Decomposition Methods for Discrete Optimization

TED RALPHS
ANAHITA HASSANZADEH
JIADONG WANG
Lehigh University
MATTHEW GALATI
SAS Institute
MENAL GÜZELSOY
SAS Institute
SCOTT DE NEGRE
The Chartis Group

INFORMS Computing Society Conference, 7 January 2013

Thanks: Work supported in part by the National Science Foundation
Outline

1. Introduction
2. Basic Principles
   - Constraint Decomposition
   - Variable Decomposition
3. Basic Methods
   - Constraint Decomposition
   - Variable Decomposition
4. Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods
5. Decomposition in Practice
   - Software
   - Modeling
6. To Infinity and Beyond...
What is Decomposition?

- Many complex models are built up from simpler structures.
  - Subsystems linked by system-wide constraints or variables.
  - Complex combinatorial structures obtained by combining simpler ones.

- Decomposition is the process of taking a model and breaking it into smaller parts.

- The goal is either to
  - reformulate the model for easier solution;
  - reformulate the model to obtain an improved relaxation (bound); or
  - separate the model into stages or levels (possibly with separate objectives).
“Classical” decomposition arises from \textit{block structure} in the constraint matrix. By relaxing/fixing the linking variables/constraints, we then get a model that is separable. A separable model consists of multiple smaller submodels that are easier to solve. The separability lends itself nicely to \textit{parallel implementation}.

\[
\begin{pmatrix}
A_{01} & A_{02} & \cdots & A_{0\kappa} \\
A_1 & A_2 & & \\
& \ddots & & \\
& & A_{\kappa\kappa}
\end{pmatrix}
\begin{pmatrix}
A_{10} & A_{11} \\
A_{20} & A_{22} \\
& \ddots & \\
A_{\gamma 0}
\end{pmatrix}
\begin{pmatrix}
A_{00} & A_{01} & A_{02} & \cdots & A_{0\kappa} \\
A_{10} & A_{11} & & & \\
A_{20} & A_{22} & & & \\
& \ddots & & & \\
A_{\gamma 0} & & & & A_{\kappa\kappa}
\end{pmatrix}
\]
The Decomposition Principle

- Decomposition methods leverage our ability to solve either a relaxation or a restriction.
- Methodology is based on the ability to solve a given subproblem repeatedly with varying inputs.
- The goal of solving the subproblem repeatedly is to obtain information about its structure that can be incorporated into a master problem.
- An overarching theme in this tutorial will be that most solution methods for discrete optimization problems are, in a sense, based on the decomposition principle.

**Constraint decomposition**
- Relax a set of linking constraints to expose structure.
- Leverages ability to solve either the optimization or separation problem for a relaxation (with varying objectives and/or points to be separated).

**Variable decomposition**
- Fix the values of linking variables to expose the structure.
- Leverages ability to solve a restriction (with varying right-hand sides).
The problem is to assign $m$ tasks to $n$ machines subject to capacity constraints.

The variable $x_{ij}$ is one if task $i$ is assigned to machine $j$.

The “profit” associated with assigning task $i$ to machine $j$ is $c_{ij}$.

If we relax the requirement that each task be assigned to only one machine, the problem decomposes into $n$ independent knapsack problems.
Facility Location Problem

\[
\begin{align*}
\text{min} & \quad \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{s.t.} & \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \\
& \quad x_{ij} \leq y_j \quad \forall i, j \\
& \quad x_{ij}, y_j \in \{0, 1\} \quad \forall i, j
\end{align*}
\]

- We are given \( n \) facility locations and \( m \) customers to be serviced from those locations.
- There is a fixed cost \( c_j \) associated with facility \( j \).
- There is a cost \( d_{ij} \) associated with serving customer \( i \) from facility \( j \).
- We have two sets of binary variables.
  - \( y_j \) is 1 if facility \( j \) is opened, 0 otherwise.
  - \( x_{ij} \) is 1 if customer \( i \) is served by facility \( j \), 0 otherwise.
- If we fix the set of open facilities, then the problem becomes easy to solve.
Traveling Salesman Problem Formulation

\[ x(\delta(\{u\})) = 2 \quad \forall u \in V \]
\[ x(E(S)) \leq |S| - 1 \quad \forall S \subset V, \ 3 \leq |S| \leq |V| - 1 \]
\[ x_e \in \{0, 1\} \quad \forall e \in E \]
Example: Underlying Combinatorial Structure

Traveling Salesman Problem Formulation

\begin{align*}
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{align*}

Two relaxations

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

\begin{align*}
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{align*}
Example: Underlying Combinatorial Structure

Traveling Salesman Problem Formulation

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

Two relaxations

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

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

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

\[
\begin{align*}
  x(\delta(\{u\})) & = 2 & \forall u \in V \\
  x_e & \in \{0, 1\} & \forall e \in E
\end{align*}
\]
Example: Eliminating Symmetry

- In some cases, the identified blocks are identical.
- In such cases, the original formulation will often be highly symmetric.
- The decomposition eliminates the symmetry by collapsing the identical blocks.

Vehicle Routing Problem (VRP)

\[
\begin{align*}
\text{min} & \quad \sum_{k \in M} \sum_{(i,j) \in A} c_{ij} x_{ijk} \\
\sum_{k \in M} \sum_{i \in V} \sum_{j \in N} x_{ijk} &= 1 \quad \forall i \in V \\
\sum_{k \in M} \sum_{j \in N} d_{ij} x_{ijk} &\leq C \quad \forall k \in M \\
\sum_{j \in N} x_{0jk} &= 1 \quad \forall k \in M \\
\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} &= 0 \quad \forall h \in V, k \in M \\
\sum_{i \in N} x_{i,n+1,k} &= 1 \quad \forall k \in M \\
x_{ijk} &\in \{0, 1\} \quad \forall (i,j) \in A, k \in M
\end{align*}
\]
**Integer Linear Program**: Minimize/Maximize a linear *objective function* over a (discrete) set of *solutions* satisfying specified *linear constraints*.

\[
 z_{\text{IP}} = \min_{x \in \mathbb{Z}^n} \left\{ c^T x \mid Ax \geq b \right\}
\]
Implicit enumeration techniques try to enumerate the solution space in an intelligent way.

The most common algorithm of this type is branch and bound.

Suppose \( F \) is the set of feasible solutions for a given MILP. We wish to solve \( \min_{x \in F} c^\top x \).

**Divide and Conquer**

Consider a partition of \( F \) into subsets \( F_1, \ldots, F_k \). Then

\[
\min_{x \in F} c^\top x = \min_{1 \leq i \leq k} \{ \min_{x \in F_i} c^\top x \}.
\]

We can then solve the resulting subproblems recursively.

Dividing the original problem into subproblems is called branching.

Taken to the extreme, this scheme is equivalent to complete enumeration.

We avoid complete enumeration primarily by deriving bounds on the value of an optimal solution to each subproblem.
A relaxation of an ILP is an auxiliary mathematical program for which:
- the feasible region contains the feasible region for the original ILP, and
- the objective function value of each solution to the original ILP is not increased.
Relaxations can be used to efficiently get bounds on the value of the original integer program.

Types of Relaxations:
- Continuous relaxation
- Combinatorial relaxation
- Lagrangian relaxations

Branch and Bound

Initialize the queue with $F$. While there are subproblems in the queue, do:

1. Remove a subproblem and solve its relaxation.
2. The relaxation is infeasible $\Rightarrow$ subproblem is infeasible and can be pruned.
3. Solution is feasible for the MILP $\Rightarrow$ subproblem solved (update upper bound).
4. Solution is not feasible for the MILP $\Rightarrow$ lower bound.
   - If the lower bound exceeds the global upper bound, we can prune the node.
   - Otherwise, we branch and add the resulting subproblems to the queue.
Branching involves partitioning the feasible region by imposing a *valid disjunction* such that:

- All optimal solutions are in one of the members of the partition.
- The solution to the current relaxation is not in any of the members of the partition.
Branch and Bound Tree
Outline

1 Introduction

2 Basic Principles
   - Constraint Decomposition
   - Variable Decomposition

3 Basic Methods
   - Constraint Decomposition
   - Variable Decomposition

4 Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods

5 Decomposition in Practice
   - Software
   - Modeling

6 To Infinity and Beyond...
Basic Strategy:

- The original linear program is “hard” to solve because of its size or other properties.
- The matrix $A'$ has structure that makes optimization “easy.”
  - Block structure
  - Network structure

With decomposition, we can exploit the structure to obtain better solution methods.
**Basic Strategy:** Leverage our ability to solve the optimization/separation problem for a relaxation to improve the bound yielded by the LP relaxation.

\[
\begin{align*}
    z_{\text{IP}} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\} \\
    z_{\text{LP}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x \geq b', A''x \geq b'' \right\} \\
    z_{\text{D}} &= \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}
\end{align*}
\]

\[z_{\text{IP}} \geq z_{\text{D}} \geq z_{\text{LP}}\]

**Assumptions:**
- \(\text{OPT}(\mathcal{P}, c)\) and \(\text{SEP}(\mathcal{P}, x)\) are “hard”
- \(\text{OPT}(\mathcal{P}', c)\) and \(\text{SEP}(\mathcal{P}', x)\) are “easy”
- \(Q''\) can be represented explicitly (description has polynomial size)
- \(\mathcal{P}'\) may be represented implicitly (description has exponential size)
**Basic Strategy:** Leverage our ability to solve the optimization/separation problem for a relaxation to improve the bound yielded by the LP relaxation.

\[
\begin{align*}
z_{IP} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A' x \geq b', A'' x \geq b'' \right\} \\
z_{LP} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A' x \geq b', A'' x \geq b'' \right\} \\
z_D &= \min_{x \in P'} \left\{ c^\top x \mid A'' x \geq b'' \right\}
\end{align*}
\]

**Assumptions:**
- \( \text{OPT}(\mathcal{P}, c) \) and \( \text{SEP}(\mathcal{P}, x) \) are “hard”
- \( \text{OPT}(\mathcal{P}', c) \) and \( \text{SEP}(\mathcal{P}', x) \) are “easy”
- \( \mathcal{Q}'' \) can be represented explicitly (description has polynomial size)
- \( \mathcal{P}' \) may be represented implicitly (description has exponential size)
Basic Strategy: Leverage our ability to solve the optimization/separation problem for a relaxation to improve the bound yielded by the LP relaxation.

\[
\begin{align*}
z_{\text{IP}} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A' x \geq b', A'' x \geq b'' \right\} \\
z_{\text{LP}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A' x \geq b', A'' x \geq b'' \right\} \\
z_{\text{D}} &= \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A'' x \geq b'' \right\}
\end{align*}
\]

\[z_{\text{IP}} \geq z_{\text{D}} \geq z_{\text{LP}}\]

Assumptions:
- \( \text{OPT}(\mathcal{P}, c) \) and \( \text{SEP}(\mathcal{P}, x) \) are "hard"
- \( \text{OPT}(\mathcal{P}', c) \) and \( \text{SEP}(\mathcal{P}', x) \) are "easy"
- \( \mathcal{Q}'' \) can be represented explicitly (description has polynomial size)
- \( \mathcal{P}' \) may be represented implicitly (description has exponential size)
Basic Strategy: Leverage our ability to solve the optimization/separation problem for a relaxation to improve the bound yielded by the LP relaxation.

\[
\begin{align*}
z_{\text{IP}} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A'x &\geq b', A''x \geq b'' \right\} \\
z_{\text{LP}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A'x &\geq b', A''x \geq b'' \right\} \\
z_D &= \min_{x \in \mathcal{P}'} \left\{ c^\top x \mid A''x \geq b'' \right\}
\end{align*}
\]

\[z_{\text{IP}} \geq z_D \geq z_{\text{LP}}\]

Assumptions:
- \(\text{OPT}(\mathcal{P}, c)\) and \(\text{SEP}(\mathcal{P}, x)\) are “hard”
- \(\text{OPT}(\mathcal{P}', c)\) and \(\text{SEP}(\mathcal{P}', x)\) are “easy”
- \(\mathcal{Q}''\) can be represented explicitly (description has polynomial size)
- \(\mathcal{P}'\) may be represented implicitly (description has exponential size)
Outline

1. Introduction
2. Basic Principles
   - Constraint Decomposition
   - Variable Decomposition
3. Basic Methods
   - Constraint Decomposition
   - Variable Decomposition
4. Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods
5. Decomposition in Practice
   - Software
   - Modeling
6. To Infinity and Beyond...
### Variable Decomposition in Linear Programming

\[ z_{LP} = \min_{(x,y) \in \mathbb{R}^n} \{ c'x + c''y \mid A'x + A''y \geq b \} \]

\[ = \min_{x \in \mathbb{R}^n} \{ c'x + \phi(b - A'x) \}, \]

where

\[ \phi(d) = \min c''y \]

s.t. \( A''y \geq d \)

\[ y \in \mathbb{R}^{n''} \]

### Basic Strategy:

- The function \( \phi \) is the value function of a linear program.
- The value function is piecewise linear and convex, but has a description of exponential size.
- We iteratively generate a lower approximation by evaluating \( \phi \) for various values of it domain (Benders’ Decomposition).
- The method is effective when we have an efficient method of evaluating \( \phi \).
Variable Decomposition in Integer Programming

\[ z_{\text{IP}} = \min_{(x,y) \in \mathbb{Z}^n} \left\{ c'x + c''y \mid A'x + A''y \geq b \right\} \]

\[ = \min_{x \in \mathbb{R}^n} \left\{ c'x + \phi(b - A'x) \right\} , \]

where

\[ \phi(d) = \min \, c''y \]

s.t. \[ A''y \geq d \]

\[ y \in \mathbb{Z}^{n''} \]

**Basic Strategy:**

- Here, \( \phi \) is the value function of an *integer program*.
- In the general case, the function \( \phi \) is piecewise linear but not convex.
- We can still iteratively generate a lower approximation by evaluating \( \phi \).
Connections Between Constraint and Variable Decomposition

- Constraint and variable decompositions are related.
- Fixing all the variables in the linking constraints also yields a decomposition.
- In the facility location example, relaxing the constraints that require any assigned facility to be open yields a constraint decomposition.
- In the linear programming case, constraint decomposition is variables decomposition in the dual.
- In the discrete case, the situation is more complex and there is no simple relationship between constraint and variables decomposition.
- A technique known as Lagrangian Decomposition can be used to decompose linking variables using constraint decomposition.
  - We make a copy of each original variable in each block.
  - We impose a constraint that all copies must take the same value.
  - We relax the new constraint in a Lagrangian fashion.
**Basic Strategy:** Leverage our ability to solve the optimization/separation problem for a relaxation to improve the bound yielded by the LP relaxation.

\[
\begin{align*}
  z_{\text{IP}} &= \min_{x \in \mathbb{Z}^n} \left\{ c^\top x \mid A' x \geq b', A'' x \geq b'' \right\} \\
  &= \min_{x', x'' \in \mathbb{Z}^n} \left\{ c^\top x' \mid A' x' \geq b', A'' x'' \geq b'', x' = x'' \right\} \\
  z_{\text{LP}} &= \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid A' x \geq b', A'' x \geq b'' \right\} \\
  z_{\text{LD}} &= \min \left\{ c^\top x \mid x \in \mathcal{P}' \cap \mathcal{P}'' \right\} \\
  z_{\text{IP}} \geq z_{\text{LD}} \geq z_{\text{LP}}
\end{align*}
\]

**Assumptions:**
- \( \text{OPT}(\mathcal{P}, c) \) and \( \text{SEP}(\mathcal{P}, x) \) are “hard”
- \( \text{OPT}(\mathcal{P}', c) \) and \( \text{OPT}(\mathcal{P}'', x) \) are “easy”
Cutting Plane Method (CPM)

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

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

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

*Exponential number of constraints*

![Diagram showing the cutting plane method with a master problem and subproblems.](image)
**Cutting Plane Method (CPM)**

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

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

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

*Exponential number of constraints*

---

Ralphs, et. al. | Decomposition Methods for Discrete Optimization
**Cutting Plane Method (CPM)**

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

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

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

*Exponential number of constraints*

---

Ralphs, et. al. Decomposition Methods for Discrete Optimization
Cutting Plane Method (CPM)

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

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

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

*Exponential number of constraints*

---

Ralphs, et. al.  
**Decomposition Methods for Discrete Optimization**
Dantzig-Wolfe Method (DW)

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

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

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

*Exponential number of variables*
Dantzig-Wolfe Method (DW)

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

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

\[ \mathcal{P}' = \left\{ x \in \mathbb{R}^n \mid x = \sum_{s \in \mathcal{E}} s \lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in \mathcal{E} \right\} \]

*Exponential number of variables*
Dantzig-Wolfe Method (DW)

**DW** combines an *inner* approximation of \( P' \) with an explicit description of \( Q'' \)

- **Master:** \( z_{DW} = \min_{\lambda \in \mathbb{R}^E_+} \{ c^T (\sum_{s \in E} s \lambda_s) \mid A'' (\sum_{s \in E} s \lambda_s) \geq b'', \sum_{s \in E} \lambda_s = 1 \} \)

- **Subproblem:** \( \text{OPT} (P', c^T - u_{DW}^T A'') \)

\[
P' = \left\{ x \in \mathbb{R}^n \mid x = \sum_{s \in E} s \lambda_s, \sum_{s \in E} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in E \right\}
\]

*Exponential number of variables*

Ralphs, et. al.  
Decomposition Methods for Discrete Optimization
Dantzig-Wolfe Method (DW)

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

- **Master:** $z_{DW} = \min_{\lambda \in \mathbb{R}_+^E} \left\{ c^T \left( \sum_{s \in E} s \lambda_s \right) \mid A'' \left( \sum_{s \in E} s \lambda_s \right) \geq b'', \sum_{s \in E} \lambda_s = 1 \right\}$

- **Subproblem:** $\text{OPT} (\mathcal{P}', c^T - u^T_{DW} A'')$

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

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

$$
\begin{align*}
  z_{LD} &= \max_{\alpha \in \mathbb{R}, \ u \in \mathbb{R}^{m''}} \left\{ \alpha + b''^\top u \mid \left( c^\top - u^\top A'' \right) s - \alpha \geq 0 \ \forall s \in \mathcal{E} \right\} = z_{DW}
\end{align*}
$$
The Lagrangian Method (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**: $z_{LD} = \max_{u \in \mathbb{R}^{m''}} \left\{ \min_{s \in \mathcal{E}} \{ c^\top s + u^\top (b'' - A'' s) \} \right\}$

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

\[
z_{LD} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}^{m''}} \left\{ \alpha + b''^\top u \mid \left( c^\top - u^\top A'' \right) s - \alpha \geq 0 \ \forall s \in \mathcal{E} \right\} = z_{DW}
\]
**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:** $z_{LD} = \max_{u \in \mathbb{R}^{m''}_+} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A'' s) \right\} \right\}$
- **Subproblem:** $\text{OPT} (\mathcal{P}', c^\top - u_{LD}^\top A'')$

$$z_{LD} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}^{m''}_+} \left\{ \alpha + b''^\top u \mid \left( c^\top - u^\top A'' \right) s - \alpha \geq 0 \forall s \in \mathcal{E} \right\} = z_{DW}$$
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**: $z_{LD} = \max_{u \in \mathbb{R}^{m''}_+} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (b'' - A'' s) \right\} \right\}$
- **Subproblem**: $\text{OPT} (\mathcal{P}', c^\top - u_{LD}^\top A'')$

$$z_{LD} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}^{m''}_+} \left\{ \alpha + b''^\top u \left| \left( c^\top - u^\top A'' \right) s - \alpha \geq 0 \ \forall s \in \mathcal{E} \right\} = z_{DW}$$
The **LP bound** is obtained by optimizing over the intersection of two explicitly defined polyhedra.

\[
z_{\text{LP}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid x \in Q' \cap Q'' \right\}
\]

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

\[
z_{\text{CP}} = z_{\text{DW}} = z_{\text{LD}} = z_{\text{D}} = \min_{x \in \mathbb{R}^n} \left\{ c^\top x \mid x \in P' \cap Q'' \right\} \geq z_{\text{LP}}
\]

Traditional constraint decomposition-based bounding methods contain two primary steps

- **Master Problem:** Update the primal/dual solution information
- **Subproblem:** Update the approximation of \( P' \): \( \text{SEP}(P', x) \) or \( \text{OPT}(P', c) \)
When to Apply Constraint Decomposition

Typical scenarios in which constraint decomposition is effective.

- The problem has **block structure** that makes solution of the subproblem very efficient and/or parallelizable.

- The subproblem has a substantial integrality gap, but we know an efficient algorithm for solving it.

- The original problem is highly **symmetric** (has identical blocks) and the decomposition eliminates the symmetry.

Choosing a particular algorithm raises additional issues.

- Cutting plane methods are hard to beat if strong cuts are known for the subproblem.

- Cutting plane methods also allow a wider variety of cuts to be generated (cuts from the tableau or from multiple relaxations).

- Among traditional decomposition methods, Dantzig-Wolfe is appropriate if cuts for the master are to be generated or when branching in the original space.

- Lagrangian methods offer fast solve times in the master and less overhead, but only approximate primal solution information.
Variable Decomposition in Linear Programming

\[ z_{LP} = \min \quad x + y \]
\[ \text{s.t.} \quad 25x - 20y \geq -30 \]
\[ -x - 2y \geq -10 \]
\[ -2x + y \geq -15 \]
\[ 2x + 10y \geq 15 \]
\[ x, y \in \mathbb{R} \]
Value Function Reformulation

\[ z_{LP} = \min_{x \in \mathbb{R}} x + \phi(x), \]

where

\[ \phi(x) = \min y \]
\[ \text{s.t. } -20y \geq -30 - 25x \]
\[ -2y \geq -10 + x \]
\[ y \geq -15 + 2x \]
\[ 10y \geq 15 - 2x \]
\[ y \in \mathbb{R} \]
Example

\[ \phi(d) = \min \ 6x_1 + 7x_2 + 5x_3 \]
\[ \text{s.t.} \ 2x_1 - 7x_2 + x_3 = d \]
\[ x_1, x_2, x_3 \in \mathbb{R}_+ \]
LP Value Function Structure

LP Value Function

\[ \phi(d) = \min c^\top x \]
\[ \text{s.t. } Ax = b \]
\[ x \in \mathbb{R}^n_+ \]

(LP)

- Suppose the dual of (LP) is feasible and bounded.
- Then the epigraph of \( \phi \) is the convex cone

\[ \{ (b, z) \mid z \geq \nu^\top b, \forall \nu \in \mathcal{E} \} \]

where \( \mathcal{E} \) is the set of extreme points of the dual of (LP).
- Thus, the value function is piecewise linear and convex with each piece corresponding to an extreme point of the dual.
Benders’ Method for Linear Programs

\[ z_{\text{LP}} = \min_{(x,y) \in \mathbb{R}^n} \{ c'x + c''y \mid A'x + A''y \geq b \} \]

\[ = \min_{x \in \mathbb{R}^n} \{ c'x + \phi(b - A'x) \} \]

\[ = \min_{x \in \mathbb{R}^n} \{ c'x + z \mid z \geq \nu(b - A'x), \nu \in \mathcal{E} \} \]

**Basic Strategy:**

- Solve the above linear program with a cutting plane method.
- We iteratively generate a lower approximation by evaluating \( \phi \) for various values of \( x \) (Benders’ Decomposition).
Variable Decomposition in Integer Programming

\[
\begin{align*}
z_{IP} &= \min \quad x + y \\
\text{s.t.} \quad &25x - 20y \geq -30 \\
&-x - 2y \geq -10 \\
&-2x + y \geq -15 \\
&2x + 10y \geq 15 \\
x, y \in \mathbb{Z}
\end{align*}
\]
\[ \begin{align*}
    z_{IP} &= \min_{x \in \mathbb{Z}} x + \phi(x), \\
    \phi(x) &= \min y \\
    \text{s.t.} & \ -20y \geq -30 - 25x \\
    & \ -2y \geq -10 + x \\
    & \ y \geq -15 + 2x \\
    & \ 10y \geq 15 - 2x \\
    & \ y \in \mathbb{Z}
\end{align*} \]
Example

\[ \phi(d) = \min \ 3x_1 + \frac{7}{2}x_2 + 3x_3 + 6x_4 + 7x_5 + 5x_6 \]

s.t. \[ 6x_1 + 5x_2 - 4x_3 + 2x_4 - 7x_5 + x_6 = d \]

\[ x_1, x_2, x_3 \in \mathbb{Z}^+, \ x_4, x_5, x_6 \in \mathbb{R}^+ \]
The epigraph of the MILP value function is the union of a countable collection of epigraphs of identical convex cones.

These convex cones are translations of the value function of the continuous restriction.
Benders’ Method for Integer Programs

\[ z_{LP} = \min_{(x,y) \in \mathbb{R}^n} \{ c'x + c''y \mid A'x + A''y \geq b \} \]
\[ = \min_{x \in \mathbb{R}^n} \{ c'x + \phi(b - A'x) \} \]
\[ \geq \min_{x \in \mathbb{R}^n} \{ c'x + z \mid z \geq \phi_D(b - A'x) \} \]

where \( \phi_D \) is a function bounding \( \phi \) from below.

**Basic Strategy:**
- Solve the above nonlinear program by iteratively constructing \( \phi_D \).
- The approximation can be updated each time we solve the MILP.
- The pieces of the approximation come from the branch-and-bound tree resulting from solution of the MILP for fixed \( d \).
Approximating the Value Function
Just as in the case of constraint decomposition, variable decomposition methods contain two primary steps:

- **Master Problem**: Update the primal/dual **solution** information
- **Subproblem**: Update the **approximation** of $\phi$ by evaluating $\phi(x)$.

The motivation for applying variable decomposition methods is a bit different than for constraint decomposition methods.

Generally, variable decomposition is appropriate when:
- we have an efficient method for evaluating $\phi$ (it has block structure) or
- we have a multilevel problem with multiple objectives.

In cases like stochastic programming, the blocks may only differ in their right-hand side, so there is only one (lower-dimensional) function needed to describe all blocks.

It may also be possible to exploit symmetry in variable decomposition using a strategy similar to that used in constraint decomposition.
Outline

1. Introduction

2. Basic Principles
   - Constraint Decomposition
   - Variable Decomposition

3. Basic Methods
   - Constraint Decomposition
   - Variable Decomposition

4. Advanced Methods
   - Hybrid Methods
     - Decomposition and Separation
     - Decomposition Cuts
     - Generic Methods

5. Decomposition in Practice
   - Software
   - Modeling

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

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

---

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

As in CPM, separate $\hat{x}_{PC} = \sum_{s \in \mathcal{E}} S \hat{\lambda}_s$ 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**: \( z_{PC} = \min_{\lambda \in \mathbb{R}^E_+} \left\{ c^T \left( \sum_{s \in E} s \lambda_s \right) \mid D \left( \sum_{s \in E} s \lambda_s \right) \geq d, \sum_{s \in E} \lambda_s = 1 \right\} \)
- **Subproblem**: \( \text{OPT} (\mathcal{P}', c^T - u^T_{PC} D) \) or \( \text{SEP} (\mathcal{P}, x_{PC}) \)

As in CPM, separate \( \hat{x}_{PC} = \sum_{s \in E} s \hat{\lambda}_s \) 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**: $z_{PC} = \min_{\lambda \in \mathbb{R}_+^E} \left\{ c^T \left( \sum_{s \in E} s \lambda_s \right) \mid D \left( \sum_{s \in E} s \lambda_s \right) \geq d, \sum_{s \in E} \lambda_s = 1 \right\}$
- **Subproblem**: $\text{OPT} (\mathcal{P}', c^T - u^T_{PC}D)$ or $\text{SEP} (\mathcal{P}, x_{PC})$

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

\[
\mathcal{P}^1_O = \text{conv}(\mathcal{E}_2) \cap \mathcal{P}'
\]
\[
\mathcal{P}^2_O = \mathcal{P}^1_O \cap \{ x \in \mathbb{R}^n \mid x_2 \geq 2 \}
\]
\[
x_{PC}^2 = (3, 2)
\]
\[
\{ s \in \mathcal{E} \mid (\lambda^2_{PC})_s > 0 \}
\]
**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:** $z_{PC} = \min_{\lambda \in \mathbb{R}_+^E} \{ c^\top (\sum_{s \in E} s \lambda_s) \mid D (\sum_{s \in E} s \lambda_s) \geq d, \sum_{s \in E} \lambda_s = 1 \}$
- **Subproblem:** $\text{OPT} (\mathcal{P}', c^\top - u_{PC} D)$ or $\text{SEP} (\mathcal{P}, x_{PC})$

As in CPM, separate $\hat{x}_{PC} = \sum_{s \in E} s \hat{\lambda}_s$ 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:** $z_{LD} = \max_{u \in \mathbb{R}^m_{+}} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **Subproblem:** $\text{OPT} \left( \mathcal{P}', c^\top - u_{LD}^\top D \right)$ 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 $\hat{x} \in \mathbb{R}^n$. 

![Diagram](image-url)
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:** $z_{LD} = \max_{u \in \mathbb{R}^m_+} \left\{ \min_{s \in \mathcal{E}} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$

- **Subproblem:** $\text{OPT} (\mathcal{P}', c^\top - u_{LD}^\top D)$ or $\text{SEP} (\mathcal{P}, s)$

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

\[
\tilde{s} = (3, 4)
\]
**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:** $z_{LD} = \max_{u \in \mathbb{R}^{m''}} \left\{ \min_{s \in E} \left\{ c^\top s + u^\top (d - Ds) \right\} \right\}$
- **Subproblem:** $\text{OPT} (\mathcal{P}', c^\top - u^\top_{LD} D)$ or $\text{SEP} (\mathcal{P}, s)$

In each iteration, separate $\hat{s} \in E$, a solution to the Lagrangian relaxation.

- **Advantage:** Often easier to separate $s \in E$ from $\mathcal{P}$ than $\hat{x} \in \mathbb{R}^n$. 

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

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

\[ c^\top \]

\[ c^\top - \hat{u}^\top D \]

\[ \tilde{s} = (2, 1) \]

\[ \tilde{s} = (2, 1) \]

\[ \tilde{s} = (3, 4) \]

\[ \tilde{s} = (2, 1) \]
Outline

1 Introduction

2 Basic Principles
   - Constraint Decomposition
   - Variable Decomposition

3 Basic Methods
   - Constraint Decomposition
   - Variable Decomposition

4 Advanced Methods
   - Hybrid Methods
     - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods

5 Decomposition in Practice
   - Software
   - Modeling

6 To Infinity and Beyond...
In general, $\text{OPT}(X, c)$ and $\text{SEP}(X, x)$ are polynomially equivalent.

**Observation:** Restrictions on input or output can change their complexity.

**The Template Paradigm**, restricts the output of $\text{SEP}(X, x)$ to valid inequalities that conform to a certain structure. This class of inequalities forms a polyhedron $C \supset X$ (the *closure*).

For example, let $P$ be the convex hull of solutions to the TSP.

- $\text{SEP}(P, x)$ is $\mathcal{NP}$-Complete.
- $\text{SEP}(C, x)$ is polynomially solvable, for $C \supset P$
  - $P_{\text{Subtour}}$, the Subtour Polytope (separation using Min-Cut), or
  - $P_{\text{Blossom}}$, the Blossom Polytope (separation using Letchford, et al.).

**Structured Separation**, restricts the input of $\text{SEP}(X, x)$, 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 Comb Inequalities:**
  \[
  x(E(H)) + \sum_{i=1}^{k} x(E(T_i)) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \lceil k/2 \rceil
  \]

- **SEP(\mathcal{P}^{\text{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 (\( \geq 3 \)) 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): 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 \( \hat{x}_{PC} \), separate each member of \( D \).

- As with **RC**, often **much easier** to separate \( s \in E \) than \( \hat{x}_{PC} \in \mathbb{R}^n \).

- **RC** only gives us **one** member of \( E \) to separate, while **PC** gives us a set, one of which must be violated by any inequality violated by \( \hat{x}_{PC} \).

- 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.

- Similarly, the violated blossom found by separating the 1-tree *also* violates the fractional point, but was found at little cost.
Price-and-Cut (Revisited)

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

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

1 Introduction

2 Basic Principles
   - Constraint Decomposition
   - Variable Decomposition

3 Basic Methods
   - Constraint Decomposition
   - Variable Decomposition

4 Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
     - Decomposition Cuts
   - Generic Methods

5 Decomposition in Practice
   - Software
   - Modeling

6 To Infinity and Beyond…
Decompose-and-Cut (DC)

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

$$\begin{align*}
\min_{\lambda \in \mathbb{R}_+^E, (x^+, x^-) \in \mathbb{R}_+^n} & \quad x^+ + x^- \\
\text{s.t.} & \quad \sum_{s \in E} s \lambda_s + x^+ - x^- = \hat{x}_{CP}, \quad \sum_{s \in E} \lambda_s = 1
\end{align*}$$
Decompose-and-Cut (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.’s of \( P' \)

\[
\min_{\lambda \in \mathbb{R}_+^E, (x^+, x^-) \in \mathbb{R}^+_n} \left\{ x^+ + x^- \mid \sum_{s \in \mathcal{E}} s \lambda_s + x^+ - x^- = \hat{x}_{CP}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\}
\]

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

\textit{Decomposition Cuts}

\[
u^t_{DC}s + \alpha^t_{DC} \leq 0 \quad \forall s \in P' \quad \text{and} \quad u^t_{DC}\hat{x}_{CP} + \alpha^t_{DC} > 0
\]
Decompose-and-Cut (DC)

Decompose-and-Cut: Each iteration of CPM, decompose into convex combo of e.p.’s of \( P' \).

\[
\min_{\lambda \in \mathbb{R}^E_+, (x^+, x^-) \in \mathbb{R}^n_+} \left\{ x^+ + x^- \left| \sum_{s \in E} s \lambda_s + x^+ - x^- = \hat{x}_{\text{CP}}, \sum_{s \in E} \lambda_s = 1 \right. \right\}
\]

- Originally proposed as a method to solve the VRP with TSP as relaxation.
- Essentially, we are transforming an optimization algorithm into a separation algorithm.
- 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
Outline

1 Introduction

2 Basic Principles
   - Constraint Decomposition
   - Variable Decomposition

3 Basic Methods
   - Constraint Decomposition
   - Variable Decomposition

4 Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods

5 Decomposition in Practice
   - Software
   - Modeling

6 To Infinity and Beyond…

Ralphs, et. al. Decomposition Methods for Discrete Optimization
Traditionally, decomposition-based branch-and-bound methods have required extensive problem-specific customization.

- identifying the decomposition (which constraints to relax);
- formulating and solving the subproblem (either optimization or separation over $P'$);
- formulating and solving the master problem; and
- performing the branching operation.

However, it is possible to replace these components with generic alternatives.

- The decomposition can be identified automatically by analyzing the matrix or through a modeling language.
- The subproblem can be solved with a generic MILP solver.
- The branching can be done in the original compact space.

The remainder of this talk focuses on our recent efforts to develop a completely generic decomposition-based MILP solver.
Working in the Compact Space

- The key to the implementation of this unified framework is that we always maintain a representation of the problem in the compact space.

- This allows us to employ most of the usual techniques used in LP-based branch and bound without modification, even in this more general setting.

- There are some challenges related to this approach that we are still working on.
  - Gomory cuts
  - Preprocessing
  - Identical subproblems
  - Strong branching

- Allowing the user to express all methods in the compact space is extremely powerful when it comes to modeling language support.

- It is important to note that DIP currently assumes the existence of a formulation in the compact space.

- We are working on relaxing this assumption, but this means the loss of the fully generic implementation of some techniques.
For unstructured problems, block structure may be detected automatically.

This is done using hypergraph partitioning methods.

We map each row of the original matrix to a hyperedge and the nonzero elements to nodes in a hypergraph.

Hypergraph partitioning results in identification of the blocks in a singly-bordered block diagonal matrix.
Hidden Block Structure

MIPLIB2003 instance: p2756

nz = 8937
Hidden Block Structure
By default, we branch on variables in the compact space.

In PC, this is done by mapping 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 makes it possible define generic branching procedures.
Branching for Lagrangian Method

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

\[ P_I = \left\{ x \in \mathbb{R}^n \mid x = \sum_{s \in B} s \lambda_s, \sum_{s \in B} \lambda_s = 1, \lambda_s \geq 0 \quad \forall s \in B \right\} \]

\[ \min_{\lambda \in \mathbb{R}_+^B} \left\{ c^\top \left( \sum_{s \in B} s \lambda_s \right) \mid A'' \left( \sum_{s \in B} s \lambda_s \right) \geq b'', \sum_{s \in B} \lambda_s = 1 \right\} \]

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

1 Introduction

2 Basic Principles
   - Constraint Decomposition
   - Variable Decomposition

3 Basic Methods
   - Constraint Decomposition
   - Variable Decomposition

4 Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods

5 Decomposition in Practice
   - Software
   - Modeling

6 To Infinity and Beyond...
There have been a number of efforts to create frameworks supporting the implementation of decomposition-based branch and bound.

**Column Generation Frameworks**
- ABACUS [Jünger and Thienel(2012)]
- COIN/BCP [Ladányi(2012)]

**Generic decomposition frameworks**
- BaPCod [Vanderbeck(2012)]
  - Dantzig-Wolfe
  - Automatic reformulation,
  - Generic cuts
  - Generic branching
- GCG [Gamrath and Lübbecke(2012)]
  - Dantzig-Wolfe
  - Automatic hypergraph-based decomposition
  - Automatic reformulation,
  - Generic cut generation
  - Generic branching
The use of decomposition methods 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.

**DIP** and **CHiPPS** are two frameworks that together allow for easier implementation of decomposition approaches.

- **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.

**DIP** can be accessed through a modeling language or by providing a model with notated structure.
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
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 $\text{OPT}(P', c)$ [optional, if $Q'$, CBC is built-in]
  - `generateCuts()`: define a method for $\text{SEP}(P', x)$ [optional, CGL is built-in]
  - `isUserFeasible()`: is $\hat{x} \in P$? [optional, if $P = \text{conv}(P' \cap Q'' \cap \mathbb{Z})$]

All methods have appropriate defaults but are **virtual** and may be overridden.

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.
DIP Framework: Feature Overview

- **One interface** to all algorithms: CP, DW, LD, PC, RC
- **Automatic reformulation** allows users to specify methods in the compact (original) space.
- **Integrate different decomposition methods**
  - Can utilize CGL cuts in all algorithms (separate from original space).
  - Can utilize *structured separation* (efficient algorithms that apply only to vectors with special structure (integer) in various ways).
  - Can separate from $P'$ using subproblem solver (DC).
- **Integrate multiple bounding methods**
  - Column generation based on *multiple/nested relaxations* can be easily defined and employed.
  - Bounds based on *multiple model/algorithm* combinations.
- **Use of generic MILP solution technology**
  - Using the mapping $\hat{x} = \sum_{s \in E} s\hat{\lambda}_s$ we can import any generic MILP technique to the PC/RC context.
  - Use generic MILP solver to solve subproblems.
  - Hooks to define branching methods, heuristics, etc.
Performance enhancements
- Detection and removal of columns that are close to parallel
- Basic dual stabilization (Wentges smoothing)
- Redesign (and simplification) of treatment of master-only variables.
- Branching can be 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

Algorithms for generating initial columns
- Solve $\text{OPT}(\mathcal{P}', c + r)$ for random perturbations
- Solve $\text{OPT}(\mathcal{P}_N)$ heuristically
- Run several iterations of LD or DC collecting extreme points

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
Many difficult MILPs have a block structure, but this structure is not part of the input (MPS) or is not exploitable by the solver.

In practice, it is common to have models composed of independent subsystems coupled by global constraints.

The result may be models that are highly symmetric and difficult to solve using traditional methods, but would be easy to solve if the structure were known.

\[
\begin{pmatrix}
A_1'' & A_2'' & \cdots & A_k'' \\
A_1' & & & \\
& A_2' & & \\
& & \ddots & \\
& & & A_k'
\end{pmatrix}
\]

MILPBlock provides a black-box solver for applying integrated methods to generic MILP.

Input is an MPS/LP and a block file specifying structure.

Optionally, the block file can be automatically generated using the hypergraph partitioning algorithm of HMetis.

This is the engine underlying DIPPY.
Outline

1. Introduction
2. Basic Principles
   - Constraint Decomposition
   - Variable Decomposition
3. Basic Methods
   - Constraint Decomposition
   - Variable Decomposition
4. Advanced Methods
   - Hybrid Methods
   - Decomposition and Separation
   - Decomposition Cuts
   - Generic Methods
5. Decomposition in Practice
   - Software
   - Modeling
6. To Infinity and Beyond...
In general, there are not many options for expressing block structure directly in a modeling language.

Part of the reason for this is that there are also not many software frameworks that can exploit this structure.

One substantial exception is GAMS, which offers the Extended Mathematical Programming (EMP) Language.

With EMP, it is possible to directly express multi-level and multi-stage problems in the modeling language.

For other modeling languages, it is possible to manually implement decomposition methods using traditional underlying solvers.

Here, we present a modeling language interface to DIP that provides the ability to express block structure and exploit it within DIP.
**DipPy**

- **DipPy** provides an interface to DIP through the modeling language **PuLP**.
- PuLP is a modeling language that provides functionality similar to other modeling languages.
- It is built on top of Python so you get the full power of that language for free.
- PuLP and DipPy are being developed by Stuart Mitchell and Mike O’Sullivan in Auckland and are part of COIN.
- Through DipPy, a user can
  - Specify the model and the relaxation, including the block structure.
  - Implement methods (coded in Python) for solving the relaxation, generating cuts, custom branching.
- With DipPy, it is possible to code a customized column-generation method from scratch in a few hours.
- This would have taken months with previously available tools.
Example: Generalized Assignment Problem

- The problem is to find a minimum cost assignment of $n$ tasks to $m$ machines such that each task is assigned to one machine subject to capacity restrictions.

- A binary variable $x_{ij}$ indicates that machine $i$ is assigned to task $j$. $M = 1, \ldots, m$ and $N = 1, \ldots, n$.

- The cost of assigning machine $i$ to task $j$ is $c_{ij}$

Generalized Assignment Problem (GAP)

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij}$$

$$\sum_{j \in N} w_{ij} x_{ij} \leq b_i \quad \forall i \in M$$

$$\sum_{i \in M} x_{ij} = 1 \quad \forall j \in N$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in M \times N$$
Creating GAP model in DipPy

```python
prob = dippy.DipProblem("GAP", LpMinimize)

# objective
prob += lpSum(assignVars[m][t] * COSTS[m][t] for m, t in MACHINES_TASKS), "min"

# machine capacity (knapsacks, relaxation)
for m in MACHINES:
    prob.relaxation[m] +=
    lpSum(assignVars[m][t] * RESOURCE_USE[m][t] for t in TASKS) <= CAPACITIES[m]

# assignment
for t in TASKS:
    prob += lpSum(assignVars[m][t] for m in MACHINES) == 1

prob.relaxed_solver = relaxed_solver

dippy.Solve(prob)
```
def relaxed_solver(prob, machine, redCosts, convexDual):
    # get tasks which have negative reduced
    task_idx = [t for t in TASKS if redCosts[assignVars[machine][t]] < 0]
    vars = [assignVars[machine][t] for t in task_idx]
    obj = [-redCosts[assignVars[machine][t]] for t in task_idx]
    weights = [RESOURCE_USE[machine][t] for t in task_idx]

    z, solution = knapsack01(obj, weights, CAPACITIES[machine])
    z = -z

    # get sum of original costs of variables in solution
    orig_cost = sum(prob.objective.get(vars[idx]) for idx in solution)
    var_values = [(vars[idx], 1) for idx in solution]

    dv = dippy.DecompVar(var_values, z-convexDual, orig_cost)

    # return, list of DecompVar objects
    return [dv]
GAP in DipPy

DipPy Auxiliary Methods

def solve_subproblem(prob, index, redCosts, convexDual):
    ...
    z, solution = knapsack01(obj, weights, CAPACITY)
    ...
    return []
prob.relaxed_solver = solve_subproblem

def knapsack01(obj, weights, capacity):
    ...
    return c[n-1][capacity], solution

def first_fit(prob):
    ...
    return bvs

def one_each(prob):
    ...
    return bvs
prob.init_vars = first_fit

def choose_antisymmetry_branch(prob, sol):
    ...
    return ([], down_branch_ub, up_branch_lb, [])
prob.branch_method = choose_antisymmetry_branch

def generate_weight_cuts(prob, sol):
    ...
    return new_cuts
prob.generate_cuts = generate_weight_cuts

def heuristics(prob, xhat, cost):
    ...
    return sols
prob.heuristics = heuristics
dippy.Solve(prob, {
    'doPriceCut': '1',
})
To Infinity and Beyond...

- **Separable subproblems** *(Important!)*
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
  - Cuts and branching?

- **Use of generic MILP solution technology**
  - Incorporation of advanced branching techniques (how to do strong branching)
  - Gomory cuts (crossover?)
  - Use generic MILP solver to generate multiple columns in each iteration.

- **Primal Heuristics**
  - For block-angular case, at end of each node, a simple heuristic is to solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
  - A number of other heuristics have been proposed.

- **Dual stabilization**

- **Presolve**
To Infinity and Beyond...

- **Separable subproblems** *(Important!)*
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
  - Cuts and branching?

- **Use of generic MILP solution technology**
  - Incorporation of advanced branching techniques (how to do strong branching)
  - Gomory cuts (crossover?)
  - Use generic MILP solver to generate multiple columns in each iteration.

- **Primal Heuristics**
  - For block-angular case, at end of each node, a simple heuristic is to solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
  - A number of other heuristics have been proposed.

- **Dual stabilization**

- **Presolve**
To Infinity and Beyond...

- **Separable subproblems** *(Important!)*
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
  - Cuts and branching?

- **Use of generic MILP solution technology**
  - Incorporation of advanced branching techniques (how to do strong branching)
  - Gomory cuts (crossover?)
  - Use generic MILP solver to generate multiple columns in each iteration.

- **Primal Heuristics**
  - For block-angular case, at end of each node, a simple heuristic is to solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen (’98), we extend to tree
  - A number of other heuristics have been proposed.

- Dual stabilization
- Presolve
To Infinity and Beyond...

- **Separable subproblems** *(Important!)*
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
  - Cuts and branching?

- **Use of generic MILP solution technology**
  - Incorporation of advanced branching techniques (how to do strong branching)
  - Gomory cuts (crossover?)
  - Use generic MILP solver to generate multiple columns in each iteration.

- **Primal Heuristics**
  - For block-angular case, at end of each node, a simple heuristic is to solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen ('98), we extend to tree
  - A number of other heuristics have been proposed.

- **Dual stabilization**
  - Presolve
To Infinity and Beyond...

- **Separable subproblems** *(Important!)*
  - Identical subproblems (symmetry)
  - Parallel solution of subproblems
  - Automatic detection
  - Cuts and branching?

- **Use of generic MILP solution technology**
  - Incorporation of advanced branching techniques (how to do strong branching)
  - Gomory cuts (crossover?)
  - Use generic MILP solver to generate multiple columns in each iteration.

- **Primal Heuristics**
  - For block-angular case, at end of each node, a simple heuristic is to solve with $\lambda \in \mathbb{Z}$
  - Used in *root node* by Barahona and Jensen (’98), we extend to tree
  - A number of other heuristics have been proposed.

- **Dual stabilization**

- **Presolve**
**Choice of master LP solver**

- Better automated choice of solver
- Interior-point methods might help with stabilization vs extremal duals
- Can we use volume or bundle along with an exact LP solver?

**Better search strategies**

- How do we warm start node processing?
- How much diving do we do?

**Nested pricing and solution methods**

- Can solve more constrained versions of subproblem heuristically to get high quality columns.
- Can we use decomposition recursively?

**Branch-and-Relax-and-Cut:** Not much done yet

**Numerics?**
**Choice of master LP solver**
- Better automated choice of solver
- Interior-point methods might help with stabilization vs extremal duals
- Can we use volume or bundle along with an exact LP solver?

**Better search strategies**
- How do we warm start node processing?
- How much diving do we do?

**Nested pricing and solution methods**
- Can solve more constrained versions of subproblem heuristically to get high quality columns.
- Can we use decomposition recursively?

**Branch-and-Relax-and-Cut**: Not much done yet

**Numerics?**
Choice of master LP solver
- Better automated choice of solver
- Interior-point methods might help with stabilization vs extremal duals
- Can we use volume or bundle along with an exact LP solver?

Better search strategies
- How do we warm start node processing?
- How much diving do we do?

Nested pricing and solution methods
- Can solve more constrained versions of subproblem heuristically to get high quality columns.
- Can we use decomposition recursively?

Branch-and-Relax-and-Cut: Not much done yet

Numerics?
To Infinity and Beyond...

- **Choice of master LP solver**
  - Better automated choice of solver
  - Interior-point methods might help with stabilization vs extremal duals
  - Can we use volume or bundle along with an exact LP solver?

- **Better search strategies**
  - How do we warm start node processing?
  - How much diving do we do?

- **Nested pricing and solution methods**
  - Can solve more constrained versions of subproblem heuristically to get high quality columns.
  - Can we use decomposition recursively?

- **Branch-and-Relax-and-Cut**: Not much done yet
To Infinity and Beyond...

- **Choice of master LP solver**
  - Better automated choice of solver
  - Interior-point methods might help with stabilization vs extremal duals
  - Can we use volume or bundle along with an exact LP solver?

- **Better search strategies**
  - How do we warm start node processing?
  - How much diving do we do?

- **Nested pricing and solution methods**
  - Can solve more constrained versions of subproblem heuristically to get high quality columns.
  - Can we use decomposition recursively?

- **Branch-and-Relax-and-Cut**: Not much done yet

- **Numerics?**
Decomposition methods are important in practice (see Mike Trick!), but have proven difficult to utilize in practice.

There is renewed interest in making these methods accessible to general users.
- Computational frameworks are being developed that employ these methods “generically.”
- Modeling language support is emerging that allows users to express structure that can be exploited.

All of this capability is still early in the development stages.

There will need to be an evolution similar to what happened when generic MILP solvers generalized problem-specific techniques.

There are LOADS of questions to be answered and research to be done.

THANKS FOR LISTENING!
Gamrath, G. and M. Lübbecke 2012.
GCG.

SYMPHONY.

Ladányi, L. 2012.
BCP.

SYMPHONY.

Vanderbeck, F. 2012.
BapCod: A generic branch-and-price code.