DIP with CHiPPS:
Decomposition Methods for Integer Linear Programming

(Or, Things I've Been Musing About Since I left Cornell...)

TED RALPHS
LEHIGH UNIVERSITY
MATTHEW GALATI
SAS INSTITUTE
YAN XU
SAS INSTITUTE

COR@L
COMPUTATIONAL OPTIMIZATION RESEARCH AT LEHIGH

Cornell University, April 12 2011

Thanks: Work supported in part by the National Science Foundation

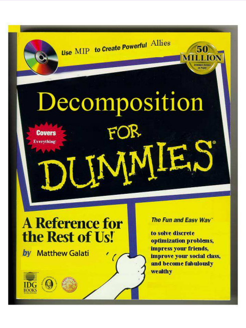
Apples from the Family Tree



- This is work that grew out of several of the three main themes from my dissertation.
- It has now produced three subsequent dissertations.
  - Matthew Galati, *Decomposition in Integer Programming* ⇐ main focus of this talk
  - Yan Xu, *Scalable Algorithms for Parallel Tree Search*
  - Zeliha Akca, *Integrated Location, Routing, and Scheduling Problems*

Overview

- *Decomposition* has long been known as a powerful paradigm for the solution of structured integer programs.
- Its application in practice is hindered by a number of serious drawbacks.
  - *Implementation is difficult*, usually requiring development of sophisticated customized codes.
  - Choosing an algorithmic strategy *requires in-depth knowledge* of theory and strategies are *difficult to compare empirically*.
  - The powerful techniques modern solvers use to solve integer programs are *difficult to integrate* with decomposition-based approaches.
- SYMPHONY was a framework for easily developing customized versions of branch and cut.
- DIP and CHiPPS are two new frameworks that generalize many of the ideas from SYMPHONY
  - CHiPPS (COIN High Performance Parallel Search Software) is a flexible library hierarchy for implementing parallel search algorithms.
  - DIP (Decomposition for Integer Programs) is a framework for implementing decomposition-based bounding methods.
  - DIP with CHiPPS is a full-blown branch-and-cut-and-price framework in which details of the implementation are hidden from the user.



Outline

Outline

The Decomposition Principle in Integer Programming

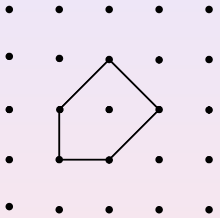
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

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

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



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

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

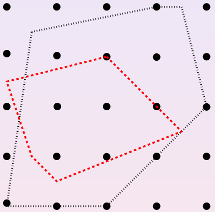
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

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

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



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

$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$ (dashed red line)

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

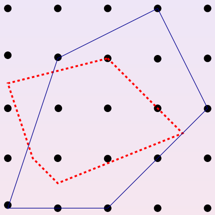
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

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

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



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

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

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

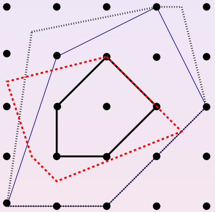
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

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

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



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

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

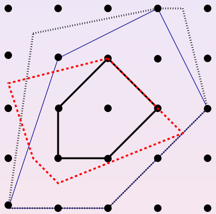
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

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

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



- (solid thick line) $\mathcal{P} = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b''\}$
- (solid thin line) $\mathcal{P}' = \mathrm{conv}\{x \in \mathbb{Z}^n \mid A'x \geq b'\}$
- (dotted line) $\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \geq b'\}$
- (red dashed line) $\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \geq b''\}$

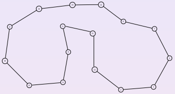
Assumptions:

- $\mathrm{OPT}(\mathcal{P}, c)$ 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

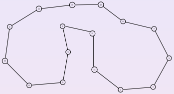
$$
\begin{array}{lcll}
x(\delta(\{u\})) & = & 2 & \forall u \in V \\
x(E(S)) & \leq & |S| - 1 & \forall S \subset V,\ 3 \leq |S| \leq |V| - 1 \\
x_e \in \{0,1\} & & & \forall e \in E
\end{array}
$$



Example - Traveling Salesman Problem (TSP)

Traveling Salesman Problem Formulation

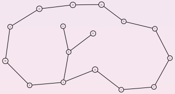
$$\begin{array}{lcll} x(\delta(\{u\})) & = & 2 & \forall u \in V \\ x(E(S)) & \leq & |S| - 1 & \forall S \subset V,\ 3 \leq |S| \leq |V| - 1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Two relaxations

Find a spanning subgraph with $|V|$ edges ($\mathcal{P}' =$ 1-Tree)

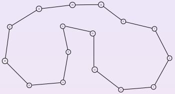
$$\begin{array}{lcll} x(\delta(\{0\})) & = & 2 & \\ x(E(V)) & = & |V| & \\ x(E(S)) & \leq & |S| - 1 & \forall S \subset V \setminus \{0\}, 3 \leq |S| \leq |V| - 1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Example - Traveling Salesman Problem (TSP)

Traveling Salesman Problem Formulation

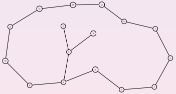
$$\begin{array}{rcll} x(\delta(\{u\})) & = & 2 & \forall u \in V \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V,\ 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Two relaxations

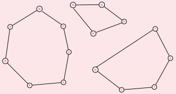
Find a spanning subgraph with $|V|$ edges ($\mathcal{P}' =$ 1-Tree)

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



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

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



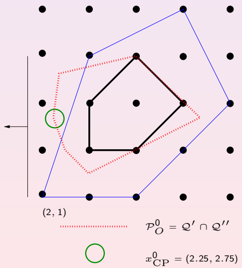
Cutting Plane Method (CPM)

CPM combines an *outer* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

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

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

Exponential number of constraints



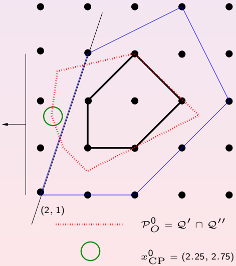
Cutting Plane Method (CPM)

CPM combines an *outer* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

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

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

Exponential number of constraints



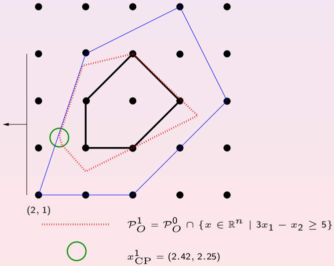
Cutting Plane Method (CPM)

CPM combines an *outer* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **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



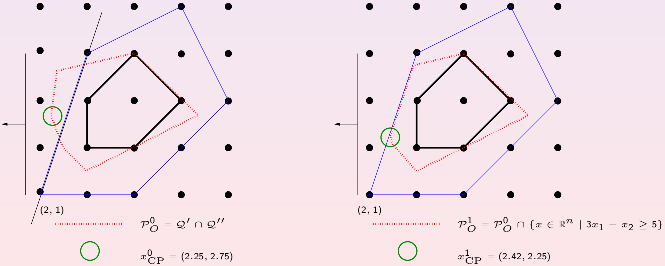
Cutting Plane Method (CPM)

CPM combines an *outer* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

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

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

Exponential number of constraints



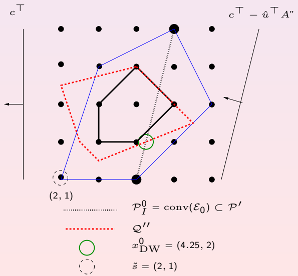
Dantzig-Wolfe Method (DW)

DW combines an *inner* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **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



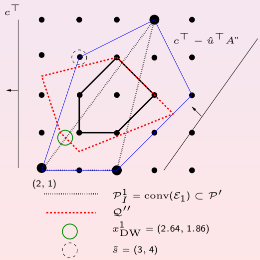
Dantzig-Wolfe Method (DW)

DW combines an *inner* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **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



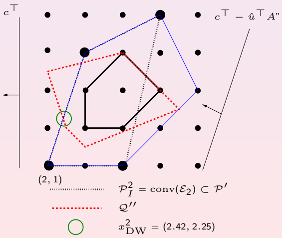
Dantzig-Wolfe Method (DW)

DW combines an *inner* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **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



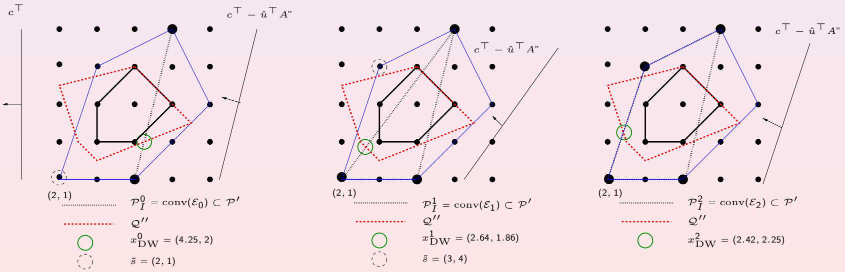
Dantzig-Wolfe Method (DW)

DW combines an *inner* approximation of $\mathcal{P}'$ with an explicit description of $\mathcal{Q}''$

- **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

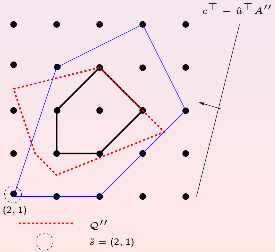


Lagrangian Method (LD)

LD iteratively produces single extreme points of $\mathcal{P}'$ and uses their violation of constraints of $\mathcal{Q}''$ to converge to the same optimal face of $\mathcal{P}'$ as CPM and DW.

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

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

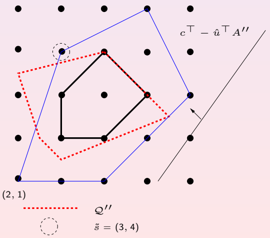


Lagrangian Method (LD)

LD iteratively produces single extreme points of $\mathcal{P}'$ and uses their violation of constraints of $\mathcal{Q}''$ to converge to the same optimal face of $\mathcal{P}'$ as CPM and DW.

- **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**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u_{\mathrm{LD}}^\top A''\right)$

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

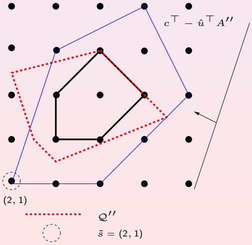


Lagrangian Method (LD)

LD iteratively produces single extreme points of $\mathcal{P}'$ and uses their violation of constraints of $\mathcal{Q}''$ to converge to the same optimal face of $\mathcal{P}'$ as CPM and DW.

- **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**: $\mathrm{OPT}\left(\mathcal{P}', c^\top - u^\top_{\mathrm{LD}} A''\right)$

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

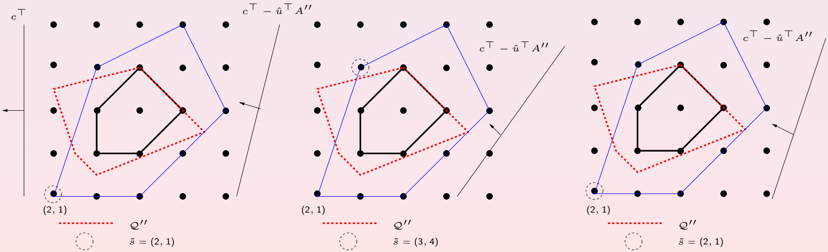


Lagrangian Method (LD)

LD iteratively produces single extreme points of $\mathcal{P}'$ and uses their violation of constraints of $\mathcal{Q}''$ to converge to the same optimal face of $\mathcal{P}'$ as CPM and DW.

- **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

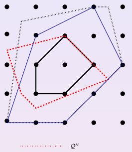
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

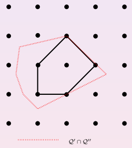
$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$

- 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

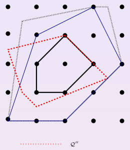
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

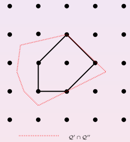
$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$

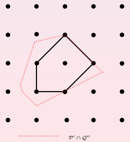
- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

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

- 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

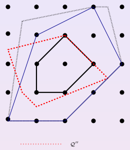
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

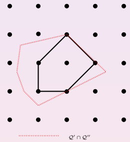
$$z_{\mathrm{LP}} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$

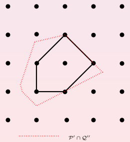
- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

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

- 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)





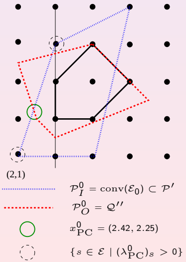


Price-and-Cut Method (PC)

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.

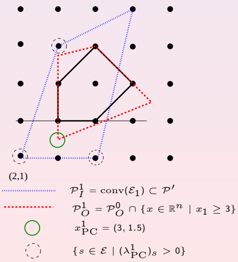


Price-and-Cut Method (PC)

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.

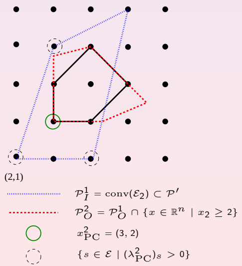


Price-and-Cut Method (PC)

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.

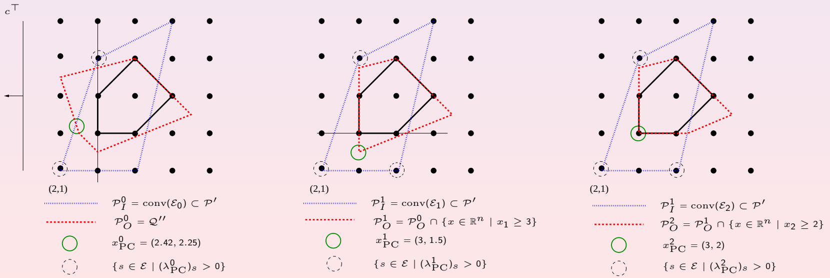


Price-and-Cut Method (PC)

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

- As in CPM, separate $\hat{x}_{\text{PC}} = \sum_{s\in\mathcal{E}} s\hat{\lambda}_s$ 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.

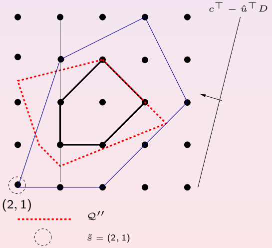


Relax-and-Cut Method (RC)

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 $\hat{x} \in \mathbb{R}^n$.

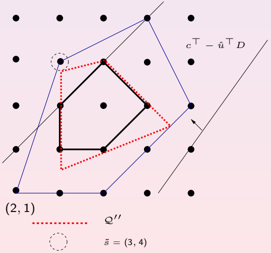


Relax-and-Cut Method (RC)

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**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}^{m''}_{+}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **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 $\hat{x} \in \mathbb{R}^n$.

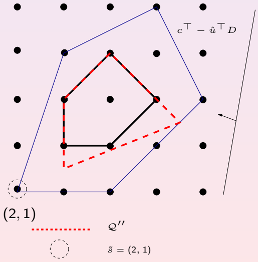


Relax-and-Cut Method (RC)

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 $\hat{x} \in \mathbb{R}^n$.

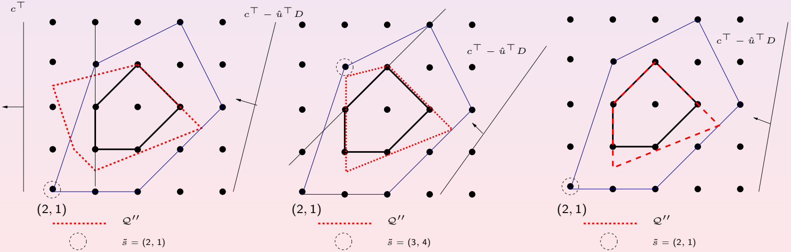


Relax-and-Cut Method (RC)

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**: $z_{\mathrm{LD}} = \max_{u \in \mathbb{R}_+^{m''}} \{\min_{s \in \mathcal{E}} \{c^\top s + u^\top (d - Ds)\}\}$
- **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, $\mathrm{OPT}(X, c)$ and $\mathrm{SEP}(X, x)$ are polynomially equivalent.
- **Observation:** Restrictions on input or output can change their complexity.
- The 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, $\mathrm{OPT}(X, c)$ and $\mathrm{SEP}(X, x)$ are polynomially equivalent.
- **Observation:** Restrictions on input or output can change their complexity.
- The 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

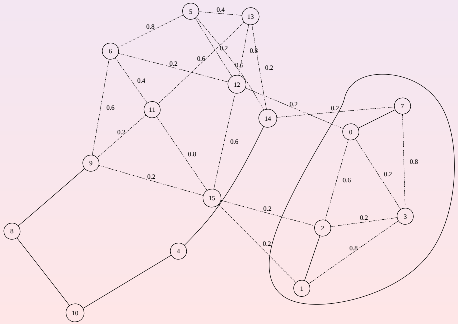
- In general, $\mathrm{OPT}(X, c)$ and $\mathrm{SEP}(X, x)$ are polynomially equivalent.
- **Observation:** Restrictions on input or output can change their complexity.
- The Template Paradigm, restricts the *output* of $\mathrm{SEP}(X, x)$ to valid inequalities that conform to a certain structure. This 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: Example

- 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).



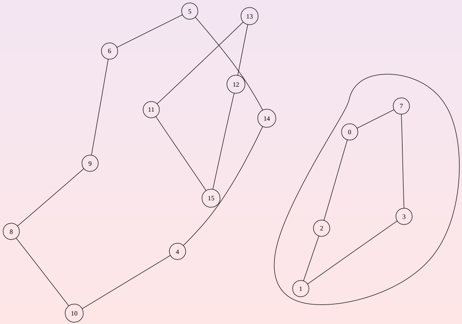
Structured Separation: Example

- 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).

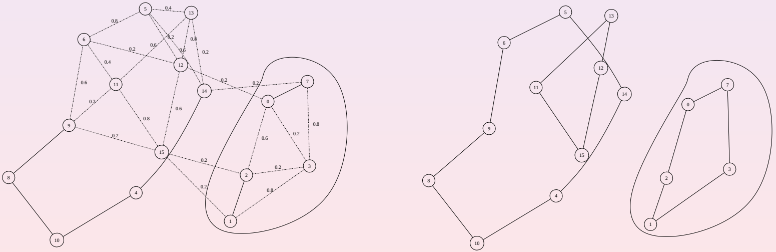


Structured Separation: Example

- 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).

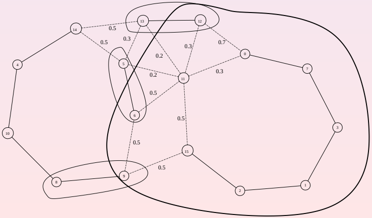


Structured Separation: Example

- Separation of Comb Inequalities:

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

- $\mathrm{SEP}(\mathcal{P}^{\mathrm{Blossom}}, x)$, 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.



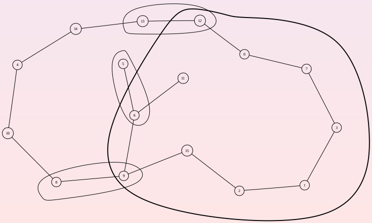
Structured Separation: Example

- 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.

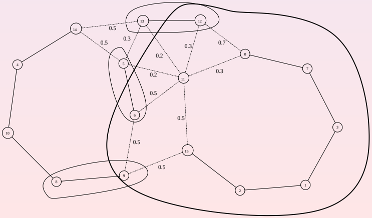


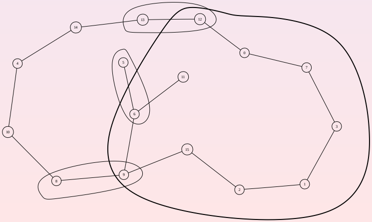
Structured Separation: Example

- 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.

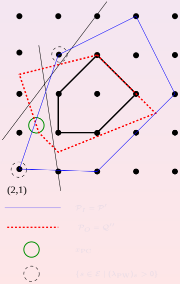




Price-and-Cut (Revisited)

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**.

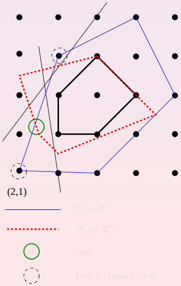
- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with RC, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- RC only gives us one member of $\mathcal{E}$ to separate, while PC gives us a set, 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**.

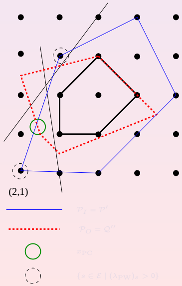
- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with **RC**, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- **RC** only gives us one member of $\mathcal{E}$ to separate, while **PC** 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**.

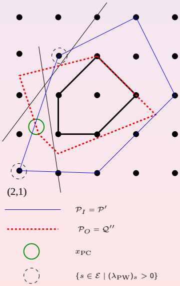
- **Key Idea:** Rather than (or in addition to) separating $\hat{x}_{\mathrm{PC}}$, separate each member of D
- As with **RC**, often much easier to separate $s \in \mathcal{E}$ than $\hat{x}_{\mathrm{PC}} \in \mathbb{R}^n$
- **RC** only gives us one member of $\mathcal{E}$ to separate, while **PC** 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)

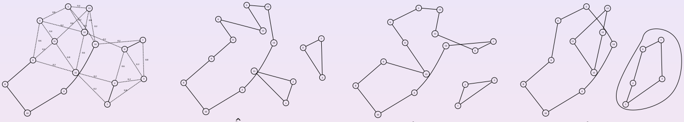
Price-and-Cut (Revisited): As normal, use **DW** as the bounding method, but use the decomposition obtained in each iteration to generate improving inequalities, as in **RC**.

- **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.

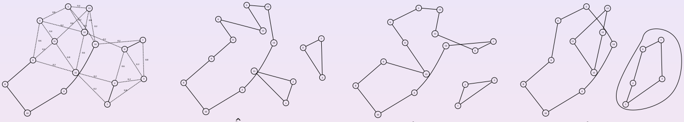


$\hat{x}$ $\quad$ $\hat{\lambda}_0 = 0.2$ $\quad$ $\hat{\lambda}_1 = 0.2$ $\quad$ $\hat{\lambda}_2 = 0.6$

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

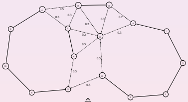
Price-and-Cut (Revisited)

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

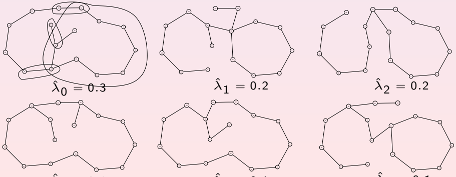


$\hat{x}$ $\hat{\lambda}_0 = 0.2$ $\hat{\lambda}_1 = 0.2$ $\hat{\lambda}_2 = 0.6$

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



$\hat{x}$



$\hat{\lambda}_0 = 0.3$ $\hat{\lambda}_1 = 0.2$ $\hat{\lambda}_2 = 0.2$

$\hat{\lambda}_3 = 0.1$ $\hat{\lambda}_4 = 0.1$ $\hat{\lambda}_5 = 0.1$

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)

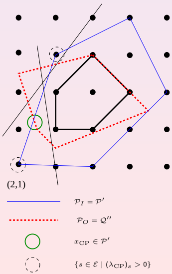
Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.'s of $\mathcal{P}'$

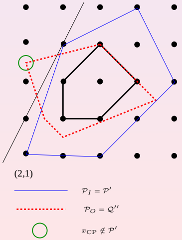
$$\min_{\lambda \in \mathbb{R}^{\mathcal{E}}_+, (x^+, x^-) \in \mathbb{R}^n_+} \left\{ x^+ + x^- \;\middle|\; \sum_{s \in \mathcal{E}} s\lambda_s + x^+ - x^- = \hat{x}_{\mathrm{CP}}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}$$

- 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

$$\begin{aligned} u^t_{\mathrm{DC}} s + \alpha^t_{\mathrm{DC}} &\leq 0 \; \forall s \in \mathcal{P}' \quad \text{and} \\ u^t_{\mathrm{DC}} \hat{x}_{\mathrm{CP}} + \alpha^t_{\mathrm{DC}} &> 0 \end{aligned}$$





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 we are missing some facets of $\mathcal{P}'$ in our current relaxation.
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because it's a *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 we are missing some facets of $\mathcal{P}'$ in our current relaxation.
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because it's a *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 we are missing some facets of $\mathcal{P}'$ in our current relaxation.
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because it's a *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 we are missing some facets of $\mathcal{P}'$ in our current relaxation.
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because it's a *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 we are missing some facets of $\mathcal{P}'$ in our current relaxation.
- The machinery for solving this already exists (=column generation)
- Much easier than DW problem because it's a *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)

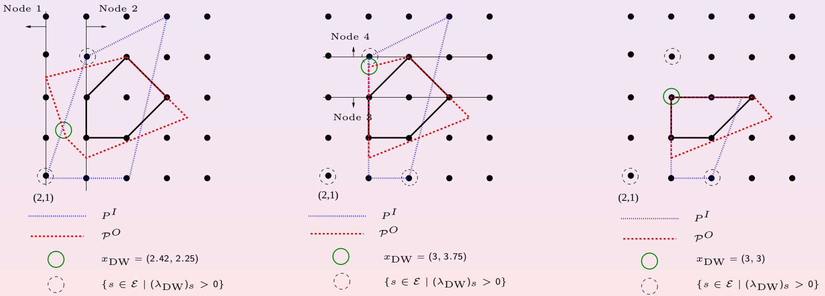
- Add column bounds to $[A'', b'']$ and map back to the compact space $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$
- Variable branching in the compact space is constraint branching in the extended space
- This idea takes care of (most of) the design issues related to branching for inner methods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

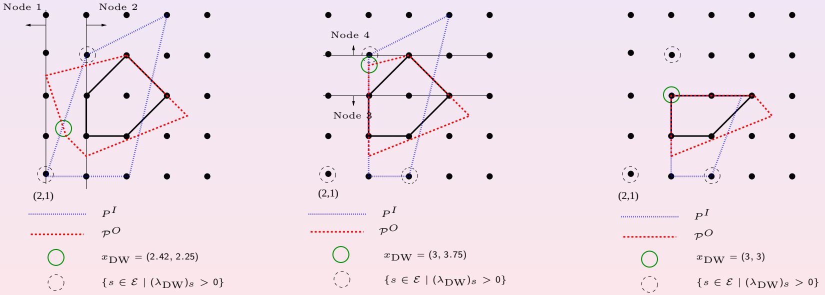
Branching for Inner Methods (PC)

- 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.



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

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

Algorithmic Details and Extensions

- **Separable subproblems** (Important!)
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
- **Use of generic MILP solution technology**
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP generation in RC/PC context
  - Use generic MILP solver to solve subproblems.
  - With automatic block decomposition can allow solution of generic MILPs with no customization!
- **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

Algorithmic Details and Extensions

- **Separable subproblems** (Important!)
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
- **Use of generic MILP solution technology**
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP generation in RC/PC context
  - Use generic MILP solver to solve subproblems.
  - With automatic block decomposition can allow solution of generic MILPs with no customization!
- **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

Algorithmic Details and Extensions

- **Separable subproblems** (Important!)
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
- **Use of generic MILP solution technology**
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP generation in RC/PC context
  - Use generic MILP solver to solve subproblems.
  - With automatic block decomposition can allow solution of generic MILPs with no customization!
- **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

Algorithmic Details and Extensions

- **Separable subproblems** (Important!)
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
- **Use of generic MILP solution technology**
  - Using the mapping $\hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s$ we can use generic MILP generation in RC/PC context
  - Use generic MILP solver to solve subproblems.
  - With automatic block decomposition can allow solution of generic MILPs with no customization!
- **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

Algorithmic Details and Extensions (cont.)

- **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
- **Nested pricing**
  - Can solve more constrained versions of subproblem heuristically to get high quality columns.

Algorithmic Details and Extensions (cont.)

- **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
- **Nested pricing**
  - Can solve more constrained versions of subproblem heuristically to get high quality columns.

Algorithmic Details and Extensions (cont.)

- **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
- **Nested pricing**
  - Can solve more constrained versions of subproblem heuristically to get high quality columns.

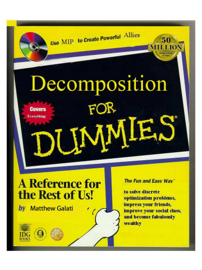
Outline

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

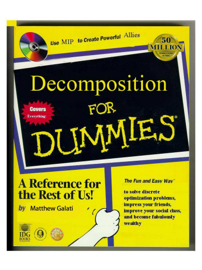


DIP Framework

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

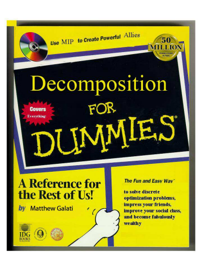


DIP Framework

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 Q''
  - setModelRelaxed(DecompConstraintSet * model, int block): define Q' [optional]
- solveRelaxed(): define a method for $\mathrm{OPT}(\mathcal{P}', c)$ [optional, if 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 $\mathcal{P} = \mathrm{conv}(\mathcal{P}' \cap Q'' \cap \mathbb{Z})$]
- 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

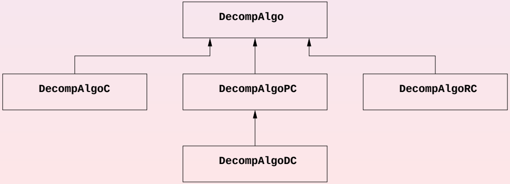
- The base class DecompApp provides an interface for user to define the application-specific components of their algorithm
- Define the model(s)
  - `setModelObjective(double * c)`: define c
  - `setModelCore(DecompConstraintSet * model)`: define $\mathcal{Q}''$
  - `setModelRelaxed(DecompConstraintSet * 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: Example Code

```
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();
}
```

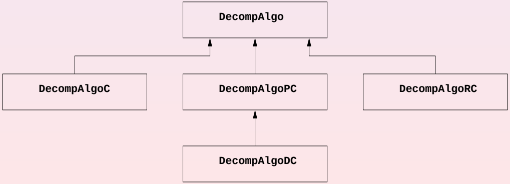
DIP Framework: Algorithms

- 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.
  - ...



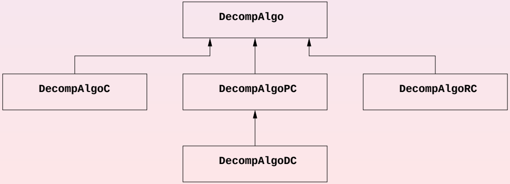
DIP Framework: Algorithms

- 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 Framework: 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 Framework: 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

Quick Introduction to CHiPPS

- CHiPPS stands for COIN-OR High Performance Parallel Search.
- CHiPPS is a set of C++ class libraries for implementing tree search algorithms for both sequential and parallel environments.

CHiPPS Components (Current)

ALPS (Abstract Library for Parallel Search)

- is the search-handling layer (parallel and sequential).
- provides various search strategies based on node priorities.

BiCePS (Branch, Constrain, and Price Software)

- is the data-handling layer for relaxation-based optimization.
- adds notion of variables and constraints.
- assumes iterative bounding process.

BLIS (BiCePS Linear Integer Solver)

- is a concretization of BiCePS.
- specific to models with linear constraints and objective function.

ALPS: Design Goals

- Intuitive object-oriented class structure.
  - `AlpsModel`
  - `AlpsTreeNode`
  - `AlpsNodeDesc`
  - `AlpsSolution`
  - `AlpsParameterSet`
- Minimal algorithmic assumptions in the base class.
  - Support for a wide range of problem classes and algorithms.
  - Support for constraint programming.
- Easy for user to develop a custom solver.
- Design for *parallel scalability*, but operate effectively in a sequential environment.
- Explicit support for *memory compression* techniques (packing/differencing) important for implementing optimization algorithms.

ALPS: Overview of Features

- The design is based on a very general concept of *knowledge*.
- Knowledge is shared asynchronously through *pools* and *brokers*.
- Management overhead is reduced with the *master-hub-worker* paradigm.
- Overhead is decreased using dynamic task granularity.
- Two static load balancing techniques are used.
- Three dynamic load balancing techniques are employed.
- Uses asynchronous messaging to the highest extent possible.
- A scheduler on each process manages tasks like
  - node processing,
  - load balaning,
  - update search states, and
  - termination checking, etc.

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.

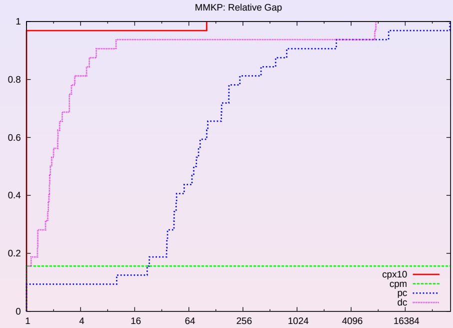
$$
\begin{array}{llll}
\max & \sum_{i \in N} \sum_{j \in L_i} v_{ij} x_{ij} & & \\
& \sum_{i \in N} \sum_{j \in L_i} r_{kij} x_{ij} \leq & b_k & \forall k \in M \\
& \sum_{j \in L_i} x_{ij} = & 1 & \forall i \in N \\
& x_{ij} \in & \{0,1\} & \forall i \in N, j \in L_i
\end{array}
$$

- 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%

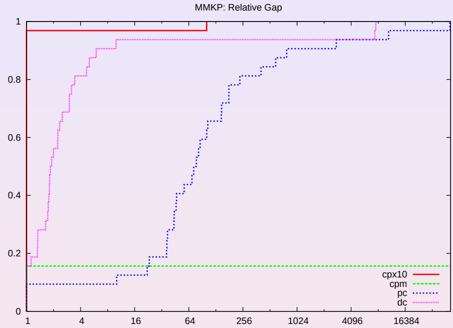


	CPX10.2	DIP-CPM	DIP-PC	DIP-DC
Optimal	5	5	3	4
$\leq$ 1% Gap	32	5	4	32
$\leq$ 10% Gap	32	5	22	32

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*

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)
- We can approximate with an MILP formulation that has a natural block-angular structure.
  - Master constraints are just the budget constraint.
  - Subproblem constraints (*the rest*) - one block for each ATM.

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)
- We can approximate with an MILP formulation that has a natural block-angular structure.
  - Master constraints are just the budget constraint.
  - Subproblem constraints (*the rest*) - one block for each ATM.

ATM Cash Management Problem - Business Problem

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

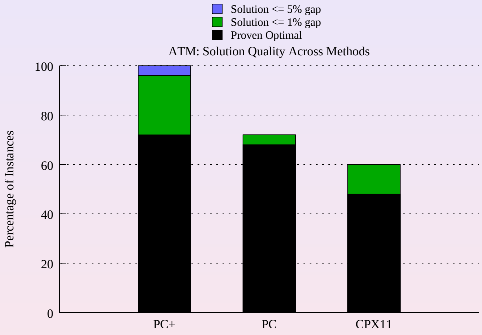
- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- 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)
- We can approximate with an MILP formulation that has a natural block-angular structure.
  - Master constraints are just the budget constraint.
  - Subproblem constraints (*the rest*) - one block for each ATM.

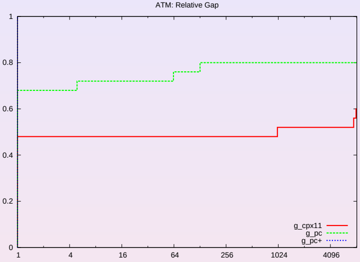
ATM Cash Management Problem - Business Problem

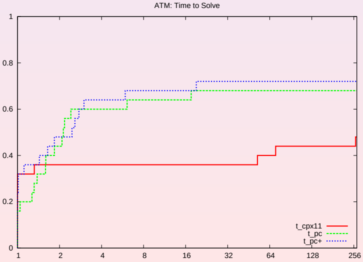
SAS Center of Excellence in Operations Research Applications (OR COE)

- 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)
- We can approximate with an MILP formulation that 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+







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

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

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

1. Decomposition Methods
   - Traditional Methods
   - Integrated Methods
   - Structured Separation
   - Decompose-and-Cut Method
   - Algorithmic Details
2. DIP
3. CHiPPS
4. Applications
   - Multi-Choice Multi-Dimensional Knapsack Problem
   - ATM Cash Management Problem
   - Generic Black-box Solver for Block-Angular MILP
5. Current and Future Research

MILPBlock: Recently Added Features

Interfaces for Pricing Algorithms (for IBM Project)

- User can provide an initial dual vector
- User can manipulate duals used at each pass (and specify per block)
- User can select which block to process next (alternative to *all* or *round-robin*)

New Options

- Branching can be auto enforced in subproblem or master (when oracle is MILP)
- Ability to stop subproblem calculation on gap/time and calculate LB (can branch early)
- For oracles that provide it, allow multiple columns for each subproblem call
- Management of compression of columns - once master gap is tight

Performance

- Detection and removal of columns that are close to parallel
- Added basic dual stabilization (Wentges smoothing)
- Redesign (and simplification) of treatment of master-only variables.

MILPBlock: Recently Added Features

Interfaces for Pricing Algorithms (for IBM Project)

- User can provide an initial dual vector
- User can manipulate duals used at each pass (and specify per block)
- User can select which block to process next (alternative to *all* or *round-robin*)

New Options

- Branching can be auto enforced in subproblem or master (when oracle is MILP)
- Ability to stop subproblem calculation on gap/time and calculate LB (can branch early)
- For oracles that provide it, allow multiple columns for each subproblem call
- Management of compression of columns - once master gap is tight

Performance

- Detection and removal of columns that are close to parallel
- Added basic dual stabilization (Wentges smoothing)
- Redesign (and simplification) of treatment of master-only variables.

MILPBlock: Recently Added Features

Interfaces for Pricing Algorithms (for IBM Project)

- User can provide an initial dual vector
- User can manipulate duals used at each pass (and specify per block)
- User can select which block to process next (alternative to *all* or *round-robin*)

New Options

- Branching can be auto enforced in subproblem or master (when oracle is MILP)
- Ability to stop subproblem calculation on gap/time and calculate LB (can branch early)
- For oracles that provide it, allow multiple columns for each subproblem call
- Management of compression of columns - once master gap is tight

Performance

- Detection and removal of columns that are close to parallel
- Added basic dual stabilization (Wentges smoothing)
- Redesign (and simplification) of treatment of master-only variables.

Related Projects Currently using DIP

- **OSDip** – Optimization Services (**OS**) wraps DIP (in CoinBazaar)
  - University of Chicago – Kipp Martin
- **Dippy** – Python interface for **DIP** through **PuLP**
  - University of Auckland – Michael O'Sullivan
- **SAS** – surface MILPBlock-like solver for `PROC OPTMODEL`
  - SAS Institute – Matthew Galati
- **Lehigh University** – Working on extensions to DIP including parallelism and automating the identification of block angular structure (missing piece for *black box MILP solver*)
  - Lehigh University – Jaidong Wang and Ted Ralphs
- **National Workforce Management, Cross-Training and Scheduling Project**
  - IBM Business Process Re-engineering – Alper Uygur
- **Transmission Switching Problem for Electricity Networks**
  - University of Denmark – Jonas Villumsem
  - University of Auckland – Andy Philipott

DIP@SAS in PROC OPTMODEL

- Prototype PC algorithm embedded in PROC OPTMODEL (based on MILPBlock)
- Minor API change - one new suffix on rows *or* cols (`.block`)

Preliminary Results (Recent Clients):

Client Problem	**IP-GAP** DIP@SAS	CPX12.1	**Real-Time** DIP@SAS	CPX12.1
ATM Cash Management and Predictive Model (India)	OPT	∞	103	2000 (T)
ATM Cash Management (Singapore)	OPT	OPT	86	831
	OPT	OPT	90	783
Retail Inventory Optimization (UK)	1.6%	9%	1200	1200 (T)
	4.7%	19%	1200	1200 (T)
	2.6%	∞	1200	1200 (T)

Future Research

- **Branch-and-Relax-and-Cut** - computational focus thus far has been on CPM/DC/PC
- 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
- Better support for **identical subproblems** (using ideas of Vanderbeck)
- **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
- 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
- Better support for **identical subproblems** (using ideas of Vanderbeck)
- **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
- 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
- Better support for **identical subproblems** (using ideas of Vanderbeck)
- **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
- 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
- Better support for **identical subproblems** (using ideas of Vanderbeck)
- **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
- 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
- Better support for **identical subproblems** (using ideas of Vanderbeck)
- **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
- 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
- Better support for **identical subproblems** (using ideas of Vanderbeck)
- **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