A Framework for Decomposition in Integer Programming

Matthew Galati[1] Ted Ralphs[2]

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

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

Column Generation Workshop 2008
Aussois, France

Outline

The Decomposition Principle in Integer Programming

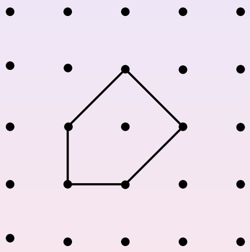
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

$$z_D = \min_{x \in \mathcal{P}'} \{c^\top x \mid A''x \geq b''\}$$

$$z_{IP} \geq z_D \geq z_{LP}$$



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

Assumptions:

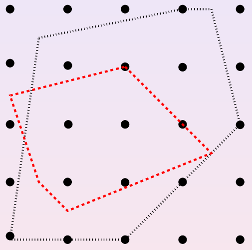
- $OPT(c, \mathcal{P})$ and $SEP(x, \mathcal{P})$ are *"hard"*.
- $OPT(c, \mathcal{P}')$ and $SEP(x, \mathcal{P}')$ are *"easy"*.
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size).
- $\mathcal{P}'$ must be represented implicitly (description has exponential size).

The Decomposition Principle in Integer Programming

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

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

$$z_{IP} \geq z_D \geq z_{LP}$$



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

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

Assumptions:

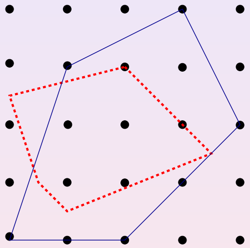
- $OPT(c, \mathcal{P})$ and $SEP(x, \mathcal{P})$ are *"hard"*.
- $OPT(c, \mathcal{P}')$ and $SEP(x, \mathcal{P}')$ are *"easy"*.
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size).
- $\mathcal{P}'$ must be represented implicitly (description has exponential size).

The Decomposition Principle in Integer Programming

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

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

$$z_{IP} \geq z_D \geq z_{LP}$$



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

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

Assumptions:

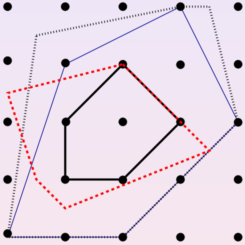
- $OPT(c, \mathcal{P})$ and $SEP(x, \mathcal{P})$ are *"hard"*.
- $OPT(c, \mathcal{P}')$ and $SEP(x, \mathcal{P}')$ are *"easy"*.
- $\mathcal{Q}''$ can be represented explicitly (description has polynomial size).
- $\mathcal{P}'$ must be represented implicitly (description has exponential size).

The Decomposition Principle in Integer Programming

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

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

$$z_{IP} \geq z_D \geq z_{LP}$$



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

Assumptions:

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

The Decomposition Principle in Integer Programming

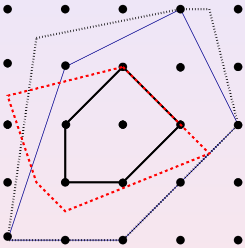
Basic Idea: By leveraging our ability to solve the optimization/separation problem for a relaxation, we can improve the bound yielded by the LP relaxation.

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

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

$$z_D = \min_{x \in \mathcal{P}'} \{c^\top x \mid A''x \geq b''\}$$

$$z_{IP} \geq z_D \geq z_{LP}$$



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

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

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

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

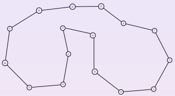
Assumptions:

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

Example - Traveling Salesman Problem

Classical Formulation

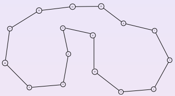
$$
\begin{array}{lll}
x(\delta(\{u\})) & = & 2 \quad\quad \forall u \in V \\
x(E(S)) & \leq & |S| - 1 \quad \forall S \subset V,\ 3 \leq |S| \leq |V| - 1 \\
x_e \in \{0,1\} & & \quad\quad\quad \forall e \in E
\end{array}
$$



Example - Traveling Salesman Problem

Classical Formulation

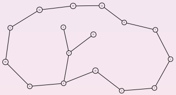
$$\begin{array}{lcll} x(\delta(\{u\})) & = & 2 & \forall u \in V \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V,\ 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Two Relaxations

1-Tree

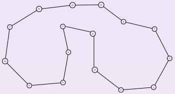
$$\begin{array}{lcll} x(\delta(\{0\})) & = & 2 & \\ x(E(V \setminus \{0\})) & = & |V|-2 & \\ x(E(S)) & \leq & |S|-1 & \forall S \subset V \setminus \{0\}, 3 \leq |S| \leq |V|-1 \\ x_e \in \{0,1\} & & & \forall e \in E \end{array}$$



Example - Traveling Salesman Problem

Classical Formulation

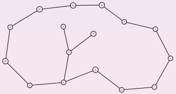
$$
\begin{array}{lcll}
x(\delta(\{u\})) & = & 2 & \forall u \in V \\
x(E(S)) & \leq & |S| - 1 & \forall S \subset V,\ 3 \leq |S| \leq |V| - 1 \\
x_e \in \{0,1\} & & & \forall e \in E
\end{array}
$$



Two Relaxations

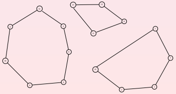
1-Tree

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



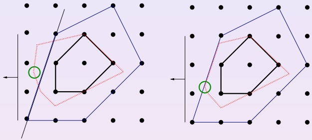
2-Matching

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



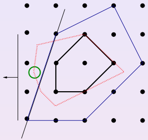
Traditional Decomposition Methods

The **Cutting Plane Method (CP)** iteratively builds an *outer* approximation of $\mathcal{P}'$ by solving a cutting plane generation subproblem.

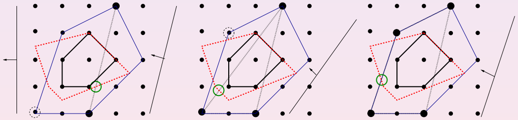


Traditional Decomposition Methods

The **Cutting Plane Method (CP)** iteratively builds an *outer* approximation of $\mathcal{P}'$ by solving a cutting plane generation subproblem.

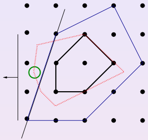


The **Dantzig-Wolfe Method (DW)** iteratively builds an *inner* approximation of $\mathcal{P}'$ by solving a column generation subproblem.

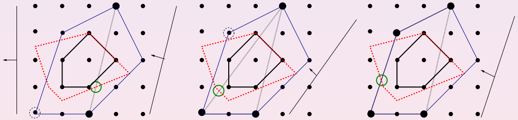


Traditional Decomposition Methods

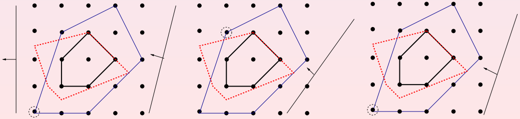
The **Cutting Plane Method (CP)** iteratively builds an *outer* approximation of $\mathcal{P}'$ by solving a cutting plane generation subproblem.



The **Dantzig-Wolfe Method (DW)** iteratively builds an *inner* approximation of $\mathcal{P}'$ by solving a column generation subproblem.



The **Lagrangian Method (LD)** iteratively solves a Lagrangian relaxation subproblem.



Common Threads

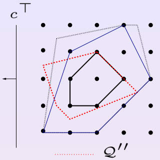
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

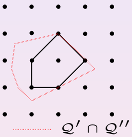
$$z_{LP} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$

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

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

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





Common Threads

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

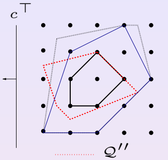
$$z_{LP} = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{Q}' \cap \mathcal{Q}''\}$$

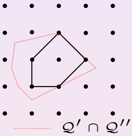
- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

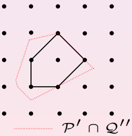
$$z_{CP} = z_{DW} = z_{LD} = z_D = \min_{x \in \mathbb{R}^n} \{c^\top x \mid x \in \mathcal{P}' \cap \mathcal{Q}''\} \geq z_{LP}$$

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

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







Common Threads

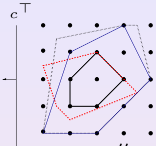
- The LP bound is obtained by optimizing over the intersection of two explicitly defined polyhedra.

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

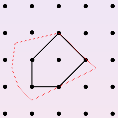
- The decomposition bound is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

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

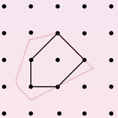
- Traditional decomposition-based bounding methods contain two primary steps
  - **Master Problem:** Update the primal/dual **solution** information.
  - **Subproblem:** Update the **approximation** of $\mathcal{P}'$: $SEP(x, \mathcal{P}')$ or $OPT(c, \mathcal{P}')$.
- Integrated decomposition methods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



$\mathcal{Q}''$



$\mathcal{Q}' \cap \mathcal{Q}''$



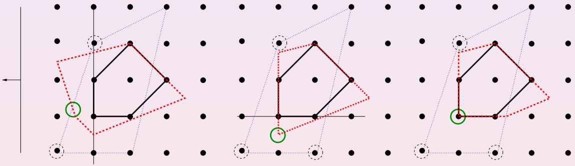
$\mathcal{P}' \cap \mathcal{Q}''$

Price and Cut

Price and Cut: Use **DW** as the bounding method. If we let $\mathcal{F}' = \mathcal{P}' \cap \mathbb{Z}^n$, then

$$z_{DW} = \min_{\lambda \in \mathbb{R}_+^{\mathcal{F}'}} \{c^\top(\sum_{s \in \mathcal{F}'} s\lambda_s) : A''(\sum_{s \in \mathcal{F}'} s\lambda_s) \geq b'', \sum_{s \in \mathcal{F}'} \lambda_s = 1\}$$

- As in the cutting plane method, separate $\hat{x} = \sum_{s \in \mathcal{F}'} s\hat{\lambda}_s$ from $\mathcal{P}$ and add cuts to $[A'', b'']$.
- **Advantage**: Cut generation takes place in the space of the compact formulation (the original space), maintaining the structure of the column generation subproblem.

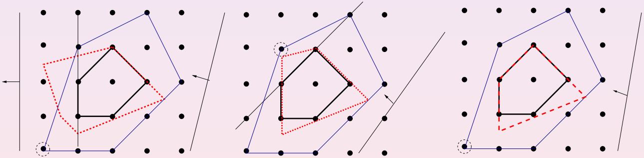


Relax and Cut

Relax and Cut: Use **LD** as the bounding method.

$$z_{LD} = \max_{u \in \mathbb{R}^n_+} \min_{s \in \mathcal{F}'} \{(c^\top - u^\top A'')s + u^\top b''\}$$

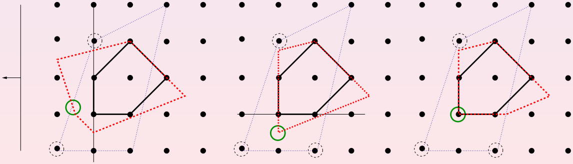
- In each iteration, separate $\hat{s} \in \operatorname{argmin}_{s \in \mathcal{F}'} \{(c^\top - u^\top A'')s + u^\top b''\}$, a solution to the Lagrangian relaxation.
- **Advantage**: It is often much easier to separate a member of $\mathcal{F}'$ from $\mathcal{P}$ than an arbitrary real vector, such as $\hat{x}$.



Decompose and Cut

Decompose and Cut: As in price and cut, use **DW** as the bounding method, but use the decomposition obtained in each iteration to generate improving inequalities as in **RC**.

- Rather than (or in addition to) separating $\hat{x}$, separate each member of $D = \{s \in \mathcal{F}' \mid \hat{\lambda}_s > 0\}$.
- As with **RC**, it is often much easier to separate a member of $\mathcal{F}'$ from $\mathcal{P}$ than an arbitrary real vector, such as $\hat{x}$.
- **RC** only gives us one member of $\mathcal{F}'$ to separate, while **PC** gives us a set, one of which must be violated by any inequality violated by $\hat{x}$.
- We can also use CP and decompose the fractional solution obtained in each iteration into a convex combination of members of $\mathcal{F}'$ and apply the same technique.
- In case this decomposition fails, we still get a Farkas cut for free.

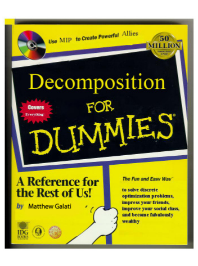


DECOMP Framework: Motivation

DECOMP Framework

DECOMP is a software framework that provides a virtual sandbox for testing and comparing various decomposition-based bounding methods.

- It's very difficult to compare the variants discussed here in a controlled way.
- The method for separation/optimization over $\mathcal{P}'$ is the primary application-dependent component of any of these algorithms.
- DECOMP abstracts the common, generic elements of these methods.
  - Key: The user defines application-specific components in the space of the compact formulation.
  - The framework takes care of reformulation and implementation for all variants described here.

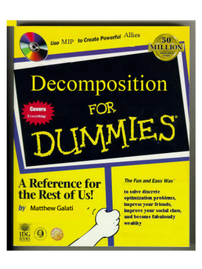


DECOMP Framework: Motivation

DECOMP Framework

DECOMP is a software framework that provides a virtual sandbox for testing and comparing various decomposition-based bounding methods.

- It's very difficult to compare the variants discussed here in a controlled way.
- The method for separation/optimization over $\mathcal{P}'$ is the primary application-dependent component of any of these algorithms.
- DECOMP abstracts the common, generic elements of these methods.
  - **Key:** The user defines application-specific components in the space of the compact formulation.
  - The framework takes care of reformulation and implementation for all variants described here.

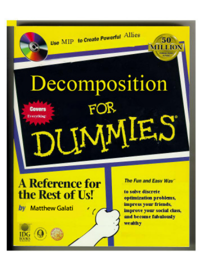


DECOMP Framework: Motivation

DECOMP Framework

DECOMP is a software framework that provides a virtual sandbox for testing and comparing various decomposition-based bounding methods.

- It's very difficult to compare the variants discussed here in a controlled way.
- The method for separation/optimization over $\mathcal{P}'$ is the primary application-dependent component of any of these algorithms.
- DECOMP abstracts the common, generic elements of these methods.
  - **Key:** The user defines application-specific components in the space of the compact formulation.
  - The framework takes care of reformulation and implementation for all variants described here.



DECOMP Framework: Implementation

COmputational INfrastructure for Operations Research

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

DECOMP Framework: Implementation

COmputational INfrastructure for Operations Research

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

DECOMP Framework: Implementation

COmputational INfrastructure for Operations Research

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

DECOMP Features

- One interface to all default algorithms: CP/DC, DW, LD, PC, RC.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the OSI interface, so easy to swap solvers (simplex to interior point).
- Can utilize CGL cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the OSI interface, so easy to swap solvers (simplex to interior point).
- Can utilize CGL cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the OSI interface, so easy to swap solvers (simplex to interior point).
- Can utilize CGL cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP Features

- One interface to all default algorithms: **CP/DC, DW, LD, PC, RC**.
- Automatic reformulation allows users to deal with variables and constraints in the original space.
- Built on top of the **OSI** interface, so easy to swap solvers (simplex to interior point).
- Can utilize **CGL** cuts in all algorithms (since cut generation is always done in the original space).
- Column generation based on *multiple algorithms* can be easily defined and employed.
- Can derive bounds based on *multiple model/algorithm* combinations.
- Provides default (naive) branching rules in the original space.
- Active LP compression, variable and cut pool management.
- Flexible parameter interface: command line, param file, direct call overrides.
- Visualization tools for graph problems (linked to graphviz).

DECOMP - Applications

- The base class `DecompApp` provides an interface for the user to define the application-specific components of their algorithm.
- In order to develop an application, the user must derive the following methods/objects.

- `DecompApp::APPcreateModel()`. Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP 1-Tree: $[A'', b'']$ define the 2-matching constraints.
  - TSP 2-Match: $[A'', b'']$ define trivial subtour constraints.
- `DecompApp::isUserFeasible()`. Does x^* define a feasible solution?
  - TSP: do we have a feasible tour?
- `DecompApp::APPsolveRelaxed()`. 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.

- All other methods have appropriate defaults but are `virtual` and may be overridden.
  - `DecompApp::APPheuristics()`
  - `DecompApp::generateInitVars()`
  - `DecompApp::generateCuts()`
  - ...

DECOMP - Applications

- The base class `DecompApp` provides an interface for the user to define the application-specific components of their algorithm.
- In order to develop an application, the user must derive the following methods/objects.

- `DecompApp::APPcreateModel()`. Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP 1-Tree: $[A'', b'']$ define the 2-matching constraints.
  - TSP 2-Match: $[A'', b'']$ define trivial subtour constraints.
- `DecompApp::isUserFeasible()`. Does x^* define a feasible solution?
  - TSP: do we have a feasible tour?
- `DecompApp::APPsolveRelaxed()`. 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.

- All other methods have appropriate defaults but are `virtual` and may be overridden.
  - `DecompApp::APPheuristics()`
  - `DecompApp::generateInitVars()`
  - `DecompApp::generateCuts()`
  - ...

DECOMP - Applications

- The base class `DecompApp` provides an interface for the user to define the application-specific components of their algorithm.
- In order to develop an application, the user must derive the following methods/objects.

- `DecompApp::APPcreateModel()`. Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP 1-Tree: $[A'', b'']$ define the 2-matching constraints.
  - TSP 2-Match: $[A'', b'']$ define trivial subtour constraints.
- `DecompApp::isUserFeasible()`. Does x^* define a feasible solution?
  - TSP: do we have a feasible tour?
- `DecompApp::APPsolveRelaxed()`. 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.

- All other methods have appropriate defaults but are `virtual` and may be overridden.
  - `DecompApp::APPheuristics()`
  - `DecompApp::generateInitVars()`
  - `DecompApp::generateCuts()`
  - ...

DECOMP - Applications

- The base class `DecompApp` provides an interface for the user to define the application-specific components of their algorithm.
- In order to develop an application, the user must derive the following methods/objects.

- `DecompApp::APPcreateModel()`. Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP 1-Tree: $[A'', b'']$ define the 2-matching constraints.
  - TSP 2-Match: $[A'', b'']$ define trivial subtour constraints.
- `DecompApp::isUserFeasible()`. Does x^* define a feasible solution?
  - TSP: do we have a feasible tour?
- `DecompApp::APPsolveRelaxed()`. 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.

- All other methods have appropriate defaults but are `virtual` and may be overridden.
  - `DecompApp::APPheuristics()`
  - `DecompApp::generateInitVars()`
  - `DecompApp::generateCuts()`
  - ...

DECOMP - Applications

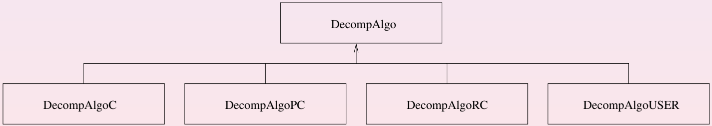
- The base class `DecompApp` provides an interface for the user to define the application-specific components of their algorithm.
- In order to develop an application, the user must derive the following methods/objects.

- `DecompApp::APPcreateModel()`. Define $[A'', b'']$ and $[A', b']$ (optional).
  - TSP 1-Tree: $[A'', b'']$ define the 2-matching constraints.
  - TSP 2-Match: $[A'', b'']$ define trivial subtour constraints.
- `DecompApp::isUserFeasible()`. Does x^* define a feasible solution?
  - TSP: do we have a feasible tour?
- `DecompApp::APPsolveRelaxed()`. 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.

- All other methods have appropriate defaults but are `virtual` and may be overridden.
  - `DecompApp::APPheuristics()`
  - `DecompApp::generateInitVars()`
  - `DecompApp::generateCuts()`
  - ...

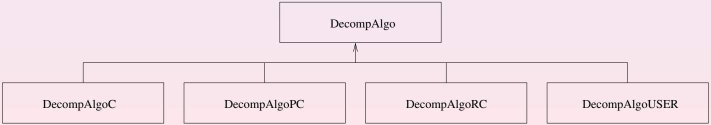
DECOMP - Algorithms

- 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**.
  - Add stabilization to the dual updates in LD, as in **bundle methods**.
  - For LD, replace subgradient with **Volume**, providing an approximate primal solution.
  - Hybrid methods like using LD to initialize the columns of the DW master.
  - During PC, adding cuts to both inner and outer approximations. simultaneously (Vanderbeck).
  - ...



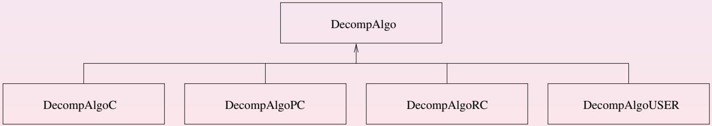
DECOMP - Algorithms

- 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**.
  - Add stabilization to the dual updates in LD, as in **bundle methods**.
  - For LD, replace subgradient with **Volume**, providing an approximate primal solution.
  - Hybrid methods like using LD to initialize the columns of the DW master.
  - During PC, adding cuts to both inner and outer approximations. simultaneously (Vanderbeck).
  - ...



DECOMP - Algorithms

- 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**.
    - Add stabilization to the dual updates in LD, as in **bundle methods**.
    - For LD, replace subgradient with **Volume**, providing an approximate primal solution.
    - Hybrid methods like using LD to initialize the columns of the DW master.
    - During PC, adding cuts to both inner and outer approximations. simultaneously (Vanderbeck).
    - ...



DECOMP - TSP Example

TSP_Main

```
int main(int argc, char ** argv){
   //create the utility class for parsing parameters
   UtilParameters utilParam(argc, argv);

   //create the user application (a DecompApp)
   TSP_DecompApp tsp(utilParam);
   tsp.createModel();

   //create the algorithm(s) (a DecompAlgo)
   DecompAlgoC  * cut       = new DecompAlgoC(&tsp, &utilParam);
   DecompAlgoPC * pcOneTree  = new DecompAlgoPC(&tsp, &utilParam,
                                  TSP_DecompApp::MODEL_ONETREE);
   DecompAlgoPC * pcTwoMatch = new DecompAlgoPC(&tsp, &utilParam,
                                  TSP_DecompApp::MODEL_TWOMATCH);
   DecompAlgoRC * rcOneTree  = new DecompAlgoRC(&tsp, &utilParam,
                                  TSP_DecompApp::MODEL_ONETREE);
   DecompAlgoRC * rcTwoMatch = new DecompAlgoRC(&tsp, &utilParam,
                                  TSP_DecompApp::MODEL_TWOMATCH);

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

   //install the algorithms
   //alpsModel.addDecompAlgo(cut);
   alpsModel.addDecompAlgo(pcOneTree);

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

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.
- **DECOMP** provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The user only needs to define the components based on the compact formulation (irrespective of algorithm).
- 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.
- Related publications:
  - T. Ralphs and M.G., *Decomposition and Dynamic Cut Generation in Integer Programming*, Mathematical Programming 106 (2006), 261
  - T. Ralphs and M.G., *Decomposition in Integer Programming*, in Integer Programming: Theory and Practice, John Karlof, ed. (2005), 57

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.
- **DECOMP** provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The user only needs to define the components based on the compact formulation (irrespective of algorithm).
- 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.
- Related publications:
  - T. Ralphs and M.G., *Decomposition and Dynamic Cut Generation in Integer Programming*, Mathematical Programming 106 (2006), 261
  - T. Ralphs and M.G., *Decomposition in Integer Programming*, in Integer Programming: Theory and Practice, John Karlof, ed. (2005), 57

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.
- **DECOMP** provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The user only needs to define the components based on the compact formulation (irrespective of algorithm).
- 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.
- Related publications:
  - T. Ralphs and M.G., *Decomposition and Dynamic Cut Generation in Integer Programming*, Mathematical Programming 106 (2006), 261
  - T. Ralphs and M.G., *Decomposition in Integer Programming*, in Integer Programming: Theory and Practice, John Karlof, ed. (2005), 57

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.
- **DECOMP** provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The user only needs to define the components based on the compact formulation (irrespective of algorithm).
- 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**.
- Related publications:
  - T. Ralphs and M.G., *Decomposition and Dynamic Cut Generation in Integer Programming*, Mathematical Programming 106 (2006), 261
  - T. Ralphs and M.G., *Decomposition in Integer Programming*, in Integer Programming: Theory and Practice, John Karlof, ed. (2005), 57

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.
- **DECOMP** provides an easy-to-use framework for comparing and developing various decomposition-based bounding methods.
  - The user only needs to define the components based on the compact formulation (irrespective of algorithm).
- 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**.
- Related publications:
  - T. Ralphs and M.G., *Decomposition and Dynamic Cut Generation in Integer Programming*, Mathematical Programming 106 (2006), 261
  - T. Ralphs and M.G., *Decomposition in Integer Programming*, in Integer Programming: Theory and Practice, John Karlof, ed. (2005), 57