Decomposition Methods for Integer Linear Programming

Matthew Galati[1] Ted Ralphs[2]

[1]SAS Institute, Advanced Analytics, Operations Research R & D

[2]COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University

Ph.D. Defense
Industrial and Systems Engineering
Lehigh University, Bethlehem, PA

The Decomposition Principle in Integer Programming

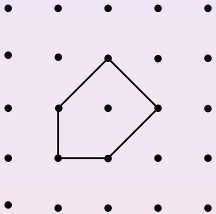
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{D}} = \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}$$

$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$



$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$

Assumptions:

- $\mathrm{OPT}(\mathcal{P}, c)$ and $\mathrm{SEP}(\mathcal{P}, x)$ are *"hard"*
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

The Decomposition Principle in Integer Programming

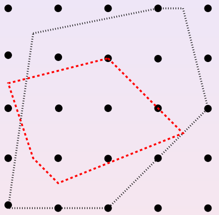
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{D}} = \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}$$

$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$



$\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

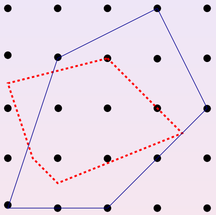
Assumptions:

- $\mathrm{OPT}(\mathcal{P}, c)$ and $\mathrm{SEP}(\mathcal{P}, x)$ are *"hard"*
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

The Decomposition Principle in Integer Programming

Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$
\begin{aligned}
z_{\mathrm{IP}} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\} \\
z_{\mathrm{LP}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\} \\
z_{\mathrm{D}} &= \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}
\end{aligned}
$$



$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$

$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Assumptions:

- $\mathrm{OPT}(\mathcal{P}, c)$ and $\mathrm{SEP}(\mathcal{P}, x)$ are *"hard"*
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

The Decomposition Principle in Integer Programming

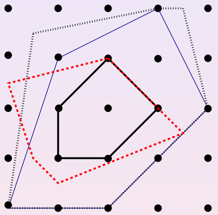
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{D}} = \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}$$

$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$



- $\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$
- $\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$
- $\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \geq b'\}$
- $\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Assumptions:

- $\mathrm{OPT}(\mathcal{P}, c)$ and $\mathrm{SEP}(\mathcal{P}, x)$ are *"hard"*
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

The Decomposition Principle in Integer Programming

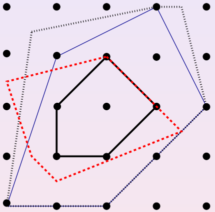
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

$$z_{\mathrm{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\}$$

$$z_{\mathrm{D}} = \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}$$

$$z_{\mathrm{IP}} \geq z_{\mathrm{D}} \geq z_{\mathrm{LP}}$$



$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$

$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$

$\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

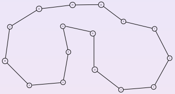
Assumptions:

- $\mathrm{OPT}(\mathcal{P}, c)$ and $\mathrm{SEP}(\mathcal{P}, x)$ are *"hard"*
- $\mathrm{OPT}(\mathcal{P}', c)$ and $\mathrm{SEP}(\mathcal{P}', x)$ are *"easy"*
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size)
- $\mathcal{P}'$ must be represented implicitly (description has exponential size)

Example - Traveling Salesman Problem (TSP)

Traveling Salesman Problem Formulation

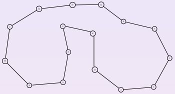
$$
\begin{array}{lll}
x(\delta(\{u\})) & = & 2 \qquad \forall u \in V \\
x(E(S)) & \leq & |S| - 1 \qquad \forall S \subset V,\ 3 \leq |S| \leq |V| - 1 \\
x_e \in \{0,1\} & & \qquad \forall e \in E
\end{array}
$$



Example - Traveling Salesman Problem (TSP)

Traveling Salesman Problem Formulation

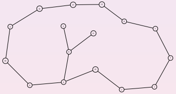
$$\begin{array}{lcll} x(\delta(\{u\})) & = & 2 & \forall u \in V \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V,\ 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Two possible decompositions

Find a spanning subgraph with $|V|$ edges that satisfies the 2-degree constraints ($\mathcal{P}' =$ 1-Tree)

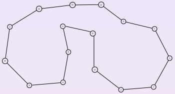
$$\begin{array}{lcll} x(\delta(\{0\})) & = & 2 & \\ x(E(V)) & = & |V| & \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V \setminus \{0\}, 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Example - Traveling Salesman Problem (TSP)

Traveling Salesman Problem Formulation

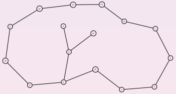
$$\begin{array}{lcll} x(\delta(\{u\})) & = & 2 & \forall u \in V \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V,\ 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Two possible decompositions

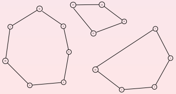
Find a spanning subgraph with $|V|$ edges that satisfies the 2-degree constraints ($\mathcal{P}' =$ 1-Tree)

$$\begin{array}{lcll} x(\delta(\{0\})) & = & 2 & \\ x(E(V)) & = & |V| & \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V \setminus \{0\}, 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Find a 2-matching that satisfies the subtour constraints ($\mathcal{P}' =$ 2-Matching)

$$\begin{array}{lcll} x(\delta(\{u\})) & = & 2 & \forall u \in V \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Outline

Outline

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**
  - Integrated methods: **price-and-cut** and **relax-and-cut**
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(***D***ecomposition for* **I***nteger* **P***rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**
  - Integrated methods: **price-and-cut** and **relax-and-cut**
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(***D***ecomposition for* **I***nteger* **P***rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**
  - Integrated methods: **price-and-cut** and **relax-and-cut**
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(**D**ecomposition for **I**nteger **P**rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**
  - Integrated methods: **price-and-cut** and **relax-and-cut**
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(**D**ecomposition for **I**nteger **P**rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**
  - Integrated methods: **price-and-cut** and **relax-and-cut**
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(**D**ecomposition for **I**nteger **P**rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**
  - Integrated methods: **price-and-cut** and **relax-and-cut**
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(***D***ecomposition for* **I***nteger* **P***rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Outline

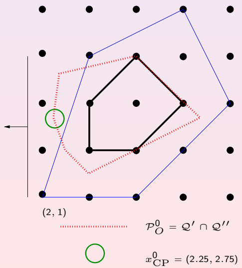
Cutting Plane Method (CPM)

CPM builds an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{CP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid Dx \geq d, A''x \geq b'' \right\}$
- **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x_{\mathrm{CP}})$

$$\mathcal{P}' = \{x \in \mathbb{R}^n \mid Dx \geq d\}$$

exponential number of constraints



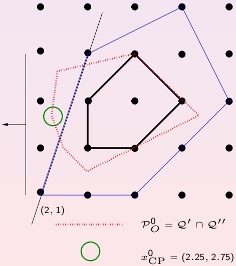
Cutting Plane Method (CPM)

CPM builds an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{CP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid Dx \geq d, A''x \geq b'' \right\}$
- **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x_{\mathrm{CP}})$

$$\mathcal{P}' = \{x \in \mathbb{R}^n \mid Dx \geq d\}$$

exponential number of constraints



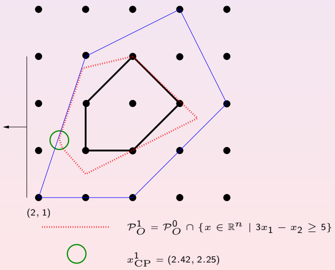
Cutting Plane Method (CPM)

CPM builds an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{CP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid Dx \geq d, A''x \geq b'' \right\}$
- **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x_{\mathrm{CP}})$

$$\mathcal{P}' = \{x \in \mathbb{R}^n \mid Dx \geq d\}$$

exponential number of constraints



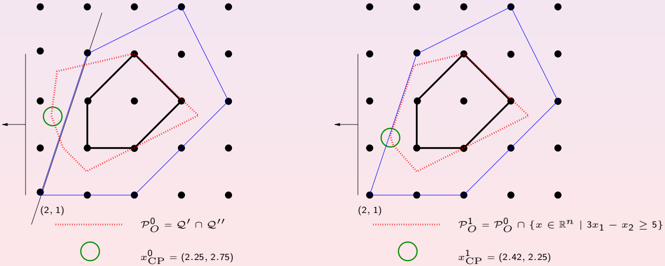
Cutting Plane Method (CPM)

CPM builds an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{CP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid Dx \geq d, A''x \geq b'' \right\}$
- **Subproblem**: $\mathrm{SEP}(\mathcal{P}', x_{\mathrm{CP}})$

$$\mathcal{P}' = \{x \in \mathbb{R}^n \mid Dx \geq d\}$$

exponential number of constraints



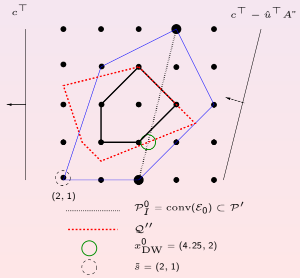
Dantzig-Wolfe Method (DW)

DW builds an *inner* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{DW}} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \mid A'' \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{DW}}^\top A''\right)$

$$\mathcal{P}' = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{E}} s \lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \; \forall s \in \mathcal{E} \right\}$$

exponential number of variables



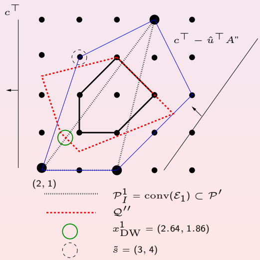
Dantzig-Wolfe Method (DW)

DW builds an *inner* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{DW}} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \mid A'' \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{DW}}^\top A''\right)$

$$\mathcal{P}' = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{E}} s\lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \; \forall s \in \mathcal{E} \right\}$$

exponential number of variables



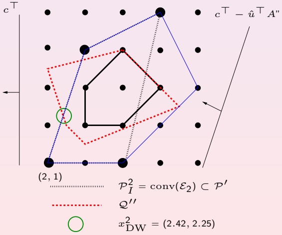
Dantzig-Wolfe Method (DW)

DW builds an *inner* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{DW}} = \min_{\lambda \in \mathbb{R}^{\mathcal{E}}_{+}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \mid A'' \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u^\top_{\mathrm{DW}} A''\right)$

$$\mathcal{P}' = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{E}} s\lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \; \forall s \in \mathcal{E} \right\}$$

exponential number of variables



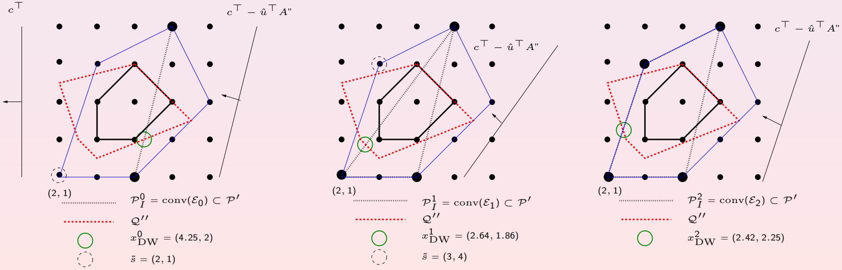
Dantzig-Wolfe Method (DW)

DW builds an *inner* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$

- **Master**: $z_{\mathrm{DW}} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \;\middle|\; A'' \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{DW}}^\top A''\right)$

$$\mathcal{P}' = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{E}} s\lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \; \forall s \in \mathcal{E} \right\}$$

exponential number of variables

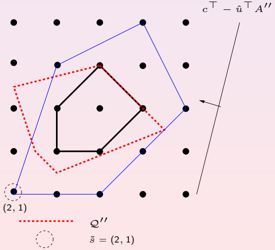


Lagrangian Method (LD)

LD iteratively *traces* an *inner* approximation of $\mathcal{P}'$ penalizing points outside $\mathcal{Q}''$

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A''s) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top A''\right)$

$$z_{\mathrm{LD}} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}_+^{m''}} \left\{ \alpha + b''^\top u \;\middle|\; \left(c^\top - u^\top A''\right) s - \alpha \geq 0 \; \forall s \in \mathcal{E} \right\} = z_{\mathrm{DW}}$$

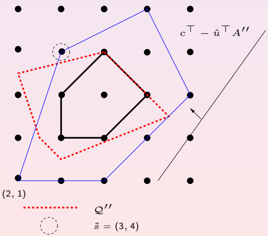


Lagrangian Method (LD)

LD iteratively *traces* an *inner* approximation of $\mathcal{P}'$ penalizing points outside $\mathcal{Q}''$

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A'' s) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top A''\right)$

$$z_{\mathrm{LD}} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}_+^{m''}} \left\{ \alpha + b''^\top u \;\middle|\; \left(c^\top - u^\top A''\right) s - \alpha \geq 0 \; \forall s \in \mathcal{E} \right\} = z_{\mathrm{DW}}$$

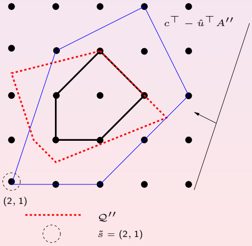


Lagrangian Method (LD)

LD iteratively *traces* an *inner* approximation of $\mathcal{P}'$ penalizing points outside $\mathcal{Q}''$

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A'' s) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top A''\right)$

$$z_{\mathrm{LD}} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}_+^{m''}} \left\{ \alpha + b''^\top u \;\middle|\; \left(c^\top - u^\top A''\right) s - \alpha \geq 0 \;\forall s \in \mathcal{E} \right\} = z_{\mathrm{DW}}$$

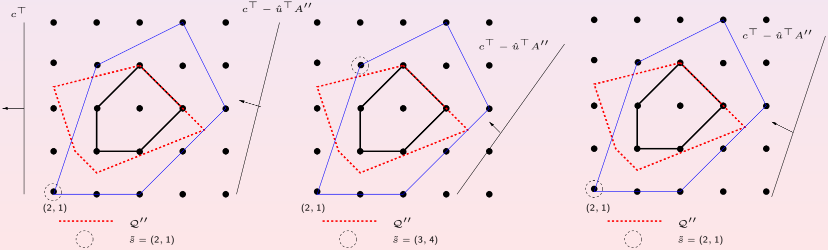


Lagrangian Method (LD)

LD iteratively *traces* an *inner* approximation of $\mathcal{P}'$ penalizing points outside $\mathcal{Q}''$

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}^{m''}_{+}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A'' s) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top A''\right)$

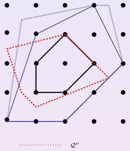
$$z_{\mathrm{LD}} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}^{m''}_{+}} \left\{ \alpha + b''^\top u \;\middle|\; \left(c^\top - u^\top A''\right) s - \alpha \geq 0 \; \forall s \in \mathcal{E} \right\} = z_{\mathrm{DW}}$$



Common Threads

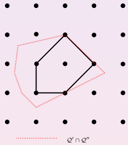
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$



- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

$$z_{\mathrm{CP}} = z_{\mathrm{DW}} = z_{\mathrm{LD}} = z_{\mathrm{D}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{LP}}$$

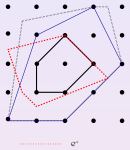


- Traditional decomp-based bounding methods contain two primary steps
  - **Master Problem:** Update the primal/dual solution information
  - **Subproblem:** Update the approximation of $\mathcal{P}'$: $\mathrm{SEP}(\mathcal{P}', x)$ or $\mathrm{OPT}(\mathcal{P}', c)$
- Integrated decomposition methods further improve the bound by considering two implicitly defined polyhedra whose descriptions are iteratively refined.
  - Price-and-Cut (PC)
  - Relax-and-Cut (RC)
  - Decompose-and-Cut (DC)

Common Threads

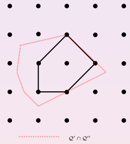
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$



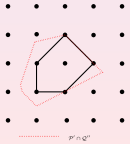
- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

$$z_{\mathrm{CP}} = z_{\mathrm{DW}} = z_{\mathrm{LD}} = z_{\mathrm{D}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{LP}}$$



- Traditional decomp-based bounding methods contain two primary steps
  - **Master Problem:** Update the primal/dual **solution** information
  - **Subproblem:** Update the **approximation** of $\mathcal{P}'$: $\mathrm{SEP}(\mathcal{P}', x)$ or $\mathrm{OPT}(\mathcal{P}', c)$

- Integrated decomposition methods further improve the bound by considering two implicitly defined polyhedra whose descriptions are iteratively refined.
  - Price-and-Cut (PC)
  - Relax-and-Cut (RC)
  - Decompose-and-Cut (DC)



Common Threads

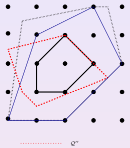
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

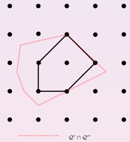
$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$

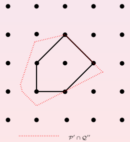
- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

$$z_{\mathrm{CP}} = z_{\mathrm{DW}} = z_{\mathrm{LD}} = z_{\mathrm{D}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{\mathrm{LP}}$$

- Traditional decomp-based bounding methods contain two primary steps
  - **Master Problem:** Update the primal/dual **solution** information
  - **Subproblem:** Update the **approximation** of $\mathcal{P}'$: $\mathrm{SEP}(\mathcal{P}', x)$ or $\mathrm{OPT}(\mathcal{P}', c)$
- Integrated decomposition methods further improve the bound by considering two implicitly defined polyhedra whose descriptions are iteratively refined.
  - **Price-and-Cut** (PC)
  - **Relax-and-Cut** (RC)
  - **Decompose-and-Cut** (DC)





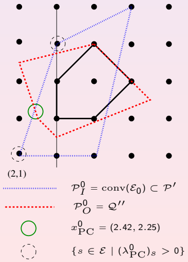


Price-and-Cut Method (PC)

PC approximates $\mathcal{P}$ by building an *inner* approximation of $\mathcal{P}'$ (as in DW) intersected with an *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{PC}} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s\lambda_s\right) \mid D\left(\sum_{s \in \mathcal{E}} s\lambda_s\right) \geq d, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{PC}}^\top D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, x_{\mathrm{PC}}\right)$

- As in CPM, separate $\hat{x}_{\mathrm{PC}} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ from $\mathcal{P}$ and add cuts to $[D, d]$.
- **Key Idea**: Cut generation takes place in the space of the compact formulation, maintaining the structure of the column generation subproblem.

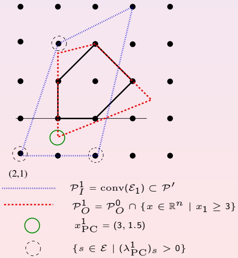


Price-and-Cut Method (PC)

PC approximates $\mathcal{P}$ by building an *inner* approximation of $\mathcal{P}'$ (as in DW) intersected with an *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{PC}} = \min_{\lambda \in \mathbb{R}^{\mathcal{E}}_{+}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \mid D \left(\sum_{s \in \mathcal{E}} s\lambda_s \right) \geq d, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{PC}}^\top D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, x_{\mathrm{PC}}\right)$

- As in CPM, separate $\hat{x}_{\mathrm{PC}} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ from $\mathcal{P}$ and add cuts to $[D, d]$.
- **Key Idea**: Cut generation takes place in the space of the compact formulation, maintaining the structure of the column generation subproblem.

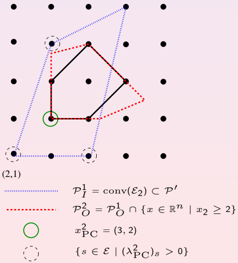


Price-and-Cut Method (PC)

PC approximates $\mathcal{P}$ by building an *inner* approximation of $\mathcal{P}'$ (as in DW) intersected with an *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{PC}} = \min_{\lambda \in \mathbb{R}^{\mathcal{E}}_{+}} \left\{ c^{\top} \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \mid D \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \geq d, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^{\top} - u_{\mathrm{PC}}^{\top} D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, x_{\mathrm{PC}}\right)$

- As in CPM, separate $\hat{x}_{\mathrm{PC}} = \sum_{s \in \mathcal{E}} s \hat{\lambda}_s$ from $\mathcal{P}$ and add cuts to $[D, d]$.
- **Key Idea**: Cut generation takes place in the space of the compact formulation, maintaining the structure of the column generation subproblem.

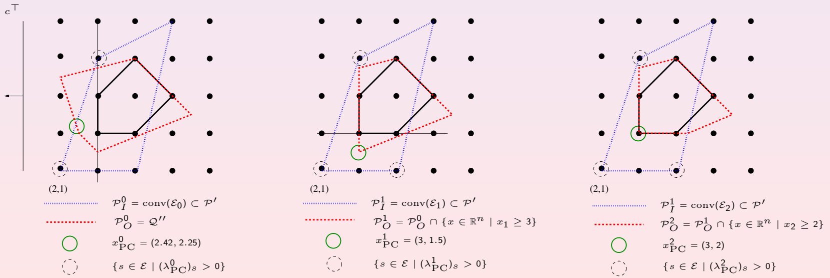


Price-and-Cut Method (PC)

PC approximates $\mathcal{P}$ by building an *inner* approximation of $\mathcal{P}'$ (as in DW) intersected with an *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{PC}} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}} \left\{ c^\top \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \mid D \left(\sum_{s \in \mathcal{E}} s \lambda_s \right) \geq d, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{PC}}^\top D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, x_{\mathrm{PC}}\right)$

- As in CPM, separate $\hat{x}_{\mathrm{PC}} = \sum_{s \in \mathcal{E}} s \hat{\lambda}_s$ from $\mathcal{P}$ and add cuts to $[D, d]$.
- **Key Idea**: Cut generation takes place in the space of the compact formulation, maintaining the structure of the column generation subproblem.

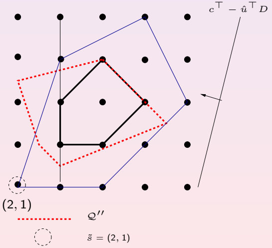


Relax-and-Cut Method (RC)

RC approximates $\mathcal{P}$ by tracing an *inner* approximation of $\mathcal{P}'$ (as in LD) penalizing points outside of a dynamically generated *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, s\right)$

- In each iteration, separate $\hat{s} \in \mathcal{E}$, a solution to the Lagrangian relaxation.
- **Advantage**: Often easier to separate $s \in \mathcal{E}$ from $\mathcal{P}$ than $\hat{x} \in \mathbb{R}^n$.

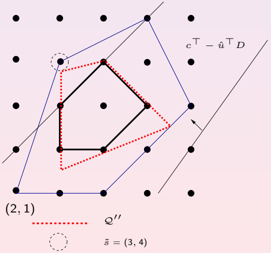


Relax-and-Cut Method (RC)

RC approximates $\mathcal{P}$ by tracing an *inner* approximation of $\mathcal{P}'$ (as in LD) penalizing points outside of a dynamically generated *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\text{LD}} = \max_{u \in \mathbb{R}^{m''}_{+}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **Subproblem**: $\text{OPT}\left(\mathcal{P}', c^\top - u_{\text{LD}}^\top D\right)$ or $\text{SEP}\left(\mathcal{P}, s\right)$

- In each iteration, separate $\hat{s} \in \mathcal{E}$, a solution to the Lagrangian relaxation.
- **Advantage**: Often easier to separate $s \in \mathcal{E}$ from $\mathcal{P}$ than $\hat{x} \in \mathbb{R}^n$.

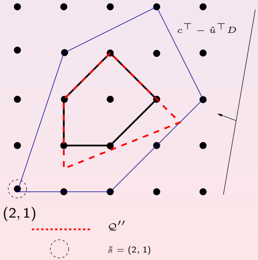


Relax-and-Cut Method (RC)

RC approximates $\mathcal{P}$ by tracing an *inner* approximation of $\mathcal{P}'$ (as in LD) penalizing points outside of a dynamically generated *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, s\right)$

- In each iteration, separate $\hat{s} \in \mathcal{E}$, a solution to the Lagrangian relaxation.
- **Advantage**: Often easier to separate $s \in \mathcal{E}$ from $\mathcal{P}$ than $\hat{x} \in \mathbb{R}^n$.

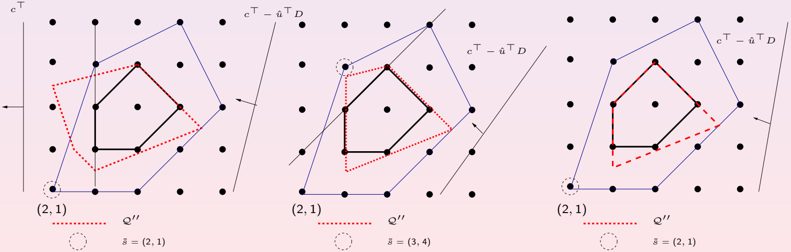


Relax-and-Cut Method (RC)

RC approximates $\mathcal{P}$ by tracing an *inner* approximation of $\mathcal{P}'$ (as in LD) penalizing points outside of a dynamically generated *outer* approximation of $\mathcal{P}$ (as in CPM)

- **Master**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **Subproblem**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top D\right)$ or $\mathrm{SEP}\left(\mathcal{P}, s\right)$

- In each iteration, separate $\hat{s} \in \mathcal{E}$, a solution to the Lagrangian relaxation.
- **Advantage**: Often easier to separate $s \in \mathcal{E}$ from $\mathcal{P}$ than $\hat{x} \in \mathbb{R}^n$.



Structured Separation

- In general, the complexity of $\mathrm{OPT}(X, c) = \mathrm{SEP}(X, x)$.
- **Observation:** Restrictions on input or output can change their complexity.
- Template Paradigm, restricts the *output* of $\mathrm{SEP}(X, x)$ to valid inequalities that conform to a certain structure. This class of inequalities forms a polyhedron $\mathcal{C} \supset X$ (the *closure*).
- For example, let $\mathcal{P}$ be the convex hull of solutions to the TSP.
  - $\mathrm{SEP}(\mathcal{P}, x)$ is $\mathcal{NP}$-Complete.
  - $\mathrm{SEP}(\mathcal{C}, x)$ is polynomially solvable, for $\mathcal{C} \supset \mathcal{P}$
    - $\mathcal{P}^{\mathrm{Subtour}}$, the Subtour Polytope (separation using Min-Cut), or
    - $\mathcal{P}^{\mathrm{Blossom}}$, the Blossom Polytope (separation using Letchford, et al.).
- Structured Separation, restricts the *input* of $\mathrm{SEP}(X, x)$, such that x conforms to some structure. For example, if x is restricted to solutions to a combinatorial problem, then separation often becomes much easier.

Structured Separation

- In general, the complexity of $\mathrm{OPT}(X, c) = \mathrm{SEP}(X, x)$.
- **Observation:** Restrictions on input or output can change their complexity.
- Template Paradigm, restricts the *output* of $\mathrm{SEP}(X, x)$ to valid inequalities that conform to a certain structure. This class of inequalities forms a polyhedron $\mathcal{C} \supset X$ (the *closure*).
- For example, let $\mathcal{P}$ be the convex hull of solutions to the TSP.
  - $\mathrm{SEP}(\mathcal{P}, x)$ is $\mathcal{NP}$-Complete.
  - $\mathrm{SEP}(\mathcal{C}, x)$ is polynomially solvable, for $\mathcal{C} \supset \mathcal{P}$
    - $\mathcal{P}^{\mathrm{Subtour}}$, the Subtour Polytope (separation using Min-Cut), or
    - $\mathcal{P}^{\mathrm{Blossom}}$, the Blossom Polytope (separation using Letchford, et al.).
- Structured Separation, restricts the *input* of $\mathrm{SEP}(X, x)$, such that x conforms to some structure. For example, if x is restricted to solutions to a combinatorial problem, then separation often becomes much easier.

Structured Separation

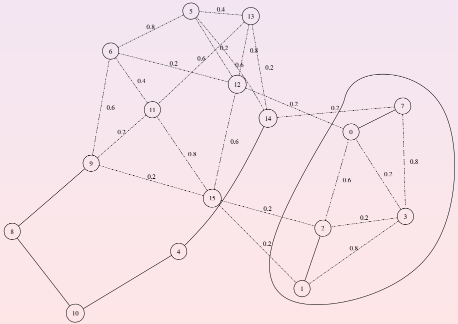
- In general, the complexity of $\mathrm{OPT}(X, c) = \mathrm{SEP}(X, x)$.
- **Observation:** Restrictions on input or output can change their complexity.
- Template Paradigm, restricts the *output* of $\mathrm{SEP}(X, x)$ to valid inequalities that conform to a certain structure. This class of inequalities forms a polyhedron $\mathcal{C} \supset X$ (the *closure*).
- For example, let $\mathcal{P}$ be the convex hull of solutions to the TSP.
  - $\mathrm{SEP}(\mathcal{P}, x)$ is $\mathcal{NP}$-Complete.
  - $\mathrm{SEP}(\mathcal{C}, x)$ is polynomially solvable, for $\mathcal{C} \supset \mathcal{P}$
    - $\mathcal{P}^{\mathrm{Subtour}}$, the Subtour Polytope (separation using Min-Cut), or
    - $\mathcal{P}^{\mathrm{Blossom}}$, the Blossom Polytope (separation using Letchford, et al.).
- Structured Separation, restricts the *input* of $\mathrm{SEP}(X, x)$, such that x conforms to some structure. For example, if x is restricted to solutions to a combinatorial problem, then separation often becomes much easier.

Example - TSP

- Separation of Subtour Inequalities:

$$x(E(S)) \leq |S| - 1$$

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Subtour}}, x)$ for $x \in \mathbb{R}^n$ can be solved in $O\left(|E||V| + |V|^2 \log |V|\right)$ (Min-Cut)
- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Subtour}}, s)$ for s a 2-matching, can be solved in $O(|V|)$
  - Simply determine the connected components C_i, and set $S = C_i$ for each component (each gives a violation of 1).



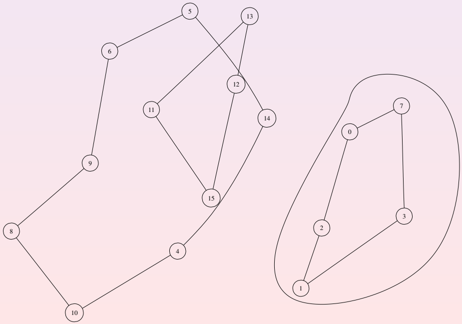
Example - TSP

- Separation of Subtour Inequalities:

$$x(E(S)) \leq |S| - 1$$

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Subtour}}, x)$ for $x \in \mathbb{R}^n$ can be solved in $O\left(|E||V| + |V|^2 \log |V|\right)$ (Min-Cut)

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Subtour}}, s)$ for s a 2-matching, can be solved in $O(|V|)$
  - Simply determine the connected components C_i, and set $S = C_i$ for each component (each gives a violation of 1).

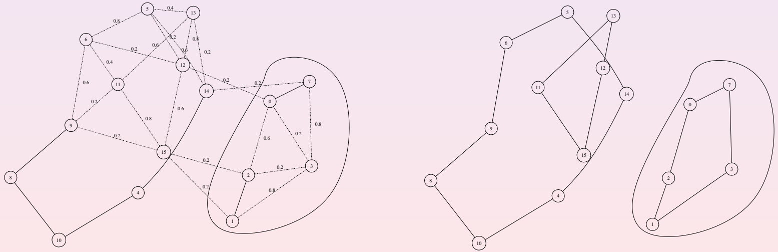


Example - TSP

- Separation of Subtour Inequalities:

$$x(E(S)) \leq |S| - 1$$

- $\text{SEP}(\mathcal{P}^{\text{Subtour}}, x)$ for $x \in \mathbb{R}^n$ can be solved in $O\left(|E||V| + |V|^2 \log |V|\right)$ (Min-Cut)
- $\text{SEP}(\mathcal{P}^{\text{Subtour}}, s)$ for s a 2-matching, can be solved in $O(|V|)$
  - Simply determine the connected components C_i, and set $S = C_i$ for each component (each gives a violation of 1).

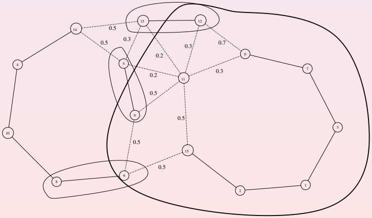


Example - TSP

- Separation of Comb Inequalities:

$$x(E(H)) + \sum_{i=1}^{k} x(E(T_i)) \leq |H| + \sum_{i=1}^{k}(|T_i| - 1) - \lceil k/2 \rceil$$

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, x)$, for $x \in \mathbb{R}^n$ can be solved in $O(|V|^2|E|\log(|V|^2/|E|))$ (Letchford, et al.)
- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, s)$, for s a 1-tree, can be solved in $O(|V|^2)$
  - Construct candidate handles H from BFS tree traversal and an odd (≥ 3) set of edges with one endpoint in H and one in $V \setminus H$ as candidate teeth (each gives a violation of $\lceil k/2 \rceil - 1$).
  - This can also be used as a quick heuristic to separate 1-trees for more general comb structures, for which there is no known polynomial algorithm for separation of arbitrary vectors.



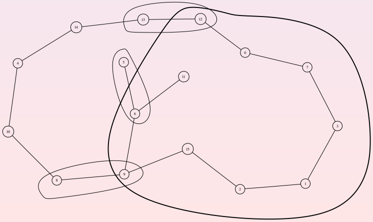
Example - TSP

- Separation of Comb Inequalities:

$$x(E(H)) + \sum_{i=1}^{k} x(E(T_i)) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \lceil k/2 \rceil$$

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, x)$, for $x \in \mathbb{R}^n$ can be solved in $O(|V|^2|E|\log(|V|^2/|E|))$ (Letchford, et al.)

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, s)$, for s a 1-tree, can be solved in $O(|V|^2)$
  - Construct candidate handles H from BFS tree traversal and an odd (≥ 3) set of edges with one endpoint in H and one in $V \setminus H$ as candidate teeth (each gives a violation of $\lceil k/2 \rceil - 1$).
  - This can also be used as a quick heuristic to separate 1-trees for more general comb structures, for which there is no known polynomial algorithm for separation of arbitrary vectors.

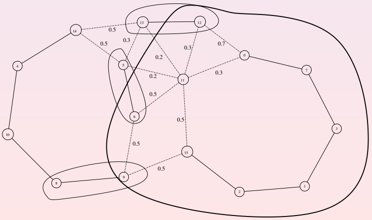


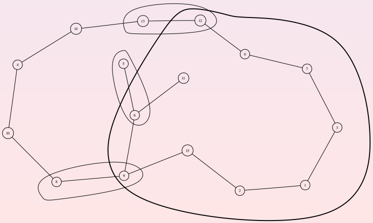
Example - TSP

- Separation of Comb Inequalities:

$$x(E(H)) + \sum_{i=1}^{k} x(E(T_i)) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \lceil k/2 \rceil$$

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, x)$, for $x \in \mathbb{R}^n$ can be solved in $O(|V|^2|E|\log(|V|^2/|E|))$ (Letchford, et al.)
- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, s)$, for s a 1-tree, can be solved in $O(|V|^2)$
  - Construct candidate handles H from BFS tree traversal and an odd (≥ 3) set of edges with one endpoint in H and one in $V \setminus H$ as candidate teeth (each gives a violation of $\lceil k/2 \rceil - 1$).
  - This can also be used as a quick heuristic to separate 1-trees for more general comb structures, for which there is no known polynomial algorithm for separation of arbitrary vectors.





Motivation

- In Relax-and-Cut, solutions to the Lagrangian subproblem $s \in \mathcal{E}$ typically have some *nice* combinatorial structure. So, in RC, $\mathrm{SEP}(\mathcal{P}, s)$, can be relatively easy as opposed to general separation.
- **Question:** Can we take advantage of this in other contexts?
- LP theory says in order to *improve the bound*, it is *necessary and sufficient* to cut off the entire face of optimal solutions F.
- This condition is difficult to verify, so we typically use the *necessary condition* that the generated inequality be violated by some member of that face, $x \in F$.
  - In CPM, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{CP}})$, where $x^t_{\mathrm{CP}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_O \cap \mathcal{Q}''$
  - In PC, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{PC}})$, where $x^t_{\mathrm{PC}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_I \cap \mathcal{P}^t_O$

Motivation

- In Relax-and-Cut, solutions to the Lagrangian subproblem $s \in \mathcal{E}$ typically have some *nice* combinatorial structure. So, in RC, $\mathrm{SEP}(\mathcal{P}, s)$, can be relatively easy as opposed to general separation.
- **Question:** Can we take advantage of this in other contexts?
- LP theory says in order to *improve the bound*, it is *necessary and sufficient* to cut off the entire face of optimal solutions F.
- This condition is difficult to verify, so we typically use the *necessary condition* that the generated inequality be violated by some member of that face, $x \in F$.
  - In CPM, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{CP}})$, where $x^t_{\mathrm{CP}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_O \cap \mathcal{Q}''$
  - In PC, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{PC}})$, where $x^t_{\mathrm{PC}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_I \cap \mathcal{P}^t_O$

Motivation

- In Relax-and-Cut, solutions to the Lagrangian subproblem $s \in \mathcal{E}$ typically have some *nice* combinatorial structure. So, in RC, $\mathrm{SEP}(\mathcal{P}, s)$, can be relatively easy as opposed to general separation.
- **Question:** Can we take advantage of this in other contexts?
- LP theory says in order to *improve the bound*, it is *necessary and sufficient* to cut off the entire face of optimal solutions F.
- This condition is difficult to verify, so we typically use the *necessary condition* that the generated inequality be violated by some member of that face, $x \in F$.
  - In CPM, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{CP}})$, where $x^t_{\mathrm{CP}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_O \cap \mathcal{Q}''$
  - In PC, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{PC}})$, where $x^t_{\mathrm{PC}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_I \cap \mathcal{P}^t_O$

Motivation

- In Relax-and-Cut, solutions to the Lagrangian subproblem $s \in \mathcal{E}$ typically have some *nice* combinatorial structure. So, in RC, $\mathrm{SEP}(\mathcal{P}, s)$, can be relatively easy as opposed to general separation.
- **Question:** Can we take advantage of this in other contexts?
- LP theory says in order to *improve the bound*, it is *necessary and sufficient* to cut off the entire face of optimal solutions F.
- This condition is difficult to verify, so we typically use the *necessary condition* that the generated inequality be violated by some member of that face, $x \in F$.
  - In CPM, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{CP}})$, where $x^t_{\mathrm{CP}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_O \cap \mathcal{Q}''$
  - In PC, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{PC}})$, where $x^t_{\mathrm{PC}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_I \cap \mathcal{P}^t_O$

Motivation

- In Relax-and-Cut, solutions to the Lagrangian subproblem $s \in \mathcal{E}$ typically have some *nice* combinatorial structure. So, in RC, $\mathrm{SEP}(\mathcal{P}, s)$, can be relatively easy as opposed to general separation.
- **Question:** Can we take advantage of this in other contexts?
- LP theory says in order to *improve the bound*, it is *necessary and sufficient* to cut off the entire face of optimal solutions F.
- This condition is difficult to verify, so we typically use the *necessary condition* that the generated inequality be violated by some member of that face, $x \in F$.
  - In CPM, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{CP}})$, where $x^t_{\mathrm{CP}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_O \cap \mathcal{Q}''$
  - In PC, we solve $\mathrm{SEP}(\mathcal{P}, x^t_{\mathrm{PC}})$, where $x^t_{\mathrm{PC}} \in F^t$, and F^t is optimal face over $\mathcal{P}^t_I \cap \mathcal{P}^t_O$

Motivation

- Consider the following set

$$\mathcal{S}(u,\alpha) = \left\{ s \in \mathcal{E} \;\middle|\; \left(c^\top - u^\top A''\right) s = \alpha \right\}$$

- $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is the set of extreme points with $rc(s) = 0$ in the DW-LP master or the set of alternative optimal solutions to the Lagrangian subproblem.

Theorems

1. $F^t \subseteq conv(\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}}))$
   - Separation of $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is also *necessary and sufficient*
2. $\mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\} \subseteq \mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$
   - The optimal decomposition is contained in $\mathcal{S}$
3. $(a,\beta) \in \mathbb{R}^{(n+1)}$ improving $\Rightarrow \exists s \in \mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\}$ s.t. $a^\top s < \beta$
   - Every improving ineq must violate at least one e.p. in the optimal decomposition

- Theorems 1-3, along with the observation that structured separation can be relatively easy, motivates the following revised Price-and-Cut method.

Motivation

- Consider the following set

$$\mathcal{S}(u,\alpha) = \left\{ s \in \mathcal{E} \;\middle|\; \left(c^\top - u^\top A''\right) s = \alpha \right\}$$

- $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is the set of extreme points with $rc(s) = 0$ in the DW-LP master or the set of alternative optimal solutions to the Lagrangian subproblem.

Theorems

1. $F^t \subseteq conv(\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}}))$
   - Separation of $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is also *necessary and sufficient*
2. $\mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\} \subseteq \mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$
   - The optimal decomposition is contained in $\mathcal{S}$
3. $(a, \beta) \in \mathbb{R}^{(n+1)}$ improving $\Rightarrow \exists s \in \mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\}$ s.t. $a^\top s < \beta$
   - Every improving ineq must violate at least one e.p. in the optimal decomposition

- Theorems 1-3, along with the observation that structured separation can be relatively easy, motivates the following revised Price-and-Cut method.

Motivation

- Consider the following set

$$\mathcal{S}(u,\alpha) = \left\{ s \in \mathcal{E} \;\middle|\; \left(c^\top - u^\top A''\right) s = \alpha \right\}$$

- $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is the set of extreme points with $rc(s) = 0$ in the DW-LP master or the set of alternative optimal solutions to the Lagrangian subproblem.

Theorems

1. $F^t \subseteq conv(\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}}))$
   - Separation of $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is also *necessary and sufficient*
2. $\mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\} \subseteq \mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$
   - The optimal decomposition is contained in $\mathcal{S}$
3. $(a, \beta) \in \mathbb{R}^{(n+1)}$ improving $\Rightarrow \exists s \in \mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\}$ s.t. $a^\top s < \beta$
   - Every improving ineq must violate at least one e.p. in the optimal decomposition

- Theorems 1-3, along with the observation that structured separation can be relatively easy, motivates the following revised Price-and-Cut method.

Motivation

- Consider the following set

$$\mathcal{S}(u,\alpha) = \left\{ s \in \mathcal{E} \;\middle|\; \left(c^\top - u^\top A''\right) s = \alpha \right\}$$

- $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is the set of extreme points with $rc(s) = 0$ in the DW-LP master or the set of alternative optimal solutions to the Lagrangian subproblem.

Theorems

1. $F^t \subseteq conv(\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}}))$
   - Separation of $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is also *necessary and sufficient*
2. $\mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\} \subseteq \mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$
   - The optimal decomposition is contained in $\mathcal{S}$
3. $(a, \beta) \in \mathbb{R}^{(n+1)}$ improving $\Rightarrow \exists s \in \mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\}$ s.t. $a^\top s < \beta$
   - Every improving ineq must violate at least one e.p. in the optimal decomposition

- Theorems 1-3, along with the observation that structured separation can be relatively easy, motivates the following revised Price-and-Cut method.

Motivation

- Consider the following set

$$\mathcal{S}(u,\alpha) = \left\{ s \in \mathcal{E} \;\middle|\; \left(c^\top - u^\top A''\right) s = \alpha \right\}$$

- $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is the set of extreme points with $rc(s) = 0$ in the DW-LP master or the set of alternative optimal solutions to the Lagrangian subproblem.

Theorems

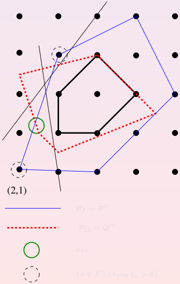
1. $F^t \subseteq conv(\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}}))$
   - Separation of $\mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$ is also *necessary and sufficient*
2. $\mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\} \subseteq \mathcal{S}(u^t_{\mathrm{PC}}, \alpha^t_{\mathrm{PC}})$
   - The optimal decomposition is contained in $\mathcal{S}$
3. $(a,\beta) \in \mathbb{R}^{(n+1)}$ improving $\Rightarrow \exists s \in \mathcal{D} = \{s \in \mathcal{E} \mid \lambda^t_s > 0\}$ s.t. $a^\top s < \beta$
   - Every improving ineq must violate at least one e.p. in the optimal decomposition

- Theorems 1-3, along with the observation that structured separation can be relatively easy, motivates the following revised Price-and-Cut method.

Price-and-Cut (Revisited)

Price-and-Cut (Revisited): As normal, use **DW** as the bounding method, but use the decomposition obtained in each iteration to generate improving inequalities, as in **RC**.

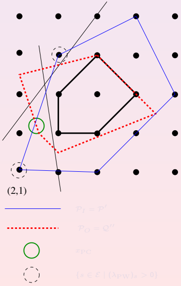
- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with RC, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- RC only gives us one member of $\mathcal{E}$ to separate, while PC gives us a set, one of which must be violated by any inequality violated by $\hat{x}_{\mathrm{PC}}$
- Provides an alternative *necessary* (but not *sufficient*) condition to find an improving inequality which is very **easy to implement and understand**.



Price-and-Cut (Revisited)

Price-and-Cut (Revisited): As normal, use **DW** as the bounding method, but use the decomposition obtained in each iteration to generate improving inequalities, as in **RC**.

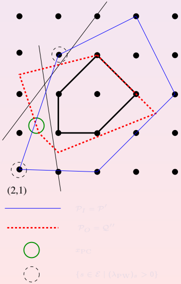
- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with **RC**, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- RC only gives us one member of $\mathcal{E}$ to separate, while PC gives us a set, one of which must be violated by any inequality violated by $\hat{x}_{\mathrm{PC}}$
- Provides an alternative *necessary* (but not *sufficient*) condition to find an improving inequality which is very **easy to implement and understand**.



Price-and-Cut (Revisited)

Price-and-Cut (Revisited): As normal, use **DW** as the bounding method, but use the decomposition obtained in each iteration to generate improving inequalities, as in **RC**.

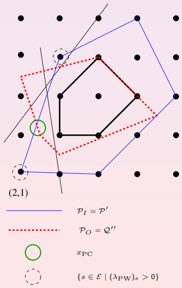
- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with **RC**, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- **RC** only gives us one member of $\mathcal{E}$ to separate, while **PC** gives us a set, one of which must be violated by any inequality violated by $\hat{x}_{\mathrm{PC}}$
- Provides an alternative *necessary* (but not *sufficient*) condition to find an improving inequality which is very **easy to implement and understand**.



Price-and-Cut (Revisited)

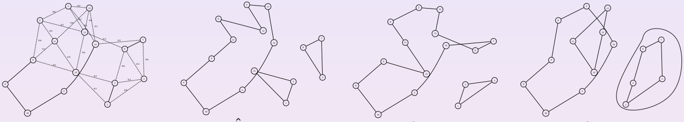
Price-and-Cut (Revisited): As normal, use **DW** as the bounding method, but use the decomposition obtained in each iteration to generate improving inequalities, as in **RC**.

- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with **RC**, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- **RC** only gives us one member of $\mathcal{E}$ to separate, while **PC** gives us a set, one of which must be violated by any inequality violated by $\hat{x}_{\mathrm{PC}}$
- Provides an alternative *necessary* (but not *sufficient*) condition to find an improving inequality which is very **easy to implement and understand**.



Price-and-Cut (Revisited)

- The violated subtour found by separating the 2-matching *also* violates the fractional point, but was found at little cost.

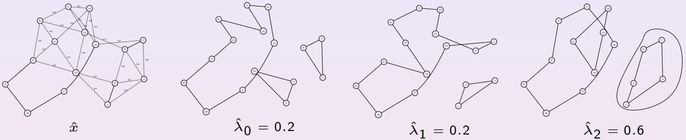


$\hat{x}$ $\qquad \hat{\lambda}_0 = 0.2$ $\qquad \hat{\lambda}_1 = 0.2$ $\qquad \hat{\lambda}_2 = 0.6$

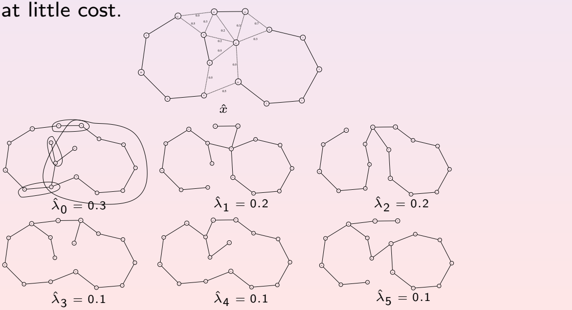
- Similarly, the violated blossom found by separating the 1-tree *also* violates the fractional point, but was found at little cost.

Price-and-Cut (Revisited)

- The violated subtour found by separating the 2-matching *also* violates the fractional point, but was found at little cost.



- Similarly, the violated blossom found by separating the 1-tree *also* violates the fractional point, but was found at little cost.



Decompose-and-Cut Method (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$

$$\min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}, (x^+, x^-) \in \mathbb{R}_+^n} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

Decompose-and-Cut Method (DC)

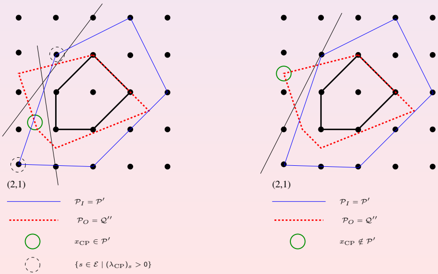
Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$

$$\min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}, (x^+, x^-) \in \mathbb{R}_+^n} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- If $\hat{x}_{\mathrm{CP}}$ lies outside $\mathcal{P}'$ the decomposition will fail
- By the *Farkas Lemma* the proof of infeasibility provides a valid and violated inequality

Decomposition Cuts

$$\begin{aligned} u_{\mathrm{DC}}^t s + \alpha_{\mathrm{DC}}^t &\le 0 \;\forall s \in \mathcal{P}' \quad \text{and} \\ u_{\mathrm{DC}}^t \hat{x}_{\mathrm{CP}} + \alpha_{\mathrm{DC}}^t &> 0 \end{aligned}$$



Decompose-and-Cut Method (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$.

$$\min_{\lambda \in \mathbb{R}^{\mathcal{E}}_{+}, (x^+, x^-) \in \mathbb{R}^n_+} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- Original idea proposed by Ralphs for VRP
  - Later used in TSP Concorde by ABCC (*non-template cuts*)
  - Now being used (in some form) for generic MILP by Gurobi
- This tells us that our cuts are *missing something* related to $\mathcal{P}'$
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because *feasibility* problem and
  - $\hat{x}_i = 0 \Rightarrow s_i = 0$, can remove constraints not in support, and
  - $\hat{x}_i = 1$ and $s_i \in \{0, 1\} \Rightarrow$ constraint is redundant with convexity constraint
  - Often gets *lucky* and produces incumbent solutions to original IP

Decompose-and-Cut Method (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$.

$$\min_{\lambda \in \mathbb{R}^{\mathcal{E}}_{+}, (x^+, x^-) \in \mathbb{R}^n_+} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- Original idea proposed by Ralphs for VRP
  - Later used in TSP **Concorde** by ABCC (*non-template cuts*)
  - Now being used (in some form) for generic MILP by **Gurobi**
- This tells us that our cuts are *missing something* related to $\mathcal{P}'$
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because *feasibility* problem and
  - $\hat{x}_i = 0 \Rightarrow s_i = 0$, can remove constraints not in support, and
  - $\hat{x}_i = 1$ and $s_i \in \{0, 1\} \Rightarrow$ constraint is redundant with convexity constraint
  - Often gets *lucky* and produces incumbent solutions to original IP

Decompose-and-Cut Method (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$.

$$\min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}, (x^+, x^-) \in \mathbb{R}_+^n} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- Original idea proposed by Ralphs for VRP
  - Later used in TSP **Concorde** by ABCC (*non-template cuts*)
  - Now being used (in some form) for generic MILP by **Gurobi**
- This tells us that our cuts are *missing something* related to $\mathcal{P}'$
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because *feasibility* problem and
  - $\hat{x}_i = 0 \Rightarrow s_i = 0$, can remove constraints not in support, and
  - $\hat{x}_i = 1$ and $s_i \in \{0, 1\} \Rightarrow$ constraint is redundant with convexity constraint
  - Often gets *lucky* and produces incumbent solutions to original IP

Decompose-and-Cut Method (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$.

$$\min_{\lambda \in \mathbb{R}_+^{\mathcal{E}}, (x^+, x^-) \in \mathbb{R}_+^n} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- Original idea proposed by Ralphs for VRP
  - Later used in TSP **Concorde** by ABCC (*non-template cuts*)
  - Now being used (in some form) for generic MILP by **Gurobi**
- This tells us that our cuts are *missing something* related to $\mathcal{P}'$
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because *feasibility* problem and
  - $\hat{x}_i = 0 \Rightarrow s_i = 0$, can remove constraints not in support, and
  - $\hat{x}_i = 1$ and $s_i \in \{0, 1\} \Rightarrow$ constraint is redundant with convexity constraint
  - Often gets *lucky* and produces incumbent solutions to original IP

Decompose-and-Cut Method (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$.

$$\min_{\lambda \in \mathbb{R}^{\mathcal{E}}_{+}, (x^+, x^-) \in \mathbb{R}^n_+} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- Original idea proposed by Ralphs for VRP
  - Later used in TSP **Concorde** by ABCC (*non-template cuts*)
  - Now being used (in some form) for generic MILP by **Gurobi**
- This tells us that our cuts are *missing something* related to $\mathcal{P}'$
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because *feasibility* problem and
  - $\hat{x}_i = 0 \Rightarrow s_i = 0$, can remove constraints not in support, and
  - $\hat{x}_i = 1 \text{ and } s_i \in \{0, 1\} \Rightarrow$ constraint is redundant with convexity constraint
  - Often gets *lucky* and produces incumbent solutions to original IP

Branching for Inner Methods (PC)

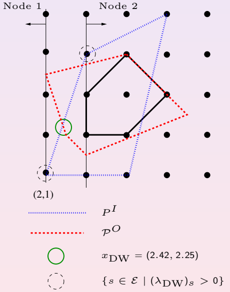
- Add column bounds to $[A'', b'']$ and map back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s \hat{\lambda}_s$
- Variable branching in the compact space is constraint branching in the extended space
- This idea takes care of (most of) the design issues related to branching for inner methods
- Current Limitation: Identical subproblems are currently treated like non-identical.

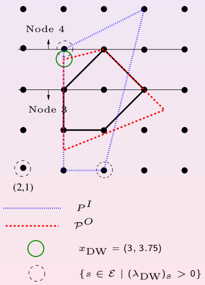
Branching for Inner Methods (PC)

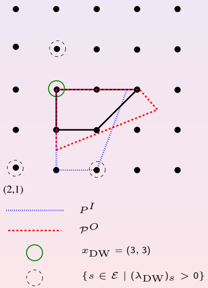
- Add column bounds to $[A'', b'']$ and map back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$
- Variable branching in the compact space is constraint branching in the extended space
- This idea takes care of (most of) the design issues related to branching for inner methods
- Current Limitation: Identical subproblems are currently treated like non-identical.

Branching for Inner Methods (PC)

- Add column bounds to $[A'', b'']$ and map back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$
- Variable branching in the compact space is constraint branching in the extended space
- This idea takes care of (most of) the design issues related to branching for inner methods
- Current Limitation: Identical subproblems are currently treated like non-identical.

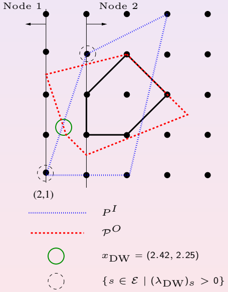


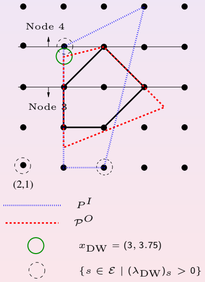


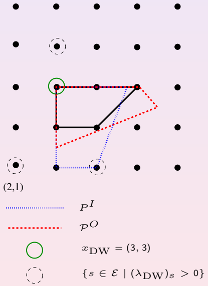


Branching for Inner Methods (PC)

- Add column bounds to $[A'', b'']$ and map back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$
- Variable branching in the compact space is constraint branching in the extended space
- This idea takes care of (most of) the design issues related to branching for inner methods
- Current Limitation: Identical subproblems are currently treated like non-identical.







$$\begin{array}{lrl} \text{Node 1:} & 4\lambda_{(4,1)} + 5\lambda_{(5,5)} + 2\lambda_{(2,1)} + 3\lambda_{(3,4)} \leq & 2 \\ \text{Node 2:} & 4\lambda_{(4,1)} + 5\lambda_{(5,5)} + 2\lambda_{(2,1)} + 3\lambda_{(3,4)} \geq & 3 \end{array}$$

Branching for Inner Methods (RC)

- In general, Lagrangian methods do *not* provide a primal solution λ
- Let $\mathcal{B}$ define the extreme points found in solving subproblems for z_{LD}
- Build an inner approximation using this set, then proceed as in PC

$$\mathcal{P}_I = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{B}} s\lambda_s, \sum_{s \in \mathcal{B}} \lambda_s = 1, \lambda_s \geq 0 \;\forall s \in \mathcal{B} \right\}$$

$$\min_{\lambda \in \mathbb{R}^{\mathcal{B}}_+} \left\{ c^\top \left(\sum_{s \in \mathcal{B}} s\lambda_s \right) \;\middle|\; A'' \left(\sum_{s \in \mathcal{B}} s\lambda_s \right) \geq b'', \sum_{s \in \mathcal{B}} \lambda_s = 1 \right\}$$

- Closely related to *volume* algorithm and *bundle* methods

Branching for Inner Methods (RC)

- In general, Lagrangian methods do *not* provide a primal solution λ
- Let $\mathcal{B}$ define the extreme points found in solving subproblems for z_{LD}
- Build an inner approximation using this set, then proceed as in PC

$$\mathcal{P}_I = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{B}} s\lambda_s, \sum_{s \in \mathcal{B}} \lambda_s = 1, \lambda_s \geq 0 \; \forall s \in \mathcal{B} \right\}$$

$$\min_{\lambda \in \mathbb{R}^{\mathcal{B}}_{+}} \left\{ c^\top \left(\sum_{s \in \mathcal{B}} s\lambda_s \right) \;\middle|\; A'' \left(\sum_{s \in \mathcal{B}} s\lambda_s \right) \geq b'', \sum_{s \in \mathcal{B}} \lambda_s = 1 \right\}$$

- Closely related to *volume* algorithm and *bundle* methods

Branching for Inner Methods (RC)

- In general, Lagrangian methods do *not* provide a primal solution λ
- Let $\mathcal{B}$ define the extreme points found in solving subproblems for z_{LD}
- Build an inner approximation using this set, then proceed as in PC

$$\mathcal{P}_I = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{s \in \mathcal{B}} s\lambda_s, \sum_{s \in \mathcal{B}} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in \mathcal{B} \right\}$$

$$\min_{\lambda \in \mathbb{R}^{\mathcal{B}}_{+}} \left\{ c^\top \left(\sum_{s \in \mathcal{B}} s\lambda_s \right) \;\middle|\; A'' \left(\sum_{s \in \mathcal{B}} s\lambda_s \right) \geq b'', \sum_{s \in \mathcal{B}} \lambda_s = 1 \right\}$$

- Closely related to *volume* algorithm and *bundle* methods

Relaxation Separability

- Key motivation of original DW was separability of the subproblem
- Independence lends itself nicely to **parallel implementation**
- Projections into subspace for each block, then take e.p.'s from each

$$
\begin{pmatrix}
A''_1 & A''_2 & \cdots & A''_\kappa \\
\hline
A'_1 & & & \\
 & A'_2 & & \\
 & & \ddots & \\
 & & & A'_\kappa
\end{pmatrix}
$$

$$
\begin{aligned}
\mathcal{P}'_k &= \operatorname{conv}\{x \in \mathbb{Z}^n \mid A'_k x \geq b'_k, x_i = 0 \ \forall i \in K \setminus \{k\}\} \\
\mathcal{P}' &= \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{k \in K} \sum_{s \in \mathcal{E}^k} s \lambda^k_s, \sum_{k \in K} \sum_{s \in \mathcal{E}^k} \lambda^k_s = 1 \ \forall k \in K, \lambda^k_s \geq 0 \ \forall k \in K,\ s \in \mathcal{E}^k \right\} \\
z_{\mathrm{DW}} &= \min_{\lambda} \left\{ c^\top \left(\sum_{k \in K} \sum_{s \in \mathcal{E}^k} s \lambda^k_s \right) \;\middle|\; A'' \left(\sum_{k \in K} \sum_{s \in \mathcal{E}^k} s \lambda^k_s \right) \geq b'', \sum_{s \in \mathcal{E}^k} \lambda^k_s = 1 \ \forall k \in K \right\}
\end{aligned}
$$

Relaxation Separability

- Key motivation of original DW was separability of the subproblem
- Independence lends itself nicely to **parallel implementation**
- Projections into subspace for each block, then take e.p.'s from each

$$\begin{pmatrix} A''_1 & A''_2 & \cdots & A''_\kappa \\ \hline A'_1 & & & \\ & A'_2 & & \\ & & \ddots & \\ & & & A'_\kappa \end{pmatrix}$$

$$\begin{aligned}
\mathcal{P}'_k &= \operatorname{conv}\{x \in \mathbb{Z}^n \mid A'_k x \geq b'_k, x_i = 0 \ \forall i \in K \setminus \{k\}\} \\
\mathcal{P}' &= \left\{ x \in \mathbb{R}^n \;\middle|\; x = \sum_{k \in K} \sum_{s \in \mathcal{E}^k} s\lambda^k_s, \sum_{k \in K} \sum_{s \in \mathcal{E}^k} \lambda^k_s = 1 \ \forall k \in K, \lambda^k_s \geq 0 \ \forall k \in K,\ s \in \mathcal{E}^k \right\} \\
z_{\mathrm{DW}} &= \min_{\lambda} \left\{ c^\top \left(\sum_{k \in K} \sum_{s \in \mathcal{E}^k} s\lambda^k_s \right) \;\middle|\; A'' \left(\sum_{k \in K} \sum_{s \in \mathcal{E}^k} s\lambda^k_s \right) \geq b'', \sum_{s \in \mathcal{E}^k} \lambda^k_s = 1 \ \forall k \in K \right\}
\end{aligned}$$

Relaxation Separability

- Key motivation of original DW was separability of the subproblem
- Independence lends itself nicely to **parallel implementation**
- Projections into subspace for each block, then take e.p.'s from each

Generalized Assignment Problem (GAP)

$$
\begin{array}{llll}
\min & \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} & & \\
 & \sum_{j \in N} w_{ij} x_{ij} \leq & b_i & \forall i \in M \\
 & \sum_{i \in M} x_{ij} = & 1 & \forall j \in N \\
 & x_{ij} \in & \{0,1\} & \forall i,j \in M \times N
\end{array}
$$

Identical Subproblems

- One motivation for using inner methods like DW is to break symmetry
- Block-diagonal special-case: *identical subproblems*

Vehicle Routing Problem (VRP)

$$
\begin{array}{llll}
\min & \displaystyle\sum_{k \in M} \sum_{(i,j) \in A} c_{ij} x_{ijk} & & \\
& \displaystyle\sum_{k \in M} \sum_{j \in N} x_{ijk} & = 1 & \forall i \in V \\
& \displaystyle\sum_{i \in V} \sum_{j \in N} d_i x_{ijk} & \leq C & \forall k \in M \\
& \displaystyle\sum_{j \in N} x_{0jk} & = 1 & \forall k \in M \\
& \displaystyle\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} & = 0 & \forall h \in V, k \in M \\
& \displaystyle\sum_{i \in N} x_{i,n+1,k} & = 1 & \forall k \in M \\
& x_{ijk} \in \{0,1\} & & \forall (i,j) \in A, k \in M
\end{array}
$$

Identical Subproblems

- One motivation for using inner methods like DW is to break symmetry
- Block-diagonal special-case: *identical subproblems*

$$\Lambda_s = \sum_{k \in K} s\lambda_s^k \ \forall s \in \mathcal{E}$$

$$z_{\mathrm{DW}} = \min_{\Lambda} \left\{ c^\top (s\Lambda_s) \;\middle|\; A'' \left(\sum_{s \in \mathcal{E}} s\Lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}} \Lambda_s^k = K \right\}$$

- Aggregation step breaks our dependence on one-to-one mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$
- Vanderbeck, et al. has been investigating disaggregation based on lexicographic ordering

Nested Polyhedra

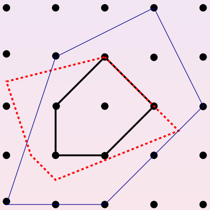
- Outer methods use various approximations to improve the bound (template paradigm)
- **New Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $\mathcal{P}'_N \subset \mathcal{P}'$, we can also *heuristically* solve $\mathrm{OPT}\left(\mathcal{P}'_N, c^\top - u^\top A''\right)$
- Can greatly improve generation of incumbents, upper bounds on z_{IP}
- The more diverse the pool of columns, the better the chance to find good incumbents

Nested Polyhedra

- Outer methods use various approximations to improve the bound (template paradigm)
- **New Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $\mathcal{P}'_N \subset \mathcal{P}'$, we can also *heuristically* solve $\mathrm{OPT}\left(\mathcal{P}'_N, c^\top - u^\top A''\right)$
- Can greatly improve generation of incumbents, upper bounds on z_{IP}
- The more diverse the pool of columns, the better the chance to find good incumbents

Nested Polyhedra

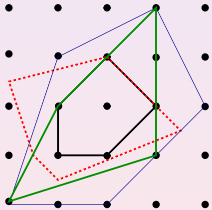
- Outer methods use various approximations to improve the bound (template paradigm)
- **New Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $\mathcal{P}'_N \subset \mathcal{P}'$, we can also *heuristically* solve $\mathrm{OPT}\left(\mathcal{P}'_N, c^\top - u^\top A''\right)$
- Can greatly improve generation of incumbents, upper bounds on z_{IP}
- The more diverse the pool of columns, the better the chance to find good incumbents



$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$
$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$
$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Nested Polyhedra

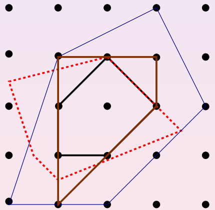
- Outer methods use various approximations to improve the bound (template paradigm)
- **New Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $\mathcal{P}'_N \subset \mathcal{P}'$, we can also *heuristically* solve $\mathrm{OPT}\left(\mathcal{P}'_N, c^\top - u^\top A''\right)$
- Can greatly improve generation of incumbents, upper bounds on z_{IP}
- The more diverse the pool of columns, the better the chance to find good incumbents



$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$
$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$
$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Nested Polyhedra

- Outer methods use various approximations to improve the bound (template paradigm)
- **New Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $\mathcal{P}'_N \subset \mathcal{P}'$, we can also *heuristically* solve $\mathrm{OPT}\left(\mathcal{P}'_N, c^\top - u^\top A''\right)$
- Can greatly improve generation of incumbents, upper bounds on z_{IP}
- The more diverse the pool of columns, the better the chance to find good incumbents



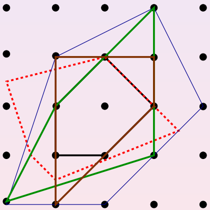
$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$

$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Nested Polyhedra

- Outer methods use various approximations to improve the bound (template paradigm)
- **New Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $\mathcal{P}'_N \subset \mathcal{P}'$, we can also *heuristically* solve $\mathrm{OPT}\left(\mathcal{P}'_N, c^\top - u^\top A''\right)$
- Can greatly improve generation of incumbents, upper bounds on z_{IP}
- The more diverse the pool of columns, the better the chance to find good incumbents



$\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$

$\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

Nested Polyhedra - Example

Vehicle Routing Problem

$$
\begin{array}{llll}
\min & \sum_{e \in E} c_e x_e & & \\
 & x(\delta(\{0\})) & = & 2k \\
 & x(\delta(\{v\})) & = & 2 & \forall v \in V \\
 & x(\delta(S)) & \geq & 2b(S) & \forall S \subseteq V,\ |S| > 1 \\
 & x_e \in \{0,1\} & & & \forall e \in E(V) \\
 & x_e \in \{0,1,2\} & & & \forall e \in \delta(\{0\})
\end{array}
$$

- **Relaxation:** b-Matching $\mathcal{P}' = \mathcal{P}^{\mathrm{bMatch}}$

$$
\begin{array}{llll}
\min & \sum_{e \in E} c_e x_e & & \\
 & x(\delta(\{0\})) & = & 2k \\
 & x(\delta(\{v\})) & = & 2 & \forall v \in V \\
 & x_e \in \{0,1\} & & & \forall e \in E(V) \\
 & x_e \in \{0,1,2\} & & & \forall e \in \delta(\{0\})
\end{array}
$$

Nested Polyhedra - Example

Vehicle Routing Problem

$$
\begin{array}{llll}
\min & \sum_{e \in E} c_e x_e & & \\
& x(\delta(\{0\})) = & 2k & \\
& x(\delta(\{v\})) = & 2 & \forall v \in V \\
& x(\delta(S)) \geq & 2b(S) & \forall S \subseteq V, \ |S| > 1 \\
& x_e \in \{0,1\} & & \forall e \in E(V) \\
& x_e \in \{0,1,2\} & & \forall e \in \delta(\{0\})
\end{array}
$$

- **Nested Relaxation:** Multiple Traveling Salesman $\mathcal{P}^{\mathrm{kTSP}} \subset \mathcal{P}^{\mathrm{bMatch}}$

$$
\begin{array}{llll}
\min & \sum_{e \in E} c_e x_e & & \\
& x(\delta(\{0\})) = & 2k & \\
& x(\delta(\{v\})) = & 2 & \forall v \in V \\
& x(\delta(S)) \geq & 2 & \forall S \subseteq V, \ |S| > 1 \\
& x_e \in \{0,1\} & & \forall e \in E(V) \\
& x_e \in \{0,1,2\} & & \forall e \in \delta(\{0\})
\end{array}
$$

Nested Polyhedra - Example

Vehicle Routing Problem

$$\begin{array}{llll} \min & \sum_{e \in E} c_e x_e & & \\ & x(\delta(\{0\})) & = & 2k & \\ & x(\delta(\{v\})) & = & 2 & \forall v \in V \\ & x(\delta(S)) & \geq & 2b(S) & \forall S \subseteq V,\ |S| > 1 \\ & x_e \in \{0,1\} & & & \forall e \in E(V) \\ & x_e \in \{0,1,2\} & & & \forall e \in \delta(\{0\}) \end{array}$$

- **Nested Relaxation:** b-Matching (plus some GSECs) $\mathcal{P}^{\mathrm{bMatch+}} \subset \mathcal{P}^{\mathrm{bMatch}}$

$$\begin{array}{llll} \min & \sum_{e \in E} c_e x_e & & \\ & x(\delta(\{0\})) & = & 2k & \\ & x(\delta(\{v\})) & = & 2 & \forall v \in V \\ & x(\delta(S)) & \geq & 2b(S) & \forall S \in \mathcal{G} \\ & x_e \in \{0,1\} & & & \forall e \in E(V) \\ & x_e \in \{0,1,2\} & & & \forall e \in \delta(\{0\}) \end{array}$$

Other Considerations

- **Integration of generic MILP cuts** - *new idea*
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP cuts in RC/PC context
- **Initial columns**
  - Solve $\mathrm{OPT}(\mathcal{P}', c + r)$ for random perturbations
  - Solve $\mathrm{OPT}(\mathcal{P}_N)$ heuristically
  - Run several iterations of LD or DC collecting extreme points
- **Price-and-branch heuristic**
  - For block-angular case, at end of each node, solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
- **Choice of master LP solver**
  - Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
  - Primal simplex after adding columns (warm-start primal feasible)
  - Interior-point methods might help with stabilization vs extremal duals
- **Compression of master LP and object pools**
  - Reduce size of master LP, improve efficiency of subproblem processing

Other Considerations

- **Integration of generic MILP cuts** - *new idea*
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP cuts in RC/PC context
- **Initial columns**
  - Solve $\mathrm{OPT}(\mathcal{P}', c + r)$ for random perturbations
  - Solve $\mathrm{OPT}(\mathcal{P}_N)$ heuristically
  - Run several iterations of LD or DC collecting extreme points
- **Price-and-branch heuristic**
  - For block-angular case, at end of each node, solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
- **Choice of master LP solver**
  - Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
  - Primal simplex after adding columns (warm-start primal feasible)
  - Interior-point methods might help with stabilization vs extremal duals
- **Compression of master LP and object pools**
  - Reduce size of master LP, improve efficiency of subproblem processing

Other Considerations

- **Integration of generic MILP cuts** - *new idea*
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP cuts in RC/PC context
- **Initial columns**
  - Solve $\mathrm{OPT}(\mathcal{P}', c + r)$ for random perturbations
  - Solve $\mathrm{OPT}(\mathcal{P}_N)$ heuristically
  - Run several iterations of LD or DC collecting extreme points
- **Price-and-branch heuristic**
  - For block-angular case, at end of each node, solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
- **Choice of master LP solver**
  - Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
  - Primal simplex after adding columns (warm-start primal feasible)
  - Interior-point methods might help with stabilization vs extremal duals
- **Compression of master LP and object pools**
  - Reduce size of master LP, improve efficiency of subproblem processing

Other Considerations

- **Integration of generic MILP cuts** - *new idea*
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s \hat{\lambda}_s$ we can use generic MILP cuts in RC/PC context
- **Initial columns**
  - Solve $\mathrm{OPT}(\mathcal{P}', c + r)$ for random perturbations
  - Solve $\mathrm{OPT}(\mathcal{P}_N)$ heuristically
  - Run several iterations of LD or DC collecting extreme points
- **Price-and-branch heuristic**
  - For block-angular case, at end of each node, solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
- **Choice of master LP solver**
  - Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
  - Primal simplex after adding columns (warm-start primal feasible)
  - Interior-point methods might help with stabilization vs extremal duals
- **Compression of master LP and object pools**
  - Reduce size of master LP, improve efficiency of subproblem processing

Other Considerations

- **Integration of generic MILP cuts** - *new idea*
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP cuts in RC/PC context
- **Initial columns**
  - Solve $\mathrm{OPT}(\mathcal{P}', c + r)$ for random perturbations
  - Solve $\mathrm{OPT}(\mathcal{P}_N)$ heuristically
  - Run several iterations of LD or DC collecting extreme points
- **Price-and-branch heuristic**
  - For block-angular case, at end of each node, solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
- **Choice of master LP solver**
  - Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
  - Primal simplex after adding columns (warm-start primal feasible)
  - Interior-point methods might help with stabilization vs extremal duals
- **Compression of master LP and object pools**
  - Reduce size of master LP, improve efficiency of subproblem processing

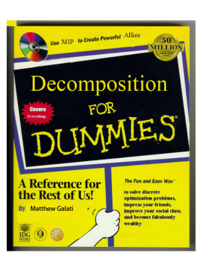
Outline

DIP Framework

DIP Framework

DIP (**D**ecomposition for **I**nteger **P**rogramming) is an open-source software framework that provides an implementation of various decomposition methods with minimal user responsibility

- Allows direct comparison CPM/DW/LD/PC/RC/DC in one framework
- DIP abstracts the common, generic elements of these methods
- Key: The user defines application-specific components in the space of the compact formulation - greatly simplifying the API
  - Define $[A'', b'']$ and/or $[A', b']$
  - Provide methods for $\mathrm{OPT}(\mathcal{P}', c)$ and/or $\mathrm{SEP}(\mathcal{P}', x)$
- Framework handles all of the algorithm-specific reformulation

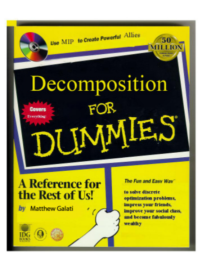


DIP Framework

DIP Framework

DIP (**D**ecomposition for **I**nteger **P**rogramming) is an open-source software framework that provides an implementation of various decomposition methods with minimal user responsibility

- Allows direct comparison CPM/DW/LD/PC/RC/DC in one framework
- DIP abstracts the common, generic elements of these methods
- Key: The user defines application-specific components in the space of the compact formulation - greatly simplifying the API
  - Define $[A'', b'']$ and/or $[A', b']$
  - Provide methods for $\mathrm{OPT}(\mathcal{P}', c)$ and/or $\mathrm{SEP}(\mathcal{P}', x)$
- Framework handles all of the algorithm-specific reformulation

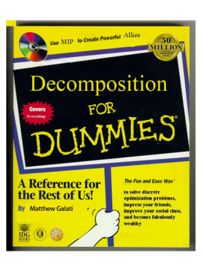


DIP Framework

DIP Framework

DIP (**D**ecomposition for **I**nteger **P**rogramming) is an open-source software framework that provides an implementation of various decomposition methods with minimal user responsibility

- Allows direct comparison CPM/DW/LD/PC/RC/DC in one framework
- DIP abstracts the common, generic elements of these methods
- **Key:** The user defines application-specific components in the space of the compact formulation - greatly simplifying the API
  - Define $[A'', b'']$ and/or $[A', b']$
  - Provide methods for $\mathrm{OPT}(\mathcal{P}', c)$ and/or $\mathrm{SEP}(\mathcal{P}', x)$
- Framework handles all of the algorithm-specific reformulation



DIP Framework: Implementation

COmputational INfrastructure for Operations Research
Have some DIP with your CHiPPs?

- **DIP** was built around data structures and interfaces provided by COIN-OR
- The **DIP** framework, written in C++, is accessed through two user interfaces:
  - Applications Interface: `DecompApp`
  - Algorithms Interface: `DecompAlgo`
- **DIP** provides the bounding method for branch and bound
- **ALPS** (Abstract Library for Parallel Search) provides the framework for tree search
  - `AlpsDecompModel : public AlpsModel`
    - a wrapper class that calls (data access) methods from `DecompApp`
  - `AlpsDecompTreeNode : public AlpsTreeNode`
    - a wrapper class that calls (algorithmic) methods from `DecompAlgo`

DIP Framework: Implementation

COmputational INfrastructure for Operations Research

Have some DIP with your CHiPPs?

- **DIP** was built around data structures and interfaces provided by COIN-OR
- The **DIP** framework, written in C++, is accessed through two user interfaces:
  - Applications Interface: `DecompApp`
  - Algorithms Interface: `DecompAlgo`
- **DIP** provides the bounding method for branch and bound
- **ALPS** (Abstract Library for Parallel Search) provides the framework for tree search
  - `AlpsDecompModel : public AlpsModel`
    - a wrapper class that calls (data access) methods from `DecompApp`
  - `AlpsDecompTreeNode : public AlpsTreeNode`
    - a wrapper class that calls (algorithmic) methods from `DecompAlgo`

DIP Framework: Implementation

COmputational INfrastructure for Operations Research

Have some DIP with your CHiPPs?

- **DIP** was built around data structures and interfaces provided by COIN-OR
- The **DIP** framework, written in C++, is accessed through two user interfaces:
  - Applications Interface: `DecompApp`
  - Algorithms Interface: `DecompAlgo`
- **DIP** provides the bounding method for branch and bound
- **ALPS** (Abstract Library for Parallel Search) provides the framework for tree search
  - `AlpsDecompModel : public AlpsModel`
    - a wrapper class that calls (data access) methods from `DecompApp`
  - `AlpsDecompTreeNode : public AlpsTreeNode`
    - a wrapper class that calls (algorithmic) methods from `DecompAlgo`

DIP - Creating an Application

- The base class `DecompApp` provides an interface for user to define the application-specific components of their algorithm
- Define the model(s)
  - `setModelObjective(double * c)`: define c
  - `setModelCore(DecompConstraintSet * model)`: define $\mathcal{Q}''$
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define $\mathcal{Q}'$ [optional]
- `solveRelaxed()`: define a method for $\mathrm{OPT}(\mathcal{P}', c)$ [optional, if $\mathcal{Q}'$, CBC is built-in]
- `generateCuts()`: define a method for $\mathrm{SEP}(\mathcal{P}', x)$ [optional, CGL is built-in]
- `isUserFeasible()`: is $\hat{x} \in \mathcal{P}$? [optional, if $\mathcal{P} = \mathrm{conv}(\mathcal{P}' \cap \mathcal{Q}'' \cap \mathbb{Z})$]
- All other methods have appropriate defaults but are `virtual` and may be overridden

DIP - Creating an Application

- The base class DecompApp provides an interface for user to define the application-specific components of their algorithm
- Define the model(s)
  - `setModelObjective(double * c)`: define c
  - `setModelCore(DecompConstraintSet * model)`: define $\mathcal{Q}''$
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define $\mathcal{Q}'$ [optional]
- `solveRelaxed()`: define a method for $\mathrm{OPT}(\mathcal{P}', c)$ [optional, if $\mathcal{Q}'$, CBC is built-in]
- `generateCuts()`: define a method for $\mathrm{SEP}(\mathcal{P}', x)$ [optional, CGL is built-in]
- `isUserFeasible()`: is $\hat{x} \in \mathcal{P}$? [optional, if $\mathcal{P} = \mathrm{conv}(\mathcal{P}' \cap \mathcal{Q}'' \cap \mathbb{Z})$]
- All other methods have appropriate defaults but are `virtual` and may be overridden

DIP - Creating an Application

- The base class `DecompApp` provides an interface for user to define the application-specific components of their algorithm
- Define the model(s)
  - `setModelObjective(double * c)`: define c
  - `setModelCore(DecompConstraintSet * model)`: define $\mathcal{Q}''$
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define $\mathcal{Q}'$ [optional]
- `solveRelaxed()`: define a method for $\mathrm{OPT}(\mathcal{P}', c)$ [optional, if $\mathcal{Q}'$, **CBC** is built-in]
- `generateCuts()`: define a method for $\mathrm{SEP}(\mathcal{P}', x)$ [optional, **CGL** is built-in]
- `isUserFeasible()`: is $\hat{x} \in \mathcal{P}$? [optional, if $\mathcal{P} = \mathrm{conv}(\mathcal{P}' \cap \mathcal{Q}'' \cap \mathbb{Z})$]
- All other methods have appropriate defaults but are `virtual` and may be overridden

DIP - Creating an Application

- The base class `DecompApp` provides an interface for user to define the application-specific components of their algorithm
- Define the model(s)
  - `setModelObjective(double * c)`: define c
  - `setModelCore(DecompConstraintSet * model)`: define $\mathcal{Q}''$
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define $\mathcal{Q}'$ [optional]
- `solveRelaxed()`: define a method for $\mathrm{OPT}(\mathcal{P}', c)$ [optional, if $\mathcal{Q}'$, **CBC** is built-in]
- `generateCuts()`: define a method for $\mathrm{SEP}(\mathcal{P}', x)$ [optional, **CGL** is built-in]
- `isUserFeasible()`: is $\hat{x} \in \mathcal{P}$? [optional, if $\mathcal{P} = \mathrm{conv}(\mathcal{P}' \cap \mathcal{Q}'' \cap \mathbb{Z})$]
- All other methods have appropriate defaults but are `virtual` and may be overridden

DIP Framework: Compare and Contrast to COIN/BCP

```
int main(int argc, char ** argv){
   //create the utility class for parsing parameters
   UtilParameters utilParam(argc, argv);
   bool doCut          = utilParam.GetSetting("doCut",       true);
   bool doPriceCut     = utilParam.GetSetting("doPriceCut",  false);
   bool doRelaxCut     = utilParam.GetSetting("doRelaxCut",  false);

   //create the user application (a DecompApp)
   SILP_DecompApp sip(utilParam);

   //create the CPM/PC/RC algorithm objects (a DecompAlgo)
   DecompAlgo    * algo = NULL;
   if(doCut)       algo = new DecompAlgoC (&sip, &utilParam);
   if(doPriceCut)  algo = new DecompAlgoPC(&sip, &utilParam);
   if(doRelaxCut)  algo = new DecompAlgoRC(&sip, &utilParam);

   //create the driver AlpsDecomp model
   AlpsDecompModel alpsModel(utilParam, algo);

   //solve
   alpsModel.solve();
}
```

DIP Framework: Compare and Contrast to COIN/BCP

- Limitations:
  - **BCP**: The user must derive methods for almost all of the algorithmic components: (master reformulation, expansion of rows and columns, branching in reformulated space, calculation of pricing mechanisms like reduced cost, etc).
  - **DIP**: There exists a compact formulation which forms the basis of the model attributes.
- Design:
  - **BCP**: The user defines the model attributes and algorithmic components based on one predefined solution *method* (i.e., PC or CPM).
  - **DIP**: The user defines the model attributes and algorithmic components based on one predefined compact *formulation*. The different algorithmic details are managed by the framework.
- Parallelization:
  - **BCP**: Designed for shared or distributed memory for branch-and-bound search.
  - **DIP**: Threaded for block-angular shared memory processing.
  - **DIP**: Built on top of Alps so potential for fully distributed branch-and-bound search (*in the future*).

DIP Framework: Compare and Contrast to COIN/BCP

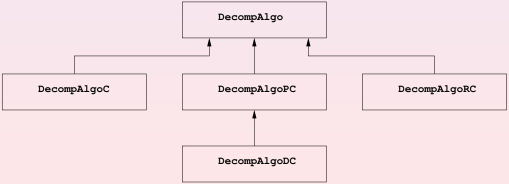
- Limitations:
  - **BCP**: The user must derive methods for almost all of the algorithmic components: (master reformulation, expansion of rows and columns, branching in reformulated space, calculation of pricing mechanisms like reduced cost, etc).
  - **DIP**: There exists a compact formulation which forms the basis of the model attributes.
- Design:
  - **BCP**: The user defines the model attributes and algorithmic components based on one predefined solution *method* (i.e., PC or CPM).
  - **DIP**: The user defines the model attributes and algorithmic components based on one predefined compact *formulation*. The different algorithmic details are managed by the framework.
- Parallelization:
  - **BCP**: Designed for shared or distributed memory for branch-and-bound search.
  - **DIP**: Threaded for block-angular shared memory processing.
  - **DIP**: Built on top of Alps so potential for fully distributed branch-and-bound search (*in the future*).

DIP Framework: Compare and Contrast to COIN/BCP

- Limitations:
  - **BCP**: The user must derive methods for almost all of the algorithmic components: (master reformulation, expansion of rows and columns, branching in reformulated space, calculation of pricing mechanisms like reduced cost, etc).
  - **DIP**: There exists a compact formulation which forms the basis of the model attributes.
- Design:
  - **BCP**: The user defines the model attributes and algorithmic components based on one predefined solution *method* (i.e., PC or CPM).
  - **DIP**: The user defines the model attributes and algorithmic components based on one predefined compact *formulation*. The different algorithmic details are managed by the framework.
- Parallelization:
  - **BCP**: Designed for shared or distributed memory for branch-and-bound search.
  - **DIP**: Threaded for block-angular shared memory processing.
  - **DIP**: Built on top of Alps so potential for fully distributed branch-and-bound search (*in the future*).

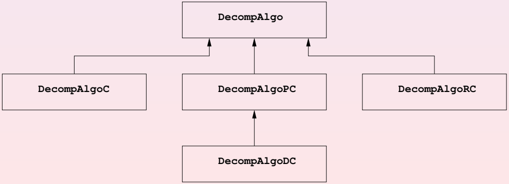
DIP - Algorithms

- The base class DecompAlgo provides the shell (init / master / subproblem / update).
- Each of the methods described has derived default implementations DecompAlgoX : public DecompAlgo which are accessible by any application class, allowing full flexibility.
- New, hybrid or extended methods can be easily derived by overriding the various subroutines, which are called from the base class. For example,
  - Alternative methods for solving the master LP in DW, such as interior point methods
  - Add stabilization to the dual updates in LD (stability centers)
  - For LD, replace subgradient with volume providing an approximate primal solution
  - Hybrid init methods like using LD or DC to initialize the columns of the DW master
  - During PC, adding cuts to either master and/or subproblem.
  - ...



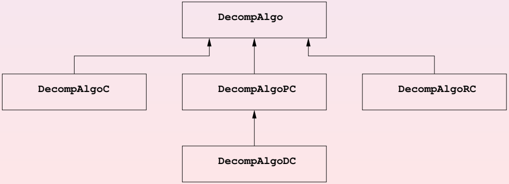
DIP - Algorithms

- The base class DecompAlgo provides the shell (init / master / subproblem / update).
- Each of the methods described has derived default implementations `DecompAlgoX : public DecompAlgo` which are accessible by any application class, allowing full flexibility.
- New, hybrid or extended methods can be easily derived by overriding the various subroutines, which are called from the base class. For example,
  - Alternative methods for solving the master LP in DW, such as interior point methods
  - Add stabilization to the dual updates in LD (stability centers)
  - For LD, replace subgradient with volume providing an approximate primal solution
  - Hybrid init methods like using LD or DC to initialize the columns of the DW master
  - During PC, adding cuts to either master and/or subproblem.
  - ...



DIP - Algorithms

- The base class `DecompAlgo` provides the shell (init / master / subproblem / update).
- Each of the methods described has derived default implementations `DecompAlgoX : public DecompAlgo` which are accessible by any application class, allowing full flexibility.
- New, hybrid or extended methods can be easily derived by overriding the various subroutines, which are called from the base class. For example,
  - Alternative methods for solving the master LP in DW, such as **interior point methods**
  - Add stabilization to the dual updates in LD (stability centers)
  - For LD, replace subgradient with **volume** providing an approximate primal solution
  - Hybrid init methods like using LD or DC to initialize the columns of the DW master
  - During PC, adding cuts to either master and/or subproblem.
  - ...



DIP - Example Applications

Application	Description	$\mathcal{P}'$	$\mathbf{OPT}(c)$	$\mathbf{SEP}(x)$	Input
AP3	3-index assignment	AP	Jonker	user	user
ATM	cash management (SAS COE)	MILP(s)	CBC	CGL	user
GAP	generalized assignment	KP(s)	Pisinger	CGL	user
MAD	matrix decomposition	MaxClique	Cliquer	CGL	user
MILP	random partition into A', A''	MILP	CBC	CGL	mps
MILPBlock	user-defined blocks for A'	MILP(s)	CBC	CGL	mps, block
MMKP	multi-dim/choice knapsack	MCKP	Pisinger	CGL	user
		MDKP	CBC	CGL	user
SILP	intro example, tiny IP	MILP	CBC	CGL	user
TSP	traveling salesman problem	1-Tree	Boost	Concorde	user
		2-Match	CBC	Concorde	user
VRP	vehicle routing problem	k-TSP	Concorde	CVRPSEP	user
		b-Match	CBC	CVRPSEP	user

Outline

Multi-Choice Multi-Dimensional Knapsack Problem (MMKP)

- **SAS Marketing Optimization** - improve ROI for marketing campaign offers by targeting higher response rates, improving channel effectiveness, and reduce spending.

$$
\begin{array}{llll}
\max & \sum_{i\in N}\sum_{j\in L_i} v_{ij}x_{ij} & & \\
& \sum_{i\in N}\sum_{j\in L_i} r_{kij}x_{ij} \leq b_k & & \forall k \in M \\
& \sum_{j\in L_i} x_{ij} = 1 & & \forall i \in N \\
& x_{ij} \in \{0,1\} & & \forall i \in N, j \in L_i
\end{array}
$$

- Relaxation - Multi-Choice Knapsack Problem (MCKP)
  - solver *mcknap* by Pisinger a DP-based branch-and-bound

$$
\begin{array}{lll}
\sum_{i\in N}\sum_{j\in L_i} r_{mij}x_{ij} \leq b_m & & \\
\sum_{j\in L_i} x_{ij} = 1 & & \forall i \in N \\
x_{ij} \in \{0,1\} & & \forall i \in N, j \in L_i
\end{array}
$$

Multi-Choice Multi-Dimensional Knapsack Problem (MMKP)

- **SAS Marketing Optimization** - improve ROI for marketing campaign offers by targeting higher response rates, improving channel effectiveness, and reduce spending.

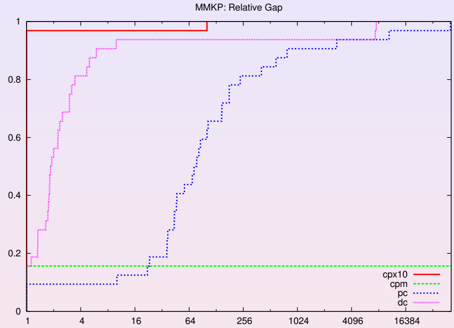
$$
\begin{array}{llll}
\max & \sum_{i \in N} \sum_{j \in L_i} v_{ij} x_{ij} & & \\
& \sum_{i \in N} \sum_{j \in L_i} r_{kij} x_{ij} \leq & b_k & \forall k \in M \\
& \sum_{j \in L_i} x_{ij} = & 1 & \forall i \in N \\
& x_{ij} \in & \{0,1\} & \forall i \in N, j \in L_i
\end{array}
$$

- Relaxation - Multi-Choice Knapsack Problem (MCKP)
  - solver *mcknap* by Pisinger a DP-based branch-and-bound

$$
\begin{array}{lll}
\sum_{i \in N} \sum_{j \in L_i} r_{mij} x_{ij} \leq & b_m & \\
\sum_{j \in L_i} x_{ij} = & 1 & \forall i \in N \\
x_{ij} \in & \{0,1\} & \forall i \in N, j \in L_i
\end{array}
$$

MMKP: CPX10.2 vs CPM/PC/DC

	CPX10.2		DIP-CPM		DIP-PC		DIP-DC	
Instance	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**
I1	0.00	OPT	0.02	OPT	0.04	OPT	0.14	OPT
I10	T	0.05%	T	∞	T	11.86%	T	0.15%
I11	T	0.03%	T	∞	T	12.25%	T	0.14%
I12	T	0.01%	T	∞	T	7.93%	T	0.10%
I13	T	0.02%	T	∞	T	11.89%	T	0.12%
I2	0.01	OPT	0.01	OPT	0.05	OPT	0.05	OPT
I3	1.17	OPT	23.23	OPT	T	1.07%	T	0.75%
I4	15.71	OPT	T	∞	T	5.14%	T	0.77%
I5	0.01	0.01%	0.01	OPT	0.13	OPT	0.05	OPT
I6	0.14	OPT	0.07	OPT	T	0.28%	0.63	OPT
I7	T	0.08%	T	∞	T	14.32%	T	0.09%
I8	T	0.09%	T	∞	T	13.36%	T	0.20%
I9	T	0.06%	T	∞	T	10.71%	T	0.19%
INST01	T	0.43%	T	∞	T	9.99%	T	0.70%
INST02	T	0.09%	T	∞	T	7.39%	T	0.45%
INST03	T	0.38%	T	∞	T	3.83%	T	0.85%
INST04	T	0.34%	T	∞	T	7.48%	T	0.45%
INST05	T	0.18%	T	∞	T	10.23%	T	0.62%
INST06	T	0.21%	T	∞	T	9.82%	T	0.38%
INST07	T	0.36%	T	∞	T	15.75%	T	0.62%
INST08	T	0.25%	T	∞	T	11.55%	T	0.46%
INST09	T	0.21%	T	∞	T	15.24%	T	0.40%
INST11	T	0.22%	T	∞	T	7.96%	T	0.39%
INST12	T	0.18%	T	∞	T	7.90%	T	0.42%
INST13	T	0.08%	T	∞	T	2.97%	T	0.14%
INST14	T	0.05%	T	∞	T	3.89%	T	0.09%
INST15	T	0.04%	T	∞	T	3.43%	T	0.10%
INST16	T	0.06%	T	∞	T	2.19%	T	0.06%
INST17	T	0.03%	T	∞	T	2.09%	T	0.09%
INST18	T	0.03%	T	∞	T	4.43%	T	0.06%
INST19	T	0.03%	T	∞	T	3.13%	T	0.04%
INST20	T	0.03%	T	∞	T	3.05%	T	0.04%

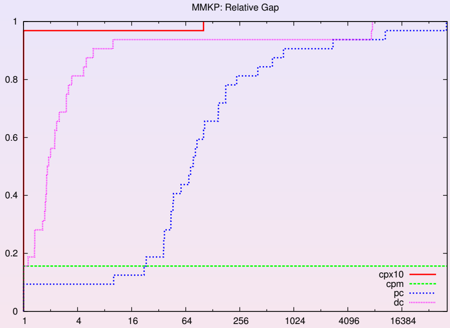


	CPX10.2	DIP-CPM	DIP-PC	DIP-DC
Optimal	5	5	3	4
$\leq$ **1% Gap**	32	5	4	32
$\leq$ **10% Gap**	32	5	22	32

CGL: missing *Gub Covers*

MMKP: CPX10.2 vs CPM/PC/DC

	CPX10.2		DIP-CPM		DIP-PC		DIP-DC	
Instance	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**
I1	0.00	OPT	0.02	OPT	0.04	OPT	0.14	OPT
I10	T	0.05%	T	∞	T	11.86%	T	0.15%
I11	T	0.03%	T	∞	T	12.25%	T	0.14%
I12	T	0.01%	T	∞	T	7.93%	T	0.10%
I13	T	0.02%	T	∞	T	11.89%	T	0.12%
I2	0.01	OPT	0.01	OPT	0.05	OPT	0.05	OPT
I3	1.17	OPT	23.23	OPT	T	1.07%	T	0.75%
I4	15.71	OPT	T	∞	T	5.14%	T	0.77%
I5	0.01	0.01%	0.01	OPT	0.13	OPT	0.05	OPT
I6	0.14	OPT	0.07	OPT	T	0.28%	0.63	OPT
I7	T	0.08%	T	∞	T	14.32%	T	0.09%
I8	T	0.09%	T	∞	T	13.36%	T	0.20%
I9	T	0.06%	T	∞	T	10.71%	T	0.19%
INST01	T	0.43%	T	∞	T	9.99%	T	0.70%
INST02	T	0.09%	T	∞	T	7.39%	T	0.45%
INST03	T	0.38%	T	∞	T	3.83%	T	0.85%
INST04	T	0.34%	T	∞	T	7.48%	T	0.45%
INST05	T	0.18%	T	∞	T	10.23%	T	0.62%
INST06	T	0.21%	T	∞	T	9.82%	T	0.38%
INST07	T	0.36%	T	∞	T	15.75%	T	0.62%
INST08	T	0.25%	T	∞	T	11.55%	T	0.46%
INST09	T	0.21%	T	∞	T	15.24%	T	0.40%
INST11	T	0.22%	T	∞	T	7.96%	T	0.39%
INST12	T	0.18%	T	∞	T	7.90%	T	0.42%
INST13	T	0.08%	T	∞	T	2.97%	T	0.14%
INST14	T	0.05%	T	∞	T	3.89%	T	0.09%
INST15	T	0.04%	T	∞	T	3.43%	T	0.10%
INST16	T	0.06%	T	∞	T	2.19%	T	0.06%
INST17	T	0.03%	T	∞	T	2.09%	T	0.09%
INST18	T	0.03%	T	∞	T	4.43%	T	0.06%
INST19	T	0.03%	T	∞	T	3.13%	T	0.04%
INST20	T	0.03%	T	∞	T	3.05%	T	0.04%



	CPX10.2	DIP-CPM	DIP-PC	DIP-DC
Optimal	5	5	3	4
$\leq$ **1% Gap**	32	5	4	32
$\leq$ **10% Gap**	32	5	22	32

CGL: missing *Gub Covers*

MMKP: Nested Pricing

- Nested Relaxations:
  - Multi-Choice 2-D Knapsack Problem (MC2KP): $\mathcal{P}_p^{\mathrm{MC2KP}} \subset \mathcal{P}^{\mathrm{MCKP}}\ \forall p \in M \setminus \{m\}$

$$\begin{array}{lcll}
\sum_{i \in N} \sum_{j \in L_i} r_{pij} x_{ij} & \leq & b_p & \\
\sum_{i \in N} \sum_{j \in L_i} r_{mij} x_{ij} & \leq & b_m & \\
\sum_{j \in L_i} x_{ij} & = & 1 & \forall i \in N \\
x_{ij} & \in & \{0,1\} & \forall i \in N, j \in L_i
\end{array}$$

  - Multi-Choice Multi-Dimensional Knapsack Problem (MMKP): $\mathcal{P} \subset \mathcal{P}^{\mathrm{MCKP}}$

MMKP: Nested Pricing

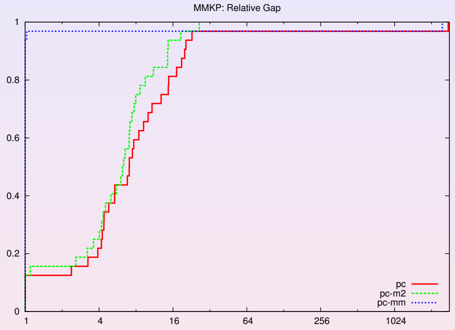
- Nested Relaxations:
  - Multi-Choice 2-D Knapsack Problem (MC2KP): $\mathcal{P}_p^{\text{MC2KP}} \subset \mathcal{P}^{\text{MCKP}} \; \forall p \in M \setminus \{m\}$

$$\begin{array}{lcll} \sum_{i \in N} \sum_{j \in L_i} r_{pij} x_{ij} & \leq & b_p & \\ \sum_{i \in N} \sum_{j \in L_i} r_{mij} x_{ij} & \leq & b_m & \\ \sum_{j \in L_i} x_{ij} & = & 1 & \forall i \in N \\ x_{ij} & \in & \{0,1\} & \forall i \in N, j \in L_i \end{array}$$

  - Multi-Choice Multi-Dimensional Knapsack Problem (MMKP): $\mathcal{P} \subset \mathcal{P}^{\text{MCKP}}$

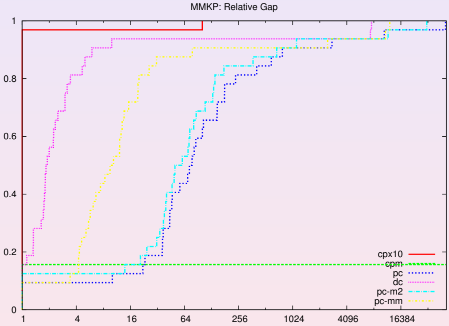
MMKP: PC vs PC Nested with MC2KP and MMKP

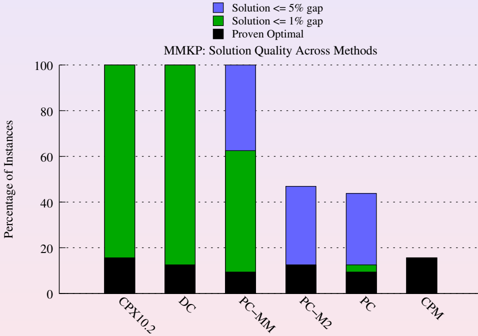
	DIP-PC		DIP-PC-M2		DIP-PC-MM	
Instance	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**
I1	0.04	OPT	0.16	OPT	0.08	OPT
I10	T	11.86%	T	6.99%	T	0.63%
I11	T	12.25%	T	11.15%	T	0.60%
I12	T	7.93%	T	11.41%	T	0.79%
I13	T	11.89%	T	13.65%	T	0.52%
I2	0.05	OPT	0.45	OPT	0.14	OPT
I3	T	1.07%	T	1.18%	T	1.10%
I4	T	5.14%	T	3.18%	T	1.23%
I5	0.13	OPT	0.14	OPT	0.07	OPT
I6	T	0.28%	483.53	OPT	T	0.25%
I7	T	14.32%	T	4.85%	T	0.97%
I8	T	13.36%	T	9.79%	T	0.67%
I9	T	10.71%	T	10.57%	T	0.73%
INST01	T	9.99%	T	5.97%	T	1.86%
INST02	T	7.39%	T	7.29%	T	1.74%
INST03	T	3.83%	T	11.93%	T	1.61%
INST04	T	7.48%	T	7.04%	T	1.56%
INST05	T	10.23%	T	8.84%	T	1.11%
INST06	T	9.82%	T	9.77%	T	1.39%
INST07	T	15.75%	T	8.78%	T	1.23%
INST08	T	11.55%	T	8.50%	T	1.37%
INST09	T	15.24%	T	8.48%	T	0.89%
INST11	T	7.96%	T	8.72%	T	1.13%
INST12	T	7.90%	T	6.72%	T	1.03%
INST13	T	2.97%	T	3.06%	T	0.76%
INST14	T	3.89%	T	3.67%	T	0.52%
INST15	T	3.43%	T	2.81%	T	0.78%
INST16	T	2.19%	T	3.01%	T	0.50%
INST17	T	2.09%	T	2.16%	T	0.39%
INST18	T	4.43%	T	2.60%	T	0.41%
INST19	T	3.13%	T	3.97%	T	0.46%
INST20	T	3.05%	T	4.06%	T	0.94%



	DIP-PC	DIP-PC-M2	DIP-PC-MM
Optimal	3	4	3
$\leq$ 1% Gap	4	4	20
$\leq$ 10% Gap	22	27	32

MMKP: CPX10.2 vs CPM/PC/DC/PC-M2/PC-MM





ATM Cash Management Problem - Business Problem

SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- Cash allocation plans finalized at beginning of month - deviations from plan are costly
- **Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals
- **Constraints:**
  - Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
  - For each ATM, limit on number of days *cash-out* based on predictive model (customer satisfaction)

ATM Cash Management Problem - Business Problem

SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- Cash allocation plans finalized at beginning of month - deviations from plan are costly
- **Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals
- **Constraints:**
  - Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
  - For each ATM, limit on number of days *cash-out* based on predictive model (customer satisfaction)

ATM Cash Management Problem - Business Problem

SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- Cash allocation plans finalized at beginning of month - deviations from plan are costly
- **Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals
- **Constraints:**
  - Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
  - For each ATM, limit on number of days *cash-out* based on predictive model (customer satisfaction)

ATM Cash Management Problem - Business Problem

SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- Cash allocation plans finalized at beginning of month - deviations from plan are costly
- **Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals
- **Constraints:**
  - Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
  - For each ATM, limit on number of days *cash-out* based on predictive model (customer satisfaction)

ATM Cash Management Problem - MINLP Formulation

- Simple *looking* nonconvex quadratic integer NLP.
- Linearize the absolute value, add binaries for count constraints.
- So far, no MINLP solvers seem to be able to solve this (several die with numerical failures).

$$
\begin{array}{lll}
\min \sum_{a \in A} \sum_{d \in D} |f_{ad}| & & \\
\text{s.t. } c^x_{ad} x_a + c^y_{ad} y_a + c^{xy}_{ad} x_a y_a + c^u_{ad} u_a + c_{ad} - w_{ad} & = f_{ad} & \forall a \in A, d \in D \\
\sum_{a \in A} (f_{ad} + w_{ad}) & \leq B_d & \forall d \in D \\
|\{d \in D \mid f_{ad} < 0\}| & \leq K_a & \forall a \in A \\
x_a, y_a & \in [0,1] & \forall a \in A \\
u_a & \geq 0 & \forall a \in A \\
f_{ad} & \geq -w_{ad} & \forall a \in A, d \in D
\end{array}
$$

Application - ATM Cash Management Problem - MILP Approx Formulation

- Discretization of x domain $\{0, 0.1, 0.2, ..., 1.0\}$.
- Linearization of product of binary and continuous, and absolute value.

$$
\begin{array}{llll}
\min & \sum_{a \in A} \sum_{d \in D} \left(f^+_{ad} + f^-_{ad} \right) & & \\
\text{s.t.} & c^x_{ad} \sum_{t \in T} c_t x_{at} + c^y_{ad} y_a + c^{xy}_{ad} \sum_{t \in T} c_t z_{at} + c^u_{ad} u_a - w_{ad} & = f^+_{ad} - f^-_{ad} & \forall a \in A, d \in D \\
& \sum_{t \in T} x_{at} & \leq 1 & \forall a \in A \\
& z_{at} & \leq x_{at} & \forall a \in A, t \in T \\
& z_{at} & \leq y_a & \forall a \in A, t \in T \\
& z_{at} & \geq x_{at} + y_a - 1 & \forall a \in A, t \in T \\
& f^-_{ad} & \leq w_{ad} v_{ad} & \forall a \in A, d \in D \\
& \sum_{a \in A} (f^+_{ad} - f^-_{ad} + w_{ad}) & \leq B_d & \forall d \in D \\
& \sum_{d \in D} v_{ad} & \leq K_a & \forall a \in A
\end{array}
$$

ATM Cash Management Problem - MILP Approx Formulation

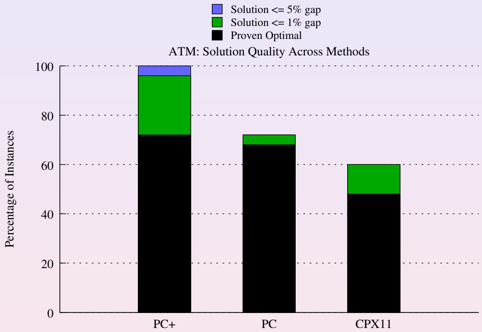
$$
\begin{array}{lll}
x_{at} & \in \{0,1\} & \forall a \in A, t \in T \\
z_{at} & \geq 0 & \forall a \in A, t \in T \\
v_{ad} & \in \{0,1\} & \forall a \in A, d \in D \\
y_{a} & \in [0,1] & \forall a \in A \\
u_{a} & \geq 0 & \forall a \in A \\
f^{+}_{ad}, f^{-}_{ad} & \in [0, w_{ad}] & \forall a \in A, d \in D
\end{array}
$$

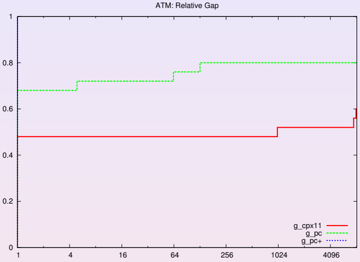
- The MILP formulation has a natural block-angular structure.
  - Master constraints are just the budget constraint.
  - Subproblem constraints (*the rest*) - one block for each ATM.

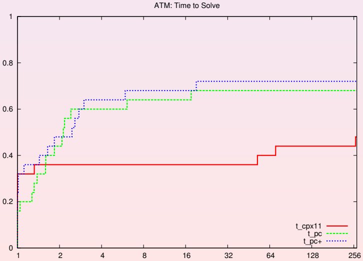
ATM: CPX11 vs PC/PC+

			CPX11			DIP-PC			DIP-PC+		
$\|A\|$	$\|D\|$	s	Time	Gap	Nodes	Time	Gap	Nodes	Time	Gap	Nodes
5	25	1	0.76	OPT	467	1.62	OPT	6	1.96	OPT	6
5	25	2	1.41	OPT	804	1.95	OPT	9	1.57	OPT	7
5	25	3	0.42	OPT	147	7.38	OPT	32	8.03	OPT	32
5	25	4	1.49	OPT	714	2.74	OPT	14	2.45	OPT	13
5	25	5	0.16	OPT	32	0.98	OPT	7	0.95	OPT	6
5	50	1	T	0.10	1264574	162.74	OPT	127	164.46	OPT	131
5	50	2	87.96	OPT	38341	183.28	OPT	273	263.24	OPT	275
5	50	3	8.09	OPT	3576	17.58	OPT	36	22.28	OPT	35
5	50	4	4.13	OPT	1317	3.13	OPT	3	3.17	OPT	3
5	50	5	57.55	OPT	32443	91.30	OPT	145	141.29	OPT	147
10	50	1	T	0.76	998624	297.65	OPT	301	234.47	OPT	156
10	50	2	1507.84	OPT	351879	28.84	OPT	29	52.99	OPT	29
10	50	3	T	0.81	667371	64.72	OPT	64	49.20	OPT	47
10	50	4	1319.00	OPT	433155	7.97	OPT	1	5.00	OPT	1
10	50	5	365.51	OPT	181013	12.49	OPT	3	5.18	OPT	3
10	100	1	T	∞	128155	T	∞	20590	T	0.11	13190
10	100	2	T	∞	116522	T	∞	60554	2437.43	OPT	135
10	100	3	T	∞	118617	T	∞	52902	T	0.20	40793
10	100	4	T	∞	108899	T	∞	47931	T	1.51	59477
10	100	5	T	∞	167617	T	∞	40283	T	0.38	26490
20	100	1	T	∞	93519	379.75	OPT	9	544.49	OPT	9
20	100	2	T	∞	68863	T	16.44	14240	T	0.26	25756
20	100	3	T	∞	95981	T	15.37	41495	T	0.12	3834
20	100	4	T	∞	81836	T	0.39	7554	T	0.08	7918
20	100	5	T	∞	101917	635.59	OPT	21	608.68	OPT	19
Optimal				12			17			18	
$\leq$ 1% Gap				15			18			25	
$\leq$ 10% Gap				15			18			25	

ATM: CPX11 vs PC/PC+







MILPBlock - Block-Angular MILP (as a Generic Solver)

- Consulting work led to numerous MILPs that cannot be solved with generic (B&C) solvers
- Often consider a decomposition approach, since a common modeling paradigm is
  - independent departmental policies which are then coupled by some global constraints
- Development time was slow due to problem-specific implementations of methods

$$\begin{pmatrix} A''_1 & A''_2 & \cdots & A''_\kappa \\ \hline A'_1 & & & \\ & A'_2 & & \\ & & \ddots & \\ & & & A'_\kappa \end{pmatrix}$$

- MILPBlock provides a black-box solver for applying integrated methods to generic MILP
  - This is the *first* framework to do this (to my knowledge).
  - Similar efforts are being talked about by F. Vanderbeck BaPCod (no cuts)
- Currently, the only input needed is MPS/LP and a *block file*
- Future work will attempt to embed automatic recognition of the block-angular structure using packages from linear algebra like: MONET, hMETIS, Mondriaan

MILPBlock - Block-Angular MILP (as a Generic Solver)

- Consulting work led to numerous MILPs that cannot be solved with generic (B&C) solvers
- Often consider a decomposition approach, since a common modeling paradigm is
  - independent departmental policies which are then coupled by some global constraints
- Development time was slow due to problem-specific implementations of methods

$$\left(\begin{array}{cccc} A_1'' & A_2'' & \cdots & A_\kappa'' \\ \hline A_1' & & & \\ & A_2' & & \\ & & \ddots & \\ & & & A_\kappa' \end{array}\right)$$

- MILPBlock provides a black-box solver for applying integrated methods to generic MILP
  - This is the *first* framework to do this (to my knowledge).
  - Similar efforts are being talked about by F. Vanderbeck **BaPCod** (no cuts)
- Currently, the only input needed is MPS/LP and a *block file*
- Future work will attempt to embed automatic recognition of the block-angular structure using packages from linear algebra like: MONET, hMETIS, Mondriaan

Application - Block-Angular MILP (applied to Retail Optimization)

SAS Retail Optimization Solution

- *Multi-tiered supply chain distribution problem* where each block represents a store
- Prototype model developed in SAS/OR's OPTMODEL (algebraic modeling language)

	CPX11			DIP-PC		
Instance	**Time**	**Gap**	**Nodes**	**Time**	**Gap**	**Nodes**
retail27	T	2.30%	2674921	3.18	OPT	1
retail31	T	0.49%	1434931	767.36	OPT	41
retail3	529.77	OPT	2632157	0.54	OPT	1
retail4	T	1.61%	1606911	116.55	OPT	1
retail6	1.12	OPT	803	264.59	OPT	303

Outline

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- Convergence issues and **stabilization** of duals (*stability centers*)
- Can we implement **Gomory cuts** in Price-and-Cut?
  - Similar to Interior Point crossover to Simplex, we can crossover from $\hat{x}$ to a feasible basis, load that into the solver and generate tableau cuts
  - Will the design of OSI and CGL work like this? **YES**. J Forrest has added a `crossover` to OsiClp
- Other generic MILP techniques for **MILPBlock**: heuristics, branching strategies, presolve
- DIP support for **identical subproblems** (using Vanderbeck's ideas)
- **Parallelization** of branch-and-bound
  - More work per node, communication overhead low - use ALPS
- **Parallelization** related to relaxed polyhedra (work-in-progress):
  - Pricing in block-angular case
  - Nested pricing - use idle cores to generate diverse set of columns simultaneously
  - Generation of decomposition cuts for various relaxed polyhedra - diversity of cuts

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- Convergence issues and **stabilization** of duals (*stability centers*)
- Can we implement **Gomory cuts** in Price-and-Cut?
  - Similar to Interior Point crossover to Simplex, we can crossover from $\hat{x}$ to a feasible basis, load that into the solver and generate tableau cuts
  - Will the design of OSI and CGL work like this? **YES.** J Forrest has added a `crossover` to OsiClp
- Other generic MILP techniques for **MILPBlock**: heuristics, branching strategies, presolve
- DIP support for **identical subproblems** (using Vanderbeck's ideas)
- **Parallelization** of branch-and-bound
  - More work per node, communication overhead low - use ALPS
- **Parallelization** related to relaxed polyhedra (work-in-progress):
  - Pricing in block-angular case
  - Nested pricing - use idle cores to generate diverse set of columns simultaneously
  - Generation of decomposition cuts for various relaxed polyhedra - diversity of cuts

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- Convergence issues and **stabilization** of duals (*stability centers*)
- Can we implement **Gomory cuts** in Price-and-Cut?
  - Similar to Interior Point crossover to Simplex, we can crossover from $\hat{x}$ to a feasible basis, load that into the solver and generate tableau cuts
  - Will the design of OSI and CGL work like this? **YES.** J Forrest has added a `crossover` to OsiClp
- Other generic MILP techniques for **MILPBlock**: heuristics, branching strategies, presolve
- DIP support for **identical subproblems** (using Vanderbeck's ideas)
- **Parallelization** of branch-and-bound
  - More work per node, communication overhead low - use ALPS
- **Parallelization** related to relaxed polyhedra (work-in-progress):
  - Pricing in block-angular case
  - Nested pricing - use idle cores to generate diverse set of columns simultaneously
  - Generation of decomposition cuts for various relaxed polyhedra - diversity of cuts

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- Convergence issues and **stabilization** of duals (*stability centers*)
- Can we implement **Gomory cuts** in Price-and-Cut?
  - Similar to Interior Point crossover to Simplex, we can crossover from $\hat{x}$ to a feasible basis, load that into the solver and generate tableau cuts
  - Will the design of OSI and CGL work like this? **YES.** J Forrest has added a `crossover` to OsiClp
- Other generic MILP techniques for **MILPBlock**: heuristics, branching strategies, presolve
- DIP support for **identical subproblems** (using Vanderbeck's ideas)
- **Parallelization** of branch-and-bound
  - More work per node, communication overhead low - use ALPS
- **Parallelization** related to relaxed polyhedra (work-in-progress):
  - Pricing in block-angular case
  - Nested pricing - use idle cores to generate diverse set of columns simultaneously
  - Generation of decomposition cuts for various relaxed polyhedra - diversity of cuts

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- Convergence issues and **stabilization** of duals (*stability centers*)
- Can we implement **Gomory cuts** in Price-and-Cut?
  - Similar to Interior Point crossover to Simplex, we can crossover from $\hat{x}$ to a feasible basis, load that into the solver and generate tableau cuts
  - Will the design of OSI and CGL work like this? **YES.** J Forrest has added a `crossover` to OsiClp
- Other generic MILP techniques for **MILPBlock**: heuristics, branching strategies, presolve
- DIP support for **identical subproblems** (using Vanderbeck's ideas)
- Parallelization of branch-and-bound
  - More work per node, communication overhead low - use ALPS
- Parallelization related to relaxed polyhedra (work-in-progress):
  - Pricing in block-angular case
  - Nested pricing - use idle cores to generate diverse set of columns simultaneously
  - Generation of decomposition cuts for various relaxed polyhedra - diversity of cuts

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- Convergence issues and **stabilization** of duals (*stability centers*)
- Can we implement **Gomory cuts** in Price-and-Cut?
  - Similar to Interior Point crossover to Simplex, we can crossover from $\hat{x}$ to a feasible basis, load that into the solver and generate tableau cuts
  - Will the design of OSI and CGL work like this? **YES.** J Forrest has added a `crossover` to OsiClp
- Other generic MILP techniques for **MILPBlock**: heuristics, branching strategies, presolve
- DIP support for **identical subproblems** (using Vanderbeck's ideas)
- **Parallelization** of branch-and-bound
  - More work per node, communication overhead low - use ALPS
- **Parallelization** related to relaxed polyhedra (work-in-progress):
  - Pricing in block-angular case
  - Nested pricing - use idle cores to generate diverse set of columns simultaneously
  - Generation of decomposition cuts for various relaxed polyhedra - diversity of cuts

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**.
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**.
  - Integrated methods: **price-and-cut** and **relax-and-cut**.
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation**.
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(***D***ecomposition for* **I***nteger* **P***rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**.
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**.
  - Integrated methods: **price-and-cut** and **relax-and-cut**.
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(***D***ecomposition for* **I***nteger* **P***rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**.
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**.
  - Integrated methods: **price-and-cut** and **relax-and-cut**.
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(***D***ecomposition for* **I***nteger* **P***rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
    - Traditional method for outer approximation: **cutting plane method**.
    - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**.
    - Integrated methods: **price-and-cut** and **relax-and-cut**.
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(**D**ecomposition for **I**nteger **P**rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
    - Traditional method for outer approximation: **cutting plane method**.
    - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**.
    - Integrated methods: **price-and-cut** and **relax-and-cut**.
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(**D**ecomposition for **I**nteger **P**rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*

Thesis Contributions

- *Conceptual framework tying together numerous decomposition-based methods for generating approximations of the convex hull of feasible solutions.*
  - Traditional method for outer approximation: **cutting plane method**.
  - Traditional methods for inner approximations: **Dantzig-Wolfe method** and **Lagrangian method**.
  - Integrated methods: **price-and-cut** and **relax-and-cut**.
- *Introduction to a relatively new integrated method called* **decompose-and-cut**, *an associated class of cutting planes called* **decomposition cuts**, *and the concept of* **structured separation.**
- *Descriptions of numerous implementation considerations for branch-and-price-and-cut, including an introduction to a relatively unknown idea of using* **nested polyhedra** *for generating inner approximations.*
- **DIP** *(**D**ecomposition for **I**nteger **P**rogramming), an extensible open-source software framework for implementing decomposition-based methods with minimal user burden.*
- *MILPBlock, a DIP application and generic black-box solver for* **block-diagonal** *MILPs that fully automates the branch-and-price-and-cut algorithm with no additional user input.*
- *Computational results using DIP on three real-world* **applications** *coming from the marketing, banking, and retail industries.*