DECOMP: A Framework for Decomposition in Integer Programming

Matthew Galati\textsuperscript{1} \quad Ted Ralphs\textsuperscript{2}

\textsuperscript{1}SAS Institute – Analytical Solutions – Operations Research and Development

\textsuperscript{2}Lehigh University – Department of Industrial and Systems Engineering

INFORMS 2005
Open-Source Tools for Mixed Integer Programming
Outline

1. Decomposition Methods
2. Integrated Decomposition Methods
3. Structured Separation
4. DECOMP Framework
Outline

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

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

where

\[
\begin{align*}
\mathcal{F} &= \{ x \in \mathbb{Z}^n \mid A'x \geq b', A''x \geq b'' \} \\
\mathcal{F}' &= \{ x \in \mathbb{Z}^n \mid A'x \geq b' \} \\
Q &= \{ x \in \mathbb{R}^n \mid A'x \geq b', A''x \geq b'' \} \\
Q' &= \{ x \in \mathbb{R}^n \mid A'x \geq b' \} \\
Q'' &= \{ x \in \mathbb{R}^n \mid A''x \geq b'' \}
\end{align*}
\]

- Denote \( \mathcal{P} = \text{conv}(\mathcal{F}) \) and \( \mathcal{P}' = \text{conv}(\mathcal{F}') \).
- \( \text{OPT}(c, X) \): 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).
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(\delta{u})$</td>
<td>$= 2$ for all $u \in V$</td>
</tr>
<tr>
<td>$x(E(S))$</td>
<td>$\leq</td>
</tr>
<tr>
<td>$x_e$</td>
<td>$\in {0, 1}$ for all $e \in E$</td>
</tr>
</tbody>
</table>

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

**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**

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

- 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 (\(Q''\)).
**Bounding**

**Goal**

Compute a **lower bound** on $z_{IP}$ 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 $\mathcal{P}$.

- 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)
Cutting Plane Method (CPM)

CPM approximates $\mathcal{P}$ by building an *outer* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$.

- Let $[D, d]$ denote the facets of $\mathcal{P}'$, so that $\mathcal{P}' = \{x \in \mathbb{R}^n \mid D x \geq d\}$.
- The method converges to the bound $z_{CP} = c^T \tilde{x}_{CP} = z_D$.

**Initialize:** Form outer approximation $[D^0, d^0] = [A'', b'']$

$$\mathcal{P}_O^0 = \{x \in \mathbb{R}^n \mid D^0 x \geq d^0\} \supseteq \mathcal{P}'$$

**Master Problem:** Obtain optimal *primal* solution $x_{CP}^t$

$$z_{CP}^t = \min_{x \in \mathbb{R}^n} \{c^T x \mid D^t x \geq d^t\} = \min_{x \in \mathcal{P}_O^t} \{c^T x\}$$

**Subproblem:** Call $SEP(x_{CP}^t, \mathcal{P}')$ to generate vi’s for $\mathcal{P}$, violated by $x_{CP}^t$.

**Update:** If found, form a new outer approximation, and go to Step 2

$$\mathcal{P}_O^{t+1} = \{x \in \mathbb{R}^n \mid D^{t+1} x \leq d^{t+1}\} \supseteq \mathcal{P}'$$
Cutting Plane Method

Cutting Plane Method (CPM)

CPM approximates $\mathcal{P}$ by building an \textit{outer} approximation of $\mathcal{P}'$ intersected with $Q''$.

- Let $[D, d]$ denote the facets of $\mathcal{P}'$, so that $\mathcal{P}' = \{x \in \mathbb{R}^n \mid D x \geq d\}$.
- The method converges to the bound $z_{CP} = c^\top \hat{x}_{CP} = z_D$.

\begin{itemize}
  \item **Initialize**: Form outer approximation $[D^0, d^0] = [A'', b'']$
  \begin{align*}
    \mathcal{P}_O^0 &= \{x \in \mathbb{R}^n \mid D^0 x \geq d^0\} \supseteq \mathcal{P}'
  \end{align*}

  \item **Master Problem**: Obtain optimal \textit{primal} solution $x_{CP}^t$
  \begin{align*}
    z_{CP}^t &= \min_{x \in \mathbb{R}^n} \{ c^\top x \mid D^t x \geq d^t \} = \min_{x \in \mathcal{P}_O^t} \{ c^\top x \}
  \end{align*}

  \item **Subproblem**: Call $SEP(x_{CP}^t, \mathcal{P}')$ to generate vi’s for $\mathcal{P}$, violated by $x_{CP}^t$

  \item **Update**: If found, form a new outer approximation, and go to Step 2
  \begin{align*}
    \mathcal{P}_O^{t+1} &= \{x \in \mathbb{R}^n \mid D^{t+1} x \leq d^{t+1}\} \supseteq \mathcal{P}'
  \end{align*}
\end{itemize}
Dantzig-Wolfe Method (DW)

Dantzig-Wolfe Method

DW approximates $P$ by building an inner approximation of $P'$ intersected with $Q''$.

- Let $E$ denote the extreme points of $P'$, so that
  \[ P' = \{ x \in \mathbb{R}^n \mid x = \sum_{s \in E} s \lambda_s, \sum_{s \in E} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in E \}. \]

- The method converges to the bound $z_{DW} = c^T (\sum_{s \in E} s \hat{\lambda}_s) = c^T \hat{x}_{DW} = z_D$.

Initialize: Form inner approximation $E^0 \subset E$

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

Master Problem: Obtain optimal dual solution $(u^t_{DW}, \alpha^t_{DW})$

\[ z^t_{DW} = \min_{\lambda \in \mathbb{R}^t_+} \{ c^T (\sum_{s \in E^t} s \lambda_s) \mid A'' (\sum_{s \in E^t} s \lambda_s) \geq b'', \sum_{s \in E^t} \lambda_s = 1 \} \]

Subproblem: Call $OPT(c^T - u^t_{DW} A'', P')$, to generate ep's with $rc(s) < 0$

Update: If found, form a new inner approximation, and go to Step 2

\[ P^{t+1}_I = \{ x \in \mathbb{R}^n \mid x = \sum_{s \in E^{t+1}} s \lambda_s, \sum_{s \in E^{t+1}} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in E^{t+1} \} \subseteq P' \]
Dantzig-Wolfe Method (DW)

**Dantzig-Wolfe Method**

DW approximates $\mathcal{P}$ by building an *inner* approximation of $\mathcal{P}'$ intersected with $\mathcal{Q}''$.

Let $\mathcal{E}$ denote the extreme points of $\mathcal{P}'$, so that
$$\mathcal{P}' = \{ x \in \mathbb{R}^n \mid x = \sum_{s \in \mathcal{E}} s \lambda_s, \sum_{s \in \mathcal{E}} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in \mathcal{E} \}.$$

The method converges to the bound $z_{DW} = c^T (\sum_{s \in \mathcal{E}} s \hat{\lambda}_s) = c^T \hat{x}_{DW} = z_D$

**Initialize**: Form inner approximation $\mathcal{E}^0 \subset \mathcal{E}$

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

**Master Problem**: Obtain optimal *dual* solution $(u_{DW}^t, \alpha_{DW}^t)$

$$\bar{z}_{DW}^t = \min_{\lambda \in \mathbb{R}^{\mathcal{E}^t}_+} \{ c^T \left( \sum_{s \in \mathcal{E}^t} s \lambda_s \right) \mid A'' \left( \sum_{s \in \mathcal{E}^t} s \lambda_s \right) \geq b'', \sum_{s \in \mathcal{E}^t} \lambda_s = 1 \}$$

**Subproblem**: Call $OPT(c^T - u_{DW}^t A'', \mathcal{P}')$, to generate ep’s with $rc(s) < 0$

**Update**: If found, form a new inner approximation, and go to Step 2

$$\mathcal{P}_{I}^{t+1} = \{ x \in \mathbb{R}^n \mid x = \sum_{s \in \mathcal{E}^{t+1}} s \lambda_s, \sum_{s \in \mathcal{E}^{t+1}} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in \mathcal{E}^{t+1} \} \subseteq \mathcal{P}'$$
Lagrangian Method

(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 $\mathcal{Q}''$.

- The Lagrangian Dual is a piecewise-linear concave function
  \[ z_{LD} = \max_{\alpha \in \mathbb{R}^n, u \in \mathbb{R}^{m''}} \{ \min_{s \in \mathcal{E}} \{ c^T s + u^T (b'' - A'' s) \} \} \]

- Rewriting LD as an LP gives the dual of the DW-LP
  \[ z_{LD} = \max_{\alpha \in \mathbb{R}^n, u \in \mathbb{R}^{m''}} \{ \alpha + b''^T u \mid \alpha \leq (c^T - u^T A'') s \forall s \in \mathcal{E} \} \]

- So, $z_{LD} = z_{DW}$ and LD also achieves the bound $z_{D}$.

\begin{itemize}
  \item Initialize: Define $s^0 \in \mathcal{E}$, initialize dual multipliers $u_{LD}^0$ for $[A'', b'']$.
  \item Master Problem: Update the dual multipliers using directional info $s^t$.
  \item Subproblem: Call $OPT(c - (u_{LD}^t)^T A'', \mathcal{P}')$, to obtain a new direction $s^{t+1} \in \mathcal{E}$. If the stopping criterion is not met, go to Step 2.
\end{itemize}
Lagrangian Method (LD)

(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 $\mathcal{Q}''$.

- The Lagrangian Dual is a piecewise-linear concave function
  \[ z_{LD} = \max_{u \in \mathbb{R}^m_+, s \in \mathcal{E}} \{ \min \{ c^\top s + u^\top (b'' - A'' s) \} \} \]

- Rewriting LD as an LP gives the dual of the DW-LP
  \[ z_{LD} = \max_{\alpha \in \mathbb{R}, u \in \mathbb{R}^m_+} \{ \alpha + b''^\top u \mid \alpha \leq (c^\top - u^\top A'') s \forall s \in \mathcal{E} \} \]

- So, $z_{LD} = z_{DW}$ and LD also achieves the bound $z_D$.

1. **Initialize**: Define $s^0 \in \mathcal{E}$, initialize dual multipliers $u^0_{LD}$ for $[A'', b'']$.
2. **Master Problem**: Update the dual multipliers using directional info $s^t$.
3. **Subproblem**: Call $OPT(c - (u^t_{LD})^\top A'', \mathcal{P}')$, to obtain a new direction $s^{t+1} \in \mathcal{E}$. If the stopping criterion is not met, go to Step 2.
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).

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

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 $P_O \subset Q''$. 
The **continuous approximation** of $\mathcal{P}$ is formed as the intersection of two explicitly defined polyhedra (*both with a small description*).

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

**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*).

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

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 $\mathcal{P}_O \subset Q''$. 
The continuous approximation of $\mathcal{P}$ is formed as the intersection of two explicitly defined polyhedra (both with a small description).

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

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).

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

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
Price and Cut

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).

Initialize: Form inner $E^0 \subset E$ and outer $[D^0, d^0] = [A'', b'']$ approximations

$$P^0_I = \{ x \in \mathbb{R}^n \mid x = \sum_{s \in E^0} s \lambda_s, \sum_{s \in E^0} \lambda_s = 1, \lambda_s \geq 0 \forall s \in E^0 \} \subseteq P' \quad P^0_O = \{ x \in \mathbb{R}^n \mid D^0 x \geq d^0 \} \supseteq P$$

Master Problem: Solve the DW-LP to obtain optimal *dual* $(u^t_{PC}, \alpha^t_{PC})$, decomposition $\lambda^t_{PC}$, and *primal* solution $x^t_{PC}$.

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

Do either:

- **Pricing Subproblem and Update**: Call $OPT(c^T - u^t_{PC} D^t, P^t)$, to generate e.p.s with $rc(s) < 0$. If found, form a new inner approximation $P^t_{I+1}$, and go to Step 2.

- **Cutting Subproblem and Update**: Call $SEP(x^t_{PC}, P)$ to generate v.i.s violated by $x^t_{PC}$. If found, form a new outer approximation $P^t_{O+1}$, and go to Step 2.
Price and Cut

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).

**Initialize:** Form inner $\mathcal{E}^0 \subseteq \mathcal{E}$ and outer $[D^0, d^0] = [A'', b'']$ approximations

\[
\mathcal{P}_I^0 = \{ x \in \mathbb{R}^n \mid x = \sum_{s \in \mathcal{E}^0} s \lambda_s, \sum_{s \in \mathcal{E}^0} \lambda_s = 1, \lambda_s \geq 0 \ \forall s \in \mathcal{E}^0 \} \subseteq \mathcal{P}' \\
\mathcal{P}_O^0 = \{ x \in \mathbb{R}^n \mid D^0 x \geq d^0 \} \supseteq \mathcal{P}
\]

**Master Problem:** Solve the DW-LP to obtain optimal dual $(u_{PC}^t, \alpha_{PC}^t)$, decomposition $\lambda_{PC}^t$, and primal solution $x_{PC}^t$.

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

Do either:

1. **Pricing Subproblem and Update:** Call $OPT(c^T - u_{PC}^t D^t, \mathcal{P})$, to generate e.p.s with $rc(s) < 0$. If found, form a new inner approximation $\mathcal{P}_I^{t+1}$, and go to Step 2.

2. **Cutting Subproblem and Update:** Call $SEP(x_{PC}^t, \mathcal{P})$ to generate v.i.s violated by $x_{PC}^t$. If found, form a new outer approximation $\mathcal{P}_O^{t+1}$, and go to Step 2.
(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.

**Initialize:** Define $s^0 \in \mathcal{E}$, $[D^0, d^0] = [A''', b''']$, initialize dual multipliers $u^0_{LD}$ for $[D^0, d^0]$.

**Master Problem:** Update the dual multipliers using directional info $s^t$.

**Do either:**
- **Pricing Subproblem:** Call $OPT(c - u^t_{LD} D^t, P')$, to obtain a new direction $s^{t+1} \in \mathcal{E}$. If the stopping criterion is not met, go to Step 2.
- **Cutting Subproblem:** Call $SEP(s^t, P)$ to generate v.i.s violated by $s^t$. If found, add them to $[D^t, d^t]$ along with new dual multipliers, and go to Step 2.
Relax and Cut

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.

**Initialize:** Define $s^0 \in \mathcal{E}$, $[D^0, d^0] = [A'', b'']$, initialize dual multipliers $u_{LD}^0$ for $[D^0, d^0]$.

**Master Problem:** Update the dual multipliers using directional info $s^t$.

**Do either:**

- **Pricing Subproblem:** Call $OPT(c - u_{LD}^t D^t, \mathcal{P}')$, to obtain a new direction $s^{t+1} \in \mathcal{E}$. If the stopping criterion is not met, go to Step 2.

- **Cutting Subproblem:** Call $SEP(s^t, \mathcal{P})$ to generate v.i.s violated by $s^t$. If found, add them to $[D^t, d^t]$ along with new dual multipliers, and go to Step 2.
Outline

1. Decomposition Methods
2. Integrated Decomposition Methods
3. Structured Separation
4. DECOMP Framework
In general, the complexity of \( OPT(c, X) = SEP(x, X) \).

**Observation:** Restrictions on the input or output of these subroutines 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 \( \mathcal{P} \) be the convex hull of solutions to the TSP.

- \( SEP(x, \mathcal{P}) \) is \( NP \)-Complete.
- \( SEP(x, C) \) is polynomially solvable, for \( C \supset \mathcal{P} \) 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:

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

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

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

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

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

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

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

\[ \text{SEP}(s, \text{Subtour}), \text{ for } s \text{ a 2-matching, can be solved in } O(|V|) \]

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:**

\[ x(E(H)) + \sum_{i=1}^{k} x(E(T_i)) \leq |H| + \sum_{i=1}^{k}(|T_i| - 1) - \left\lfloor k/2 \right\rfloor \]

- **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 \(\lfloor k/2 \rfloor - 1\)).
  - 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:**

\[
x(E(H)) + \sum_{i=1}^{k} x(E(T_i)) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \left\lfloor k/2 \right\rfloor
\]

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

- \textit{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 \(\left\lfloor k/2 \right\rfloor - 1\)).
  - 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.
Motivation

- In Relax and Cut, solutions to the Lagrangian subproblem $s \in \mathcal{E}$ typically have some nice combinatorial structure. So, in RC, $SEP(s, \mathcal{P})$, can be relatively easy as opposed to general separation.

**Question:** Can we take advantage of this in other contexts?

- LP theory says in order to improve the bound, it is necessary and sufficient to cut off the entire face of optimal solutions $F'$.

- This condition is difficult to verify, so we typically use the necessary condition that the generated inequality be violated by some member of that face, $x \in F'$.

  - In CPM, we solve $SEP(x^t_{CP}, \mathcal{P})$, where $x^t_{CP} \in F^t$, and $F^t$ is optimal face over $\mathcal{P}_O^t \cap Q''$.
  - In PC, we solve $SEP(x^t_{PC}, \mathcal{P})$, where $x^t_{PC} \in F^t$, and $F^t$ is optimal face over $\mathcal{P}_I^t \cap \mathcal{P}_O^t$.
Motivation

- Now, consider the following set
  \[ S(u, \alpha) = \{ s \in \mathcal{E} \mid (c^T - u^T A'')s = \alpha \} , \]

- Then, \( S(u^t_{DW}, \alpha^t_{DW}) \) is the set ep’s with \( rc(s) = 0 \) in the DW-LP master or the set of alternative optimal solutions to the Lagrangian subproblem.

Theorems

1. \( F^t \subseteq \text{conv}(S(u^t_{DW}, \alpha^t_{DW})) \).
   - Separation of \( S(u^t_{DW}, \alpha^t_{DW}) \) is also necessary and sufficient.

2. \((a, \beta) \in \mathbb{R}^{(n+1)}\) improving \( \Rightarrow \exists s \in D = \{ s \in \mathcal{E} \mid \lambda^t_s > 0 \} \) s.t. \( a^T s < \beta \).
   - Every improving ineq must violate at least one e.p. in the optimal decomposition.

3. \( D = \{ s \in \mathcal{E} \mid \lambda^t_s > 0 \} \subseteq S(u^t_{PC}, \alpha^t_{PC}) \).
   - The optimal decomposition is contained in \( S \).

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

**Key Idea:** In the cutting subproblem, rather than (or in addition to) separating \( x^t_{PC} \), separate each \( s \in D \).
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.
Decomposition Methods
Integrated Decomposition Methods
Structured Separation
DECOMP Framework

**Decomp and Cut**

- 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:

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

- 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.

### Diagram:

- Valid inequality for $\mathcal{P}$: $\mathcal{P}'$
- Farkas inequality:
  - (a) $x_{CP} \in \mathcal{P}'$
  - (b) $x_{CP} \notin \mathcal{P}'$
Decomp and Cut

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:

\[
\max \{ 0^T \lambda | \sum s \lambda_s = x_{CP}, \sum \lambda_s = 1 \}, \quad \lambda \in \mathbb{R}^E_+
\]

If we find a decomposition \( \mathcal{D} \), then we separate each \( s \in \mathcal{D} \), 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.

---

(a) \( x_{CP} \in \mathcal{P}' \)

(b) \( x_{CP} \notin \mathcal{P}' \)
In the context of PC or DC, Theorem 2 provides an alternative *necessary* condition to finding an improving inequality.

SS alone, is *not sufficient* for finding an improving inequality. 

**SS alone, is *not sufficient* for separating the original fractional \( \hat{x} \).**

- Unless we enumerate all cuts from a class which violate each \( s \in \mathcal{D} \).

SS does provide an *easy to implement and understand* alternative method for what is typically a difficult task (general separation).

In addition, one can quickly separate several classes of cuts using one decomposition (Example: VRP/k-Tree GSECs and Combs).
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 $\text{SEP}(x,C)$.
  - TSP/1-Tree: $\text{SEP}(x, \text{Blossoms})$ Padberg-Rao, $\text{SEP}(s, \text{Blossoms})$ 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 $\text{SEP}(x,C)$.
  - VRP,k-Tree: $\text{SEP}(x, \text{GSECs})$ heuristics, $\text{SEP}(s, \text{GSECs})$ in $O(|E|)$.
  - SS provides an alternative and typically simple heuristic for separation.

- **Case 3:** There exists no algorithm for $\text{SEP}(x,C)$, but ineq's in $C$ are facet-defining.
  - KCCP,CP: $\text{SEP}(x, \text{MaxSet})$ heuristics, $\text{SEP}(s, \text{MaxSet})$ 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 $\text{SEP}(x,C)$.
- TSP/1-Tree: $\text{SEP}(x, \text{Blossoms})$ Padberg-Rao, $\text{SEP}(s, \text{Blossoms})$ 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 $\text{SEP}(x,C)$.
- VRP,k-Tree: $\text{SEP}(x, \text{GSECs})$ heuristics, $\text{SEP}(s, \text{GSECs})$ in $O(|E|)$.
- SS provides an alternative and typically simple heuristic for separation.

**Case 3:** There exists no algorithm for $\text{SEP}(x,C)$, but ineq’s in $C$ are facet-defining.
- KCCP,CP: $\text{SEP}(x, \text{MaxSet})$ heuristics, $\text{SEP}(s, \text{MaxSet})$ 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]
Outline

1. Decomposition Methods
2. Integrated Decomposition Methods
3. Structured Separation
4. DECOMP Framework
DECOMP provides a flexible software framework for testing and extending the theoretical framework presented thus far, with the primary goal of *minimal user responsibility*.

- DECOMP was built around data structures and interfaces provided by COIN-OR:
  
  **COmputational INfrastructure for Operations Research**

- **BCP** provides a framework for parallel PC with *LP-Based Bounding*.
- A generalization of BCP currently under development:
  
  - **ALPs**: Abstract Library for Parallel Search (INFORMS'05 - WC44)
  - **BiCePS**: Branch, Constrain and Price [*Generic Bounding*]
  - **BLIS**: BiCePS Linear Integer Solver = BCP

- DECOMP could provide an implementation of the **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 \(\text{OPT}(c, P')\).
  - 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 \(\text{SEP}(s, C)\)
  - 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 \(\text{SEP}(x, C)\), for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
**DECOMP - Applications**

- 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 \(OPT(c, \mathcal{P}')\).
  - 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.
**DECOMP - Applications**

- 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `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 \(\text{OPT}(c, P')\).  
  - 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 \(\text{SEP}(s, C)\).  
  - 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 \(\text{SEP}(x, C)\), for separation of arbitrary real vectors. Note: the user also has the option to turn on CGL cuts.
DECOMP - Applications

- 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.
An important 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.

This 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.

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.
- Active LP compression, variable and cut pool management.
- Easily switch between relaxations (choice of $\mathcal{P}'$) - just redefine solveRelaxed().
- Flexible parameter interface: command line, parm file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).
DECOMP - TSP Example

class TSP_DecompApp : public DecompApp {
    virtual void createModel(DecompConstraintSet & modelCore,
                              DecompConstraintSet & modelRelax) = 0;

    virtual int solveRelaxed(const double * redCostX,
                              DecompVarList & vars) = 0;

    virtual int generateCuts(const double * x,
                              const DecompVar & var,
                              DecompCutList & newCuts) = 0;

    // 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);

};

DecompApp

TSP_DecompApp

TSP_MatchDecompApp

TSP_OneTreeDecompApp
class TSP_OneTreeDecompApp : public TSP_DecompApp {
    // Define modelCore as 2-degree constraints.
    // Define modelRelax as empty.
    void createModel(DecompConstraintSet & modelCore,
                     DecompConstraintSet & modelRelax);

    // A 1-tree solver.
    int solveRelaxed(const double * redCostX,
                     DecompVarList & vars);

    // Structured separation routines:
    // BFS for blossoms and general combs
    int generateCuts(const double * x,
                     const DecompVar & var,
                     DecompCutList & new_cuts);
    ...
}
TSP_MatchDecompApp

class TSP_MatchDecompApp : public TSP_DecompApp {
   // Define modelCore as 2-degrees constraints (optional).
   // Define modelRelax as empty.
   void createModel(DecompConstraintSet & modelCore,
                     DecompConstraintSet & modelRelax);

   // A 2-matching solver.
   int solveRelaxed(const double * redCostX,
                    DecompVarList & vars);

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

   ...
}
**DECOMP - TSP Example**

**TSP_Main**

```c
int main(int argc, char **argv){
    UtilApp utilApp(argc, argv);
    TSP_MatchDecompApp tspMatch(utilApp);
    TSP_OneTreeDecompApp tspOneTree(utilApp);

    tspMatch.Cut.solve();
    tspMatch.PriceAndCut.solve();
    tspMatch.RelaxAndCut.solve();

    tspOneTree.Cut.solve();
    tspOneTree.PriceAndCut.solve();
    tspOneTree.RelaxAndCut.solve();
}
```

---

**Diagram**

```
DecompAlgo
    /|
   / |
DecompAlgoC  DecompAlgoPC  DecompAlgoRC  DecompAlgoUSER
```

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:

```
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 $\mathcal{P}_O^{t+1}$ and $\mathcal{P}_I^{t+1}$ simultaneously (Vanderbeck).
Summary

- 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.

- We gave some motivation for two new techniques based on SS: revised-PC and DC.
  - The question remains: Empirically, how good are the cuts generated by separation of $s \in \mathcal{D}$?
  - In some cases, this does not matter - we simply don’t know how to separate $x \in \mathbb{R}^n$. These ideas provide a starting point.

- DECOMP provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.

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