DECOMP: A Framework for Decomposition in Integer Programming

Matthew Galati¹  Ted Ralphs²

¹SAS Institute – Analytical Solutions – Operations Research and Development
²Lehigh University – Department of Industrial and Systems Engineering

INFORMS 2006
Computational Optimization and Software
1. Decomposition Methods
2. Integrated Decomposition Methods
3. DECOMP Framework
Outline

1. Decomposition Methods
2. Integrated Decomposition Methods
3. DECOMP Framework
Consider the following integer linear program (ILP):

\[
  z_{IP} = \min_{x \in F} \{ c^\top x \} = \min_{x \in P} \{ c^\top x \} = \min_{x \in \mathbb{Z}^n} \{ c^\top x \mid Ax \geq b \}
\]

where

\[
  F = \{ x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b'' \}
\]

\[
  F' = \{ x \in \mathbb{Z}^n \mid A'x \geq b' \}
\]

\[
  Q = \{ x \in \mathbb{R}^n \mid A'x \geq b', A''x \geq b'' \}
\]

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

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

- Denote \( \mathcal{P} = \text{conv}(F) \) and \( \mathcal{P}' = \text{conv}(F') \).
- \( \text{OPT}(c, X) \): Subroutine returns \( x \in X \) that minimizes \( c^\top x \).
- \( \text{SEP}(x, X) \): Subroutine returns \((a, \beta)\) which separates \( x \) from \( X \) (if exists).
**Assumption:**
- $OPT(c, \mathcal{P})$ and $SEP(x, \mathcal{P})$ are “hard”.
- $OPT(c, \mathcal{P}')$ and $SEP(x, \mathcal{P}')$ are “easy”.
- $Q''$ can be represented explicitly (description has polynomial size).
- $\mathcal{P}'$ must be represented implicitly (description has exponential size).

**Example - Traveling Salesman Problem**

- $x(\delta\{u\}) = 2 \quad \forall u \in V$
- $x(E(S)) \leq |S| - 1 \quad \forall S \subseteq V, 3 \leq |S| \leq |V| - 1$
- $x_e \in \{0, 1\} \quad \forall e \in E$

One classical decomposition of TSP is to look for a spanning subgraph with $|V|$ edges ($\mathcal{P}' = 1$-Tree) that satisfies the 2-degree constraints ($Q''$).
Preliminaries

**Assumption:**

- $OPT(c, P)$ and $SEP(x, P)$ are “hard”.
- $OPT(c, P')$ and $SEP(x, P')$ are “easy”.
- $Q''$ can be represented explicitly (description has polynomial size).
- $P'$ must be represented implicitly (description has exponential size).

**Example - Traveling Salesman Problem**

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

One classical decomposition of TSP is to look for a spanning subgraph with $|V|$ edges ($P' = 1$-Tree) that satisfies the 2-degree constraints ($Q''$).
Bounding

Goal

Compute a lower bound on \( z_{LP} \) by building an approximation to \( \mathcal{P} \).

- The most common approach is to use the LP relaxation.

\[
    z_{LP} = \min_{x \in \mathcal{Q}} \{ c^T x \} = \min_{x \in \mathbb{R}^n} \{ c^T x \mid A'x \geq b', A''x \geq b'' \}
\]

- Decomposition methods attempt to improve on this bound by utilizing the fact that \( OPT(c, \mathcal{P}') \) or \( SEP(x, \mathcal{P}') \) is easy.

\[
    z_D = \min_{x \in \mathcal{P}'} \{ c^T x \mid A''x \geq b'' \} = \min_{x \in \mathcal{P}' \cap \mathcal{Q}''} \{ c^T x \} \geq z_{LP}
\]

- \( \mathcal{P}' \) is represented implicitly through solution of a subproblem.
- Decomposition Methods
  - Cutting Plane Method (Outer Method)
  - Dantzig-Wolfe Method / Lagrangian Method (Inner Methods)
Bounding

Goal

Compute a lower bound on $z_{IP}$ by building an approximation to $P$.

- The most common approach is to use the LP relaxation.

$$z_{LP} = \min_{x \in Q} \{c^T x\} = \min_{x \in \mathbb{R}^n} \{c^T x \mid A'x \geq b', A''x \geq b''\}$$

- Decomposition methods attempt to improve on this bound by utilizing the fact that $OPT(c, \mathcal{P}')$ or $SEP(x, \mathcal{P}')$ is easy.

$$z_D = \min_{x \in \mathcal{P}'} \{c^T x \mid A''x \geq b''\} = \min_{x \in \mathcal{P}' \cap Q''} \{c^T x\} \geq z_{LP}$$

- $\mathcal{P}'$ is represented *implicitly* through solution of a subproblem.

Decomposition Methods

- Cutting Plane Method (Outer Method)
- Dantzig-Wolfe Method / Lagrangian Method (Inner Methods)
### Traditional Decomposition Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Master Equation</th>
<th>Subproblem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cutting Plane Method (CPM)</strong></td>
<td>CPM builds an <em>outer</em> approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$.</td>
<td>$z_{CP} = \min_{x \in \mathbb{R}^n} { c^T x \mid Dx \geq d }$</td>
<td>$SEP(x_{CP}, \mathcal{P}')$</td>
</tr>
<tr>
<td><strong>Dantzig-Wolfe Method</strong></td>
<td>DW builds an <em>inner</em> approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$.</td>
<td>$z_{DW} = \min_{\lambda \in \mathbb{R}^E_+} { c^T (\sum_{s \in E} s\lambda_s) \mid A''(\sum_{s \in E} s\lambda_s) \geq b'', \sum_{s \in E} \lambda_s = 1 }$</td>
<td>$OPT(c^T - u^T_{DW} A'', \mathcal{P}')$</td>
</tr>
<tr>
<td><strong>Lagrangian Method</strong></td>
<td>LD formulates a relaxation as finding the minimal extreme point of $\mathcal{P}'$ with respect to a cost which is penalized if the point lies outside of $\mathcal{Q}''$.</td>
<td>$z_{LD} = \max_{u \in \mathbb{R}^m''} { \min_{s \in E} { c^T s + u^T (b'' - A'' s) } }$</td>
<td>$OPT(c - u^T_{LD} A'', \mathcal{P}')$</td>
</tr>
</tbody>
</table>
Traditional Decomposition Methods

Cutting Plane Method (CPM)

CPM builds an outer approximation of $P'$ intersected with $Q''$.
- Master: $z_{CP} = \min_{x \in \mathbb{R}^n} \{c^T x \mid Dx \geq d\}$
- Subproblem: $SEP(x_{CP}, P')$

Dantzig-Wolfe Method

DW builds an inner approximation of $P'$ intersected with $Q''$.
- Master: $\bar{z}_{DW} = \min_{\lambda \in \mathbb{R}^E_+} \{c^T (\sum_{s \in E} s \lambda_s) \mid A''(\sum_{s \in E} s \lambda_s) \geq b'', \sum_{s \in E} \lambda_s = 1\}$
- Subproblem: $OPT(c^T - u_{DW}^T A'', P')$

Lagrangian Method

LD formulates a relaxation as finding the minimal extreme point of $P'$ with respect to a cost which is penalized if the point lies outside of $Q''$.
- Master: $z_{LD} = \max_{u \in \mathbb{R}^m_{++}} \{\min_{s \in E} \{c^T s + u^T (b'' - A'' s)\}\}$
- Subproblem: $OPT(c - u_{LD}^T A'', P')$
### Traditional Decomposition Methods

<table>
<thead>
<tr>
<th>Cutting Plane Method (CPM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPM builds an <em>outer</em> approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$.</td>
</tr>
<tr>
<td>Master: $z_{CP} = \min_{x \in \mathbb{R}^n} {c^T x \mid Dx \geq d}$</td>
</tr>
<tr>
<td>Subproblem: $SEP(x_{CP}, \mathcal{P}')$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dantzig-Wolfe Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW builds an <em>inner</em> approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$.</td>
</tr>
<tr>
<td>Master: $\bar{z}<em>{DW} = \min</em>{\lambda \in \mathbb{R}^E} {c^T (\sum_{s \in \mathcal{E}} s \lambda_s) \mid A''(\sum_{s \in \mathcal{E}} s \lambda_s) \geq b'', \sum_{s \in \mathcal{E}} \lambda_s = 1}$</td>
</tr>
<tr>
<td>Subproblem: $OPT(c^T - u_{DW} A'', \mathcal{P}')$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lagrangian Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD formulates a relaxation as finding the minimal extreme point of $\mathcal{P}'$ with respect to a cost which is penalized if the point lies outside of $\mathcal{Q}''$.</td>
</tr>
<tr>
<td>Master: $z_{LD} = \max_{u \in \mathbb{R}^{m''}} {\min_{s \in \mathcal{E}} {c^T s + u^T (b'' - A'' s)}}$</td>
</tr>
<tr>
<td>Subproblem: $OPT(c - u_{LD} A'', \mathcal{P}')$</td>
</tr>
</tbody>
</table>
The continuous approximation of $P$ is formed as the intersection of two explicitly defined polyhedra (both with a small description).

$$z_{LP} = \min_{x \in \mathbb{R}^n} \{ c^T x \mid x \in Q' \cap Q'' \}$$

Decomposition Methods form an approximation as the intersection of one explicitly defined polyhedron (with a small description) and one implicitly defined polyhedron (with a large description).

$$z_{CP} = z_{DW} = z_{LD} = z_D = \min_{x \in \mathbb{R}^n} \{ c^T x \mid x \in P' \cap Q'' \} \geq z_{LP}$$

Each of the traditional decomposition methods contain two primary steps
- Master Problem: Update the primal or dual solution information.
- Subproblem: Update the approximation of $P$: $SEP(x, P')$ or $OPT(c, P')$.

Integrated Decomposition Methods form an approximation as the intersection of two implicitly defined polyhedra (both with a large description).

So, we improve on the bound $z_D$ by building both an inner approximation $P_I$ of $P'$ intersected with some outer approximation $P_O \subset Q''$. 

---

*Galati, Ralphs*  
*DECOMP: A Framework for Decomposition in IP*
**Common Framework**

- The continuous approximation of $\mathcal{P}$ is formed as the intersection of two explicitly defined polyhedra (*both with a small description*).
  \[
  z_{LP} = \min_{x \in \mathbb{R}^n} \{ c^\top x \mid x \in Q' \cap Q'' \}
  \]

- Decomposition Methods form an approximation as the intersection of one explicitly defined polyhedron (*with a small description*) and one implicitly defined polyhedron (*with a large description*).
  \[
  z_{CP} = z_{DW} = z_{LD} = z_D = \min_{x \in \mathbb{R}^n} \{ c^\top x \mid x \in \mathcal{P}' \cap Q'' \} \geq z_{LP}
  \]

- Each of the traditional decomposition methods contain two primary steps
  - **Master Problem**: Update the primal or dual solution information.
  - **Subproblem**: Update the approximation of $\mathcal{P}$: $SEP(x, \mathcal{P}')$ or $OPT(c, \mathcal{P}')$.

- Integrated Decomposition Methods form an approximation as the intersection of two implicitly defined polyhedra (*both with a large description*).
  - So, we improve on the bound $z_D$ by building both an inner approximation $\mathcal{P}_I$ of $\mathcal{P}'$ intersected with some outer approximation $\mathcal{P}_O \subset Q''$. 

---

*Galati, Ralphs*  
*DECOMP: A Framework for Decomposition in IP*
The continuous approximation of $\mathcal{P}$ is formed as the intersection of two explicitly defined polyhedra (both with a small description).

$$z_{LP} = \min_{x \in \mathbb{R}^n} \left\{ c^T x \mid x \in Q' \cap Q'' \right\}$$

Decomposition Methods form an approximation as the intersection of one explicitly defined polyhedron (with a small description) and one implicitly defined polyhedron (with a large description).

$$z_{CP} = z_{DW} = z_{LD} = z_D = \min_{x \in \mathbb{R}^n} \left\{ c^T x \mid x \in \mathcal{P}' \cap Q'' \right\} \geq z_{LP}$$

Each of the traditional decomposition methods contain two primary steps

- **Master Problem:** Update the primal or dual solution information.
- **Subproblem:** Update the approximation of $\mathcal{P}$: $SEP(x, \mathcal{P}')$ or $OPT(c, \mathcal{P}')$.

Integrated Decomposition Methods form an approximation as the intersection of two implicitly defined polyhedra (both with a large description).

So, we improve on the bound $z_D$ by building both an inner approximation $\mathcal{P}_I$ of $\mathcal{P}'$ intersected with some outer approximation $\mathcal{P}_O \subset Q''$. 
Outline

1. Decomposition Methods
2. Integrated Decomposition Methods
3. DECOMP Framework
## Integrated Decomposition Methods

### Price and Cut (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:**

$$\bar{z}_{PC} = \min_{\lambda \in \mathbb{R}^E_+} \{c^T (\sum_{s \in E}s \lambda_s) \mid D(\sum_{s \in E}s \lambda_s) \geq d, \sum_{s \in E} \lambda_s = 1\}$$

**Subproblem:**

- **Pricing:** $OPT(c^T - u^T_{PC} D, \mathcal{P}')$, or
- **Cutting:** $SEP(x_{PC}, \mathcal{P})$

### Relax and Cut (RC)

RC improves on the bound $z_D$ using LD and augmenting the multiplier space with valid inequalities that are violated by the solution to the Lagrangian subproblem.

**Master:**

$$z_{RC} = \max_{u \in \mathbb{R}^{m''}_+} \{\min_{s \in E} \{c^T s + u^T (d - D s)\}\}$$

**Subproblem:**

- **Pricing:** $OPT(c - u^T_{LD} D, \mathcal{P}')$, or
- **Cutting:** $SEP(s, \mathcal{P})$
Integrated Decomposition Methods

Price and Cut (PC)

PC approximates $P$ by building an inner approximation of $P'$ (as in DW) intersected with an outer approximation of $P$ (as in CPM).

- **Master:**
  \[ z_{PC} = \min_{\lambda \in \mathbb{R}_+} \{ c^T (\sum_{s \in E} s\lambda_s) \mid D(\sum_{s \in E} s\lambda_s) \geq d, \sum_{s \in E} \lambda_s = 1 \} \]

- **Subproblem:**
  - Pricing: $OPT(c^T - u^T \cdot P')$, or
  - Cutting: $SEP(x_{PC}, P)$

Relax and Cut (RC)

RC improves on the bound $z_D$ using LD and augmenting the multiplier space with valid inequalities that are violated by the solution to the Lagrangian subproblem.

- **Master:**
  \[ z_{RC} = \max_{u \in \mathbb{R}_+^{m''}} \{ \min_{s \in E} \{ c^T s + u^T (d - Ds) \} \} \]

- **Subproblem:**
  - Pricing: $OPT(c - u^T \cdot LD, P')$, or
  - Cutting: $SEP(s, P)$
Structured Separation

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

Traveling Salesman Problem Formulation:

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

\( \mathcal{P}' = 1\)-Tree Relaxation: \( OPT(c, 1 - \text{Tree}) \) in \( O(|E| \log |V|) \)

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

\( \mathcal{P}' = 2\)-Matching Relaxation: \( OPT(c, 2 - \text{Match}) \) in polynomial time

\[
x(\delta(u)) = 2 \quad \forall u \in V \\
x_e \in \{0, 1\} \quad \forall e \in E
\]
Example - TSP

- Separation of Subtour Inequalities:

\[ x(E(S)) \leq |S| - 1 \]

- \( SEP(x, Subtour) \), for \( x \in \mathbb{R}^n \) can be solved in \( O(|V|^4) \) (Min-Cut)

- \( SEP(s, Subtour) \), for \( s \) a 2-matching, can be solved in \( O(|V|) \)

  - Simply determine the connected components \( C_i \), and set \( S = C_i \) for each component (each gives a violation of 1).
Example - TSP

- Separation of Subtour Inequalities:
  \[ x(E(S)) \leq |S| - 1 \]

- \( SEP(x, \text{Subtour}) \), for \( x \in \mathbb{R}^n \) can be solved in \( O(|V|^4) \) (Min-Cut)

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

- Separation of Subtour Inequalities:
  \[ x(E(S)) \leq |S| - 1 \]

- \( SEP(x, Subtour) \), for \( x \in \mathbb{R}^n \) can be solved in \( O(|V|^4) \) (Min-Cut)

- \( SEP(s, Subtour) \), for \( s \) a 2-matching, can be solved in \( O(|V|) \)

  - Simply determine the connected components \( C_i \), and set \( S = C_i \) for each component (each gives a violation of 1).
Example - TSP

- **Separation of Comb Inequalities:**

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

- \( SEP(x, Blossoms) \), for \( x \in \mathbb{R}^n \) can be solved in \( O(|V|^5) \) (Padberg-Rao)

- \( SEP(s, Blossoms) \), 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.
Example - TSP

- Separation of Comb Inequalities:

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

- \text{SEP}(x, Blossoms), for \( x \in \mathbb{R}^n \) can be solved in \( O(|V|^5) \) (Padberg-Rao)

- \text{SEP}(s, Blossoms), 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.
Example - TSP

- Separation of Comb Inequalities:

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

- \(SEP(x, Blossoms)\), for \(x \in \mathbb{R}^n\) can be solved in \(O(|V|^5)\) (Padberg-Rao)

- \(SEP(s, Blossoms)\), 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.
**Structured Separation Motivation**

- **Motivation:** RC, solutions to LR, \( s \in \mathcal{E} \), have some *nice* combinatorial structure.

- **New Algorithms:**
  - Price and Cut (Revisited) - see fig
  - Decomp and Cut - inverse process

Provides an alternative *necessary* (but not *sufficient*) condition to find an improving inequality which is very easy to implement and understand.

For some theoretical details see:

Motivation: RC, solutions to LR, \( s \in \mathcal{E} \), have some nice combinatorial structure.

New Algorithms:
- Price and Cut (Revisited) - see fig
- Decomp and Cut - inverse process

Provides an alternative necessary (but not sufficient) condition to find an improving inequality which is very easy to implement and understand.

For some theoretical details see:
Price and Cut (Revisited)

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

![Graphs showing subtours and lambda values](image)

- 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.
In the context of the traditional CPM, we can construct (inverse DW) the decomposition $\lambda$ from the current fractional solution $x_{CP}$ by solving the following LP

$$\max \left\{ 0^T \lambda \mid \sum_{s \in \mathcal{E}} s \lambda_s = x_{CP}, \sum_{s \in \mathcal{E}} \lambda_s = 1 \right\},$$

If we find a decomposition $D$, then we separate each $s \in D$, as in revised PC.

If we fail, then the LP proof of infeasibility (Farkas Cut) gives us a separating hyperplane which can be used to cut off the current fractional point.
Decomp and Cut

- In the context of the traditional CPM, we can construct \((\text{inverse DW})\) the decomposition \(\lambda\) from the current fractional solution \(x_{CP}\) by solving the following LP:

\[
\max_{\lambda \in \mathbb{R}^E_+} \left\{ 0^\top \lambda \mid \sum_{s \in E} s \lambda_s = x_{CP}, \sum_{s \in E} \lambda_s = 1 \right\},
\]

- If we find a decomposition \(D\), then we separate each \(s \in D\), as in revised PC.
- If we fail, then the LP proof of infeasibility (Farkas Cut) gives us a separating hyperplane which can be used to cut off the current fractional point.

\(\text{P’}\)

\(\text{P}\)

\(\text{P’}\)

\(\text{P}\)

\(s \in E : \lambda_s > 0\)

(a) \(x_{CP} \in \text{P’}\)

(b) \(x_{CP} \notin \text{P’}\)
Structured Separation - Useful?

**Case 1:** There exists a polynomial algorithm for $SEP(x, C)$.
- TSP/1-Tree: $SEP(x, Blossoms)$ Padberg-Rao, $SEP(s, Blossoms)$ BFS.
- Unlikely that SS will ever outperform a well written exact method, but very easy to implement.

**Case 2:** There exists no polynomial algorithm for $SEP(x, C)$.
- VRP,k-Tree: $SEP(x, GSECs)$ heuristics, $SEP(s, GSECs)$ in $O(|E|)$.
- SS provides an alternative and typically simple heuristic for separation.

**Case 3:** There exists no algorithm for $SEP(x, C)$, but ineq's in $C$ are facet-defining.
- KCCP,CP: $SEP(x, MaxSet)$ heuristics, $SEP(s, MaxSet)$ in $O(|E|)$.
- SS provides a starting point for constructing separation heuristics.

**Case 4:** We have a new problem class for which we are searching for new valid inequalities.
- Trying to analyze integral points (as in SS) seems much more promising.
Structured Separation - Useful?

- **Case 1**: There exists a polynomial algorithm for \( SEP(x, C) \).
  - TSP/1-Tree: \( SEP(x, Blossoms) \) Padberg-Rao, \( SEP(s, Blossoms) \) BFS.
  - Unlikely that SS will ever outperform a well written exact method, but very easy to implement.

- **Case 2**: There exists no polynomial algorithm for \( SEP(x, C) \).
  - VRP,k-Tree: \( SEP(x, GSECs) \) heuristics, \( SEP(s, GSECs) \) in \( O(|E|) \).
  - SS provides an alternative and typically simple heuristic for separation.

- **Case 3**: There exists no algorithm for \( SEP(x, C) \), but ineq's in \( C \) are facet-defining.
  - KCCP,CP: \( SEP(x, MaxSet) \) heuristics, \( SEP(s, MaxSet) \) in \( O(|E|) \).
  - SS provides a starting point for constructing separation heuristics.

- **Case 4**: We have a new problem class for which we are searching for new valid inequalities.
  - Trying to analyze integral points (as in SS) seems much more promising.
Structured Separation - Useful?

- **Case 1:** There exists a polynomial algorithm for $SEP(x, C)$.
  - TSP/1-Tree: $SEP(x, Blossoms)$ Padberg-Rao, $SEP(s, Blossoms)$ BFS.
  - Unlikely that SS will ever outperform a well written exact method, but very easy to implement.

- **Case 2:** There exists no polynomial algorithm for $SEP(x, C)$.
  - VRP, k-Tree: $SEP(x, GSECs)$ heuristics, $SEP(s, GSECs)$ in $O(|E|)$.
  - SS provides an alternative and typically simple heuristic for separation.

- **Case 3:** There exists no algorithm for $SEP(x, C)$, but ineq’s in $C$ are facet-defining.
  - KCCP, CP: $SEP(x, MaxSet)$ heuristics, $SEP(s, MaxSet)$ in $O(|E|)$.
  - SS provides a starting point for constructing separation heuristics.

- **Case 4:** We have a new problem class for which we are searching for new valid inequalities.
  - Trying to analyze integral points (as in SS) seems much more promising.
Structured Separation - Useful?

**Case 1:** There exists a polynomial algorithm for $SEP(x, C)$.
- TSP/1-Tree: $SEP(x, Blossoms)$ Padberg-Rao, $SEP(s, Blossoms)$ BFS.
- Unlikely that SS will ever outperform a well written exact method, but very easy to implement.

**Case 2:** There exists no polynomial algorithm for $SEP(x, C)$.
- VRP,k-Tree: $SEP(x, GSECs)$ heuristics, $SEP(s, GSECs)$ in $O(|E|)$.
- SS provides an alternative and typically simple heuristic for separation.

**Case 3:** There exists no algorithm for $SEP(x, C)$, but ineq’s in $C$ are facet-defining.
- KCCP,CP: $SEP(x, MaxSet)$ heuristics, $SEP(s, MaxSet)$ in $O(|E|)$.
- SS provides a starting point for constructing separation heuristics.

**Case 4:** We have a new problem class for which we are searching for new valid inequalities.
- Trying to analyze integral points (as in SS) seems much more promising.
Structured Separation - Applications

- **Steiner Tree Problem**
  - Minimum Spanning Tree: Lifted SECs, Partition - **RC** [Lucena 92]

- **Traveling Salesman Problem**
  - One-Tree: Blossoms, Combs
  - Matching: SECs

- **Vehicle Routing Problem**
  - k-Traveling Salesman Problem: GSECs - **DC** [Ralphs, et al. 03]
  - k-Tree: GSECs, Combs, Multistars - **RC** [Marthinhon, et al. 01]

- **Axial Assignment Problem**
  - Assignment Problem: Clique-Facets - **RC** [Balas, Saltzman 91]

- **Knapsack Constrained Circuit Problem**
  - Knapsack Problem: Cycle Cover, Maximal-Set Inequalities
  - Circuit Problem: Cycle Cover, Maximal-Set Inequalities

- **Edge-Weighted Clique Problem**
  - Tree Relaxation: Trees, Cliques - **RC** [Hunting, et al. 01]

- **Subtour Elimination Problem [G. Benoit / S. Boyd]**
  - Fractional 2-Factor Problem: SECs - **DC / LP Context** [Benoit, Boyd 03]
Multiple Polytopes

- Common paradigm - tighten bounds by solving $SEP(x, X)$, for various $X$.
  - $X$ is a polyhedron - the closure of the set of all cuts in a particular template.
  - Successive outer approximation from the intersection of various polyhedra.
  - TSP: $X =$ subtour closure, blossom closure, comb closure, etc...
  - MILP: $X =$ knapsack closure, flow cover closure, clique closure, etc...

- Common paradigm - column generation based on one oracle. Why only one?
- New paradigm - generate vars from multiple polytopes, solving $OPT(c, X)$, for various $X$.
  - Successive inner approximation from the intersection of various polyhedra.
  - TSP: $X =$ one-tree, 2-matching, TSP, etc...
Multiple Polytopes

- Common paradigm - tighten bounds by solving $SEP(x, X)$, for various $X$.
  - $X$ is a polyhedron - the closure of the set of all cuts in a particular template.
  - Successive outer approximation from the intersection of various polyhedra.
  - TSP: $X =$ subtour closure, blossom closure, comb closure, etc...
  - MILP: $X =$ knapsack closure, flow cover closure, clique closure, etc...

- Common paradigm - column generation based on one oracle **Why only one?**
  - New paradigm - generate vars from multiple polytopes, solving $OPT(c, X)$, for various $X$.
  - Successive inner approximation from the intersection of various polyhedra.
  - TSP: $X =$ one-tree, 2-matching, $TSP$, etc...
Multiple Polytopes

- Common paradigm - tighten bounds by solving $SEP(x, X)$, for various $X$.
  - $X$ is a polyhedron - the closure of the set of all cuts in a particular template.
  - Successive outer approximation from the intersection of various polyhedra.
  - TSP: $X = \text{subtour closure, blossom closure, comb closure, etc...}$
  - MILP: $X = \text{knapsack closure, flow cover closure, clique closure, etc...}$

- Common paradigm - column generation based on one oracle **Why only one?**
- New paradigm - generate vars from multiple polytopes, solving $OPT(c, X)$, for various $X$.
  - Successive inner approximation from the intersection of various polyhedra.
  - TSP: $X = \text{one-tree, 2-matching, TSP, etc...}$
DECOMP Framework

DECOMP provides a flexible software framework for testing and extending the theoretical framework presented thus far, with the primary goal of \textit{minimal user responsibility}.

- DECOMP was built around data structures and interfaces provided by COIN-OR:
  
  \textbf{CO}mputational \textbf{IN}frastructure for \textbf{O}perations \textbf{R}esearch

- BCP provides a framework for parallel PC with \textit{LP-Based Bounding}.
  
  A generalization of BCP currently under development:
  
  \textbf{ALPs}: Abstract Library for Parallel Search (INFORMS’06 - MD02)
  
  \textbf{BiCePS}: Branch, Constrain and Price \textit{[Generic Bounding]}
  
  \textbf{BLIS}: BiCePS Linear Integer Solver = BCP

- DECOMP will provide an implementation of the \textbf{BiCePS} layer.

- The DECOMP framework, written in C++, is accessed through two user interfaces:
  
  Applications Interface: DecompApp
  
  Algorithms Interface: DecompAlgo
The base class `DecompApp` provides all default algorithms: CPM, DW, LD, PC, RC, DC.

In order to develop an application, the user must derive the following methods/objects. All other methods have appropriate defaults but are virtual and may be overridden.

- `DecompApp::createModel()`. Define \([A'', b'']\) and \([A', b']\) (optional).
  - TSP: \([A'', b'']\) define the degree constraints. \([A', b']\) is empty.
- `DecompApp::isFeasible()`. Does \(x^*\) define a feasible solution?
  - TSP: do we have a feasible tour?
- `DecompApp::solveRelaxed()`. Provide a subroutine for \(OPT(c, P')\).
  - This is optional as well, if \([A', b']\) is defined (it will call the built in IP solver, currently CBC).
  - TSP 1-Tree: provide a solver for 1-tree.
  - TSP 2-Match: provide a solver for 2-matching.
- `DecompApp::generateCuts(s)`.
  - Provide a subroutine for \(SEP(s, C)\)
    - TSP 1-Tree: provide separation for comb/blossoms which violate a 1-tree (BFS).
    - TSP 2-Match: provide separation for subtours which violate a 2-matching (connect-comp).

To perform traditional CPM, if known, the user can also derive a subroutine to solve \(SEP(x, C)\), for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
The base class `DecompApp` provides all default algorithms: CPM, DW, LD, PC, RC, DC.

In order to develop an application, the user must derive the following methods/objects. All other methods have appropriate defaults but are virtual and may be overridden.

- **DecompApp::createModel().** Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP: $[A'', b'']$ define the degree constraints. $[A', b']$ is empty.

- **DecompApp::isFeasible().** Does $x^*$ define a feasible solution?
  - TSP: do we have a feasible tour?

- **DecompApp::solveRelaxed().** Provide a subroutine for $OPT(c,P')$.
  - This is optional as well, if $[A', b']$ is defined (it will call the built in IP solver, currently CBC).
  - TSP 1-Tree: provide a solver for 1-tree.
  - TSP 2-Match: provide a solver for 2-matching.

- **DecompApp::generateCuts(s).** Provide a subroutine for $SEP(s,C)$
  - TSP 1-Tree: provide separation for comb/blossoms which violate a 1-tree (BFS).
  - TSP 2-Match: provide separation for subtours which violate a 2-matching (connect-comp).

To perform traditional CPM, if known, the user can also derive a subroutine to solve $SEP(x,C)$, for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
The base class `DecompApp` provides all default algorithms: **CPM, DW, LD, PC, RC, DC.**

In order to develop an application, the user must derive the following methods/objects. All other methods have appropriate defaults but are *virtual* and may be overridden.

- **DecompApp::createModel().** Define \([A'', b'']\) and \([A', b']\) (optional).
  - TSP: \([A'', b'']\) define the degree constraints. \([A', b']\) is empty.

- **DecompApp::isFeasible().** Does \(x^*\) define a feasible solution?
  - TSP: do we have a feasible tour?

- **DecompApp::solveRelaxed().** Provide a subroutine for \(OPT(c, P')\).
  - This is optional as well, if \([A', b']\) is defined (it will call the built-in IP solver, currently CBC).
  - TSP 1-Tree: provide a solver for 1-tree.
  - TSP 2-Match: provide a solver for 2-matching.

- **DecompApp::generateCuts(s).** Provide a subroutine for \(SEP(s, C)\)
  - TSP 1-Tree: provide separation for comb/blossoms which violate a 1-tree (BFS).
  - TSP 2-Match: provide separation for subtours which violate a 2-matching (connect-comp).

To perform traditional CPM, if known, the user can also derive a subroutine to solve \(SEP(x, C)\), for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
The base class DecompApp provides all default algorithms: CPM, DW, LD, PC, RC, DC.

In order to develop an application, the user must derive the following methods/objects. All other methods have appropriate defaults but are virtual and may be overridden.

- **DecompApp::createModel().** Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP: $[A'', b'']$ define the degree constraints. $[A', b']$ is empty.

- **DecompApp::isFeasible().** Does $x^*$ define a feasible solution?
  - TSP: do we have a feasible tour?

- **DecompApp::solveRelaxed().** Provide a subroutine for $OPT(c, P')$.
  - This is optional as well, if $[A', b']$ is defined (it will call the built in IP solver, currently CBC).
  - TSP 1-Tree: provide a solver for 1-tree.
  - TSP 2-Match: provide a solver for 2-matching.

- **DecompApp::generateCuts(s).** Provide a subroutine for $SEP(s, C)$
  - TSP 1-Tree: provide separation for comb/blossoms which violate a 1-tree (BFS).
  - TSP 2-Match: provide separation for subtours which violate a 2-matching (connect-comp).

To perform traditional CPM, if known, the user can also derive a subroutine to solve $SEP(x, C)$, for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
The base class **DecompApp** provides all default algorithms: **CPM, DW, LD, PC, RC, DC**.

In order to develop an application, the user must derive the following methods/objects. All other methods have appropriate defaults but are **virtual** and may be overridden.

- **DecompApp::createModel()**: Define \([A'', b'']\) and \([A', b']\) (optional).
  - TSP: \([A'', b'']\) define the degree constraints. \([A', b']\) is empty.
- **DecompApp::isFeasible()**: Does \(x^*\) define a feasible solution?
  - TSP: do we have a feasible tour?
- **DecompApp::solveRelaxed()**: Provide a subroutine for \(OPT(c, P')\).
  - This is optional as well, if \([A', b']\) is defined (it will call the built in IP solver, currently CBC).
  - TSP 1-Tree: provide a solver for 1-tree.
  - TSP 2-Match: provide a solver for 2-matching.
- **DecompApp::generateCuts(s)**: Provide a subroutine for \(SEP(s, C)\)
  - TSP 1-Tree: provide separation for comb/blossoms which violate a 1-tree (BFS).
  - TSP 2-Match: provide separation for subtours which violate a 2-matching (connect-comp).

To perform traditional CPM, if known, the user can also derive a subroutine to solve \(SEP(x, C)\), for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
The base class `DecompApp` provides all default algorithms: CPM, DW, LD, PC, RC, DC.

In order to develop an application, the user must derive the following methods/objects. All other methods have appropriate defaults but are virtual and may be overridden.

- **DecompApp::createModel().** Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP: $[A'', b'']$ define the degree constraints. $[A', b']$ is empty.

- **DecompApp::isFeasible().** Does $x^*$ define a feasible solution?
  - TSP: do we have a feasible tour?

- **DecompApp::solveRelaxed().** Provide a subroutine for $OPT(c, P')$.
  - This is optional as well, if $[A', b']$ is defined (it will call the built in IP solver, currently CBC).
  - TSP 1-Tree: provide a solver for 1-tree.
  - TSP 2-Match: provide a solver for 2-matching.

- **DecompApp::generateCuts(s).** Provide a subroutine for $SEP(s, C)$
  - TSP 1-Tree: provide separation for comb/blossoms which violate a 1-tree (BFS).
  - TSP 2-Match: provide separation for subtours which violate a 2-matching (connect-comp).

To perform traditional CPM, if known, the user can also derive a subroutine to solve $SEP(x, C)$, for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
A key feature of DECOMP is that the user only needs to provide methods for their application in the original space ($x$-space), rather than in the space of a particular reformulation.

Automatic reformulation allows for users to consider cuts and variables in their most intuitive form and greatly simplifies the process of expansion into rows and columns. This is a major differentiator for DECOMP, compared to others BCP, ABACUS, MINTO, etc.

Structured separation allows for fast and easy prototyping without the need for implementation of difficult separation routines.

Features:

- One interface to all default algorithms: CPM/DC, DW, LD, PC, RC.
- Built on top of the COIN/OSI interface, so easily interchangeable LP solvers.
- Column generation based on multiple polytopes can be easily defined and employed.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, parm file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).
A key feature of DECOMP is that the user only needs to provide methods for their application in the original space (\(x\)-space), rather than in the space of a particular reformulation.

Automatic reformulation allows for users to consider cuts and variables in their most intuitive form and greatly simplifies the process of expansion into rows and columns.

This is a major differentiator for DECOMP, compared to others BCP, ABACUS, MINTO, etc.

Structured separation allows for fast and easy prototyping without the need for implementation of difficult separation routines.

Features:

- One interface to all default algorithms: CPM/DC, DW, LD, PC, RC.
- Built on top of the COIN/OSI interface, so easily interchange LP solvers.
- Column generation based on multiple polytopes can be easily defined and employed.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, parm file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).
A **key feature** of DECOMP is that the user only needs to provide methods for their application in the original space (\(x\)-space), rather than in the space of a particular reformulation.  

**Automatic reformulation** allows for users to consider cuts and variables in their most *intuitive* form and greatly simplifies the process of expansion into rows and columns.  

This is a major differentiator for **DECOMP**, compared to others **BCP, ABACUS, MINTO, etc.**  

Structured separation allows for fast and easy prototyping without the need for implementation of difficult separation routines.  

**Features:**  
- One interface to all default algorithms: **CPM/DC, DW, LD, PC, RC.**  
- Built on top of the COIN/OSI interface, so easily interchange LP solvers.  
- Column generation based on multiple polytopes can be easily defined and employed.  
- Active LP compression, variable and cut pool management.  
- Flexible parameter interface: command line, parm file, direct call overrides.  
- Visualization tools for graph problems (linked to graphviz).
DECOMP - TSP Example

TSP_DecomApp

class TSP_DecomApp : public DecomApp {
    // Define modelCore as 2-degree constraints, modelRelax empty.
    void APPcreateModel(double * & objCoeff,
                         vector<DecompConstraintSet*> & modelCore,
                         vector<DecompConstraintSet*> & modelRelax);

    // A 2-matching solver, a 1-tree solver.
    decompStat APPsolveRelaxed(const int whichModel,
                                const double * redCostX,
                                const double * origCost,
                                list<DecompVar*> & vars);

    // Structured separation routines:
    // connected components for subtours
    // BFS for blossoms and general combs
    virtual int generateCuts(const double * x,
                              const DecompVar & var,
                              DecompCutList & new_cuts);

    // Standard separation routines:
    // 1.) min-cut for subtours
    // 2.) padberg-rao for blossoms
    // 3.) heuristics for general combs
    int generateCuts(const double * x,
                     const DecompConstraintSet & modelCore,
                     const DecompConstraintSet & modelRelax,
                     DecompCutList & newCuts);

    // Does x define a tour?
    bool isFeasible(const double * x,
                    const int nCols,
                    DecompCutList & newCuts);

    ...}

Galati, Ralphs  DECOMP: A Framework for Decomposition in IP
The base class `DecompAlgo` provides the shell (init / master / subproblem / update).

Each of the methods described have derived default implementations `DecompAlgoX`:

```java
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** or **ACCPM**.
- The user might choose to add some advanced stabilizing factor to the dual updates in LD, as in **bundle methods**.
- The user might choose the **Volume algorithm** for solving the LD, which provides an approximate primal solution, for which cuts can be generated.
- Hybrid methods like using LD to initialize the columns of the DW master.
- During PC, adding cuts to both $P^{t+1}_O$ and $P^{t+1}_I$ simultaneously (Vanderbeck).
```c
int main(int argc, char ** argv){
    UtilApp utilApp(argc, argv);

    // create the user application (a DecompApp)
    TSP_Decomapp tsp(utilApp);
    tsp.createModel();

    // create instances of each algorithm and solve
    DecompAlgoC algoC(&tsp);
    algoC.initSetup(TSP_Decomapp::BASIC);
    algoC.processNode();

    DecompAlgoPC algoPC(&tsp);
    algoPC.initSetup(TSP_Decomapp::TWO_MATCH);
    algoPC.processNode();
    algoPC.refresh();
    algoPC.initSetup(TSP_Decomapp::ONE_TREE);
    algoPC.processNode();

    DecompAlgoRC algoRC(&tsp);
    algoRC.initSetup(TSP_Decomapp::TWO_MATCH);
    algoRC.processNode();
    algoRC.refresh();
    algoRC.initSetup(TSP_Decomapp::ONE_TREE);
    algoRC.processNode();
}
```
XXX_DecompApp is also built as a **callable library**

- **BcpsDecompModel** : public BcpsModel
  - a wrapper class that calls (data access) methods from DecompApp

- **BcpsDecompTreeNode** : public BcpsDecompTreeNode
  - a wrapper class that calls (algorithmic) methods from DecompAlgo

**Minimal** additional user responsibilities:
- **(optional)** branching decisions - defaults are straightforward since we work in the original $x$-space
  - each built-in DecompAlgoXXX provides a default BiCePs branching routine
  - default RC branching assumes use of volume or bundle (need primal solutions)

- **(not optional)** enforcing branching decisions in subproblem oracle

- **(not optional)** encoding/decoding Decomp objects for distributed processing
Traditional Decomposition Methods approximate $\mathcal{P}$ as $\mathcal{P}' \cap \mathcal{Q}''$.  
- $\mathcal{P}' \supset \mathcal{P}$ may have a large description.

Integrated Decomposition Methods approximate $\mathcal{P}$ as $\mathcal{P}_I \cap \mathcal{P}_O$.  
- Both $\mathcal{P}_I \subset \mathcal{P}'$ and $\mathcal{P}_O \supset \mathcal{P}$ may have a large description.

Structured separation can be much easier than general separation.  
- Two new techniques based on SS: revised-PC and DC

A new paradigm for column generation using multiple oracles.  

DECOMP provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.  
- The key to ease-of-use, is that the user can stay in the original (intuitive) space.

The interface to ALPS allows us to investigate large-scale problems on distributed networks.  

The code is open-source, currently released under CPL and will soon be available through the COIN-OR project repository www.coin-or.org.
Traditional Decomposition Methods approximate $\mathcal{P}$ as $\mathcal{P}' \cap \mathcal{Q}''$. $\mathcal{P}' \supset \mathcal{P}$ may have a large description.

Integrated Decomposition Methods approximate $\mathcal{P}$ as $\mathcal{P}_I \cap \mathcal{P}_O$. Both $\mathcal{P}_I \subset \mathcal{P}'$ and $\mathcal{P}_O \supset \mathcal{P}$ may have a large description.

Structured separation can be much easier than general separation.
- Two new techniques based on SS: revised-PC and DC.

A new paradigm for column generation using multiple oracles.
- DECOMP provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The key to ease-of-use, is that the user can stay in the original (intuitive) space.
- The interface to ALPS allows us to investigate large-scale problems on distributed networks.
- The code is open-source, currently released under CPL and will soon be available through the COIN-OR project repository www.coin-or.org.
Summary

- Traditional Decomposition Methods approximate $P$ as $P' \cap Q''$.
  - $P' \supset P$ may have a large description.

- Integrated Decomposition Methods approximate $P$ as $P_I \cap P_O$.
  - Both $P_I \subset P'$ and $P_O \supset P$ may have a large description.

- Structured separation can be much easier than general separation.
  - Two new techniques based on SS: revised-PC and DC

- A new paradigm for column generation using multiple oracles.
  - DECOMP provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
    - The key to ease-of-use, is that the user can stay in the original (intuitive) space.
  
  - The interface to ALPS allows us to investigate large-scale problems on distributed networks.
  
  - The code is open-source, currently released under CPL and will soon be available through the COIN-OR project repository www.coin-or.org.
Summary

- Traditional Decomposition Methods approximate $P$ as $P' \cap Q''$.
  - $P' \supset P$ may have a large description.

- Integrated Decomposition Methods approximate $P$ as $P_I \cap P_O$.
  - Both $P_I \subset P'$ and $P_O \supset P$ may have a large description.

- **Structured separation** can be much easier than general separation.
  - Two new techniques based on SS: revised-PC and DC

- A **new paradigm** for column generation using multiple oracles.

- **DECOMP** provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The key to ease-of-use, is that the user can stay in the original (intuitive) space.

- The interface to ALPS allows us to investigate large-scale problems on distributed networks.

- The code is open-source, currently released under CPL and will soon be available through the COIN-OR project repository www.coin-or.org.
Summary

- Traditional Decomposition Methods approximate \( P \) as \( P' \cap Q'' \).
  - \( P' \supset P \) may have a \textit{large} description.

- Integrated Decomposition Methods approximate \( P \) as \( P_I \cap P_O \).
  - Both \( P_I \subset P' \) and \( P_O \supset P \) may have a \textit{large} description.

- \textbf{Structured separation} can be much easier than general separation.
  - Two new techniques based on SS: revised-PC and DC

- A \textbf{new paradigm} for column generation using multiple oracles.

- \textbf{DECOMP} provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The key to \textit{ease-of-use}, is that the user can stay in the original (intuitive) space.

- The interface to \textbf{ALPS} allows us to investigate large-scale problems on distributed networks.

- The code is open-source, currently released under CPL and will soon be available through the \textbf{COIN-OR} project repository \url{www.coin-or.org}. 