bound is worse than incumbent, prune. Else branch.

Bi–objective branch and bound

- Pick an active node
- If node is infeasible → prune it.
- Solve (bi–objective) relaxation
  - If solution(s) integral, update incumbent *set* and **branch**.
  - If subproblem contains no efficient solutions, prune. Else **branch**.

Integer branching – No–good inequalities

Let $\bar{x} \in \mathcal{X}$. Create one! child node with the inequality

$$\sum_{i:\bar{x}_i=1} (1 - x_i) + \sum_{i:\bar{x}_i=0} x_i \geq 1$$

Integer branching – No–good inequalities

Let $\bar{x} \in \mathcal{X}$. Create one! child node with the inequality

$$\sum_{i:\bar{x}_i=1} (1 - x_i) + \sum_{i:\bar{x}_i=0} x_i \geq 1$$

- Does only remove solution in decision space!

Integer branching – No–good inequalities

Let $\bar{x} \in \mathcal{X}$. Create one! child node with the inequality

$$\sum_{i:\bar{x}_i=1} (1 - x_i) + \sum_{i:\bar{x}_i=0} x_i \geq 1$$

- Does only remove solution in decision space!
- Might be many *equivalent solutions*

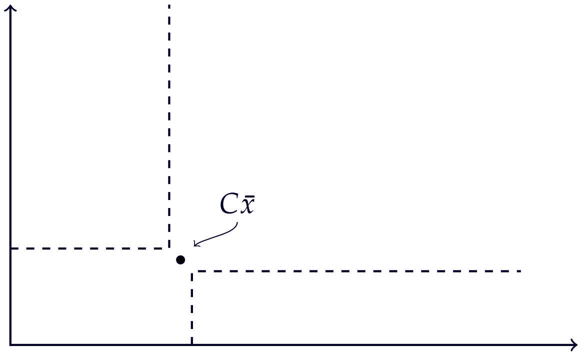
Integer branching – No–good inequalities

Let $\bar{x} \in \mathcal{X}$. Create one! child node with the inequality

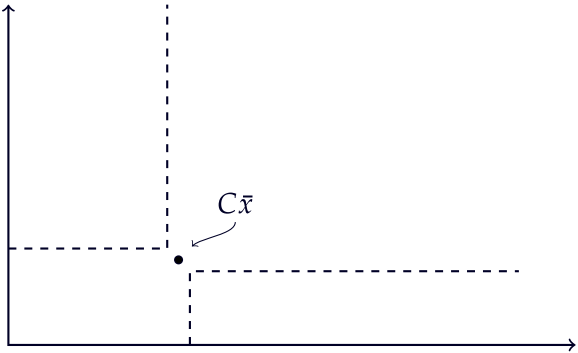
$$\sum_{i:\bar{x}_i=1} (1 - x_i) + \sum_{i:\bar{x}_i=0} x_i \geq 1$$

- Does only remove solution in decision space!
- Might be many *equivalent solutions*
- Use Pareto branching!

Integer branching - No–good in objective space

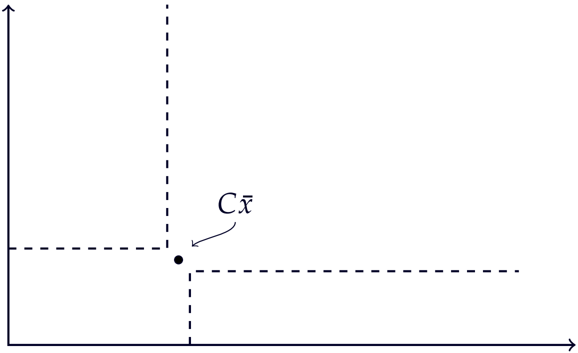


Integer branching - No–good in objective space



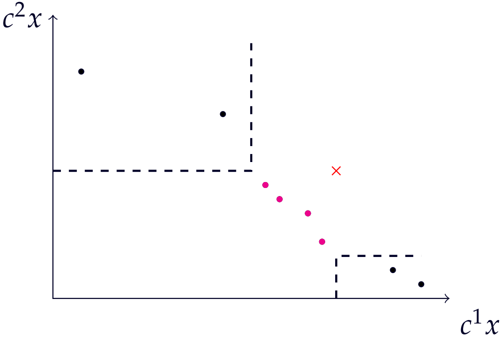
- Create two new child nodes, one mapping to the north west of $C\bar{x}$ and one to the south east.

Integer branching - No–good in objective space



- Create two new child nodes, one mapping to the north west of $C\bar{x}$ and one to the south east.
- Generalize to *Pareto branching*

Pareto branching – Illustration



Generalized Pareto branching

- Let η be an active branching node, and let $L(\eta)$ be a lower bound set of the node.

Generalized Pareto branching

- Let η be an active branching node, and let $L(\eta)$ be a lower bound set of the node.
- Let $\mathcal{N}^L(\eta)$ be a set of local Nadir points where

$$z^N \in L(\eta) + \mathbb{R}^2_{\geqq}.$$

Generalized Pareto branching

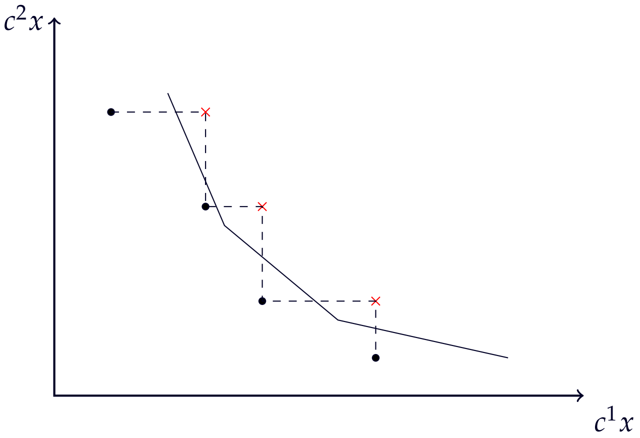
- Let η be an active branching node, and let $L(\eta)$ be a lower bound set of the node.
- Let $\mathcal{N}^L(\eta)$ be a set of local Nadir points where

$$z^N \in L(\eta) + \mathbb{R}^2_{\geqq}.$$

- All non–dominated out comes in the sub–problem η maps to

$$\bigcup_{z \in \mathcal{N}^L(\eta)} \left(\{z\} - \mathbb{R}^2_{\geqq} \right)$$

Extended Pareto branching – Illustration



Results

- Tested different ways of comparing lower and upper bound sets

Results

- Tested different ways of comparing lower and upper bound sets
  1. When stating the lower bound sets explicitly, LP based test worse than point–in–polytope test.

Results

- Tested different ways of comparing lower and upper bound sets
  1. When stating the lower bound sets explicitly, LP based test worse than point–in–polytope test.
  2. When stating the lower bound sets implicitly, extended Pareto branching is not improving the performance

Results

- Tested if a bi–objective approach to cutting planes works

Results

- Tested if a bi–objective approach to cutting planes works
  - It does! The algorithm becomes much more robust and also faster.

Results

- Tested if a bi–objective approach to cutting planes works
  - It does! The algorithm becomes much more robust and also faster.
- Tested an updating strategy of the lower bound set
  - Works very well. Lower bounds are worse, but we can check many more subproblems.

Results

- Compared with a two phase method

Results

- Compared with a two phase method
  1. Ranking based two phase method works very bad on our problems

Results

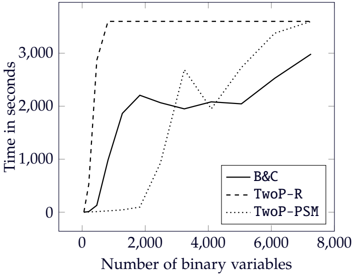
- Compared with a two phase method
  1. Ranking based two phase method works very bad on our problems
  2. PSM based two phase method works better, and even best on smaller problems

Results

- Compared with a two phase method
  1. Ranking based two phase method works very bad on our problems
  2. PSM based two phase method works better, and even best on smaller problems
  3. Our best algorithm, outperforms two phase methods on larger problems

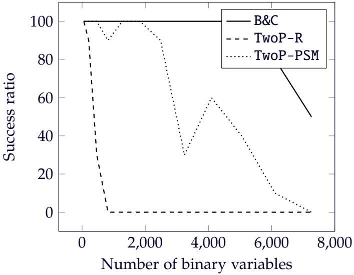
Results

- Compared with a two phase method
  1. Ranking based two phase method works very bad on our problems
  2. PSM based two phase method works better, and even best on smaller problems
  3. Our best algorithm, outperforms two phase methods on larger problems



Results

- Compared with a two phase method
  1. Ranking based two phase method works very bad on our problems
  2. PSM based two phase method works better, and even best on smaller problems
  3. Our best algorithm, outperforms two phase methods on larger problems



Questions
?