A Framework for Decomposition in Integer Programming

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

\textsuperscript{1}SAS Institute, Advanced Analytics, Operations Research R & D
\textsuperscript{2}COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University

Column Generation Workshop 2008
Aussois, France
Outline

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

\[
\begin{align*}
    z_{IP} &= \min_{x \in \mathbb{Z}^n} \{ c^T x \mid A'x \geq b', A''x \geq b'' \} \\
    z_{LP} &= \min_{x \in \mathbb{R}^n} \{ c^T x \mid A'x \geq b', A''x \geq b'' \} \\
    z_D &= \min_{x \in P'} \{ c^T x \mid A''x \geq b'' \}
\end{align*}
\]

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

**Assumptions:**
- \( OPT(c, \mathcal{P}) \) 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).

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

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

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

**Assumptions:**
- \( OPT(c, P) \) 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).
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.

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

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

**Assumptions:**
- \(OPT(c, P)\) 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).
**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.

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

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

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

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

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

**Assumptions:**
- \(\text{OPT}(c, \mathcal{P})\) and \(\text{SEP}(x, \mathcal{P})\) are “hard”.
- \(\text{OPT}(c, \mathcal{P}')\) and \(\text{SEP}(x, \mathcal{P}')\) 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

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

Classical Formulation

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

Two Relaxations

1-Tree

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

### Classical Formulation

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

### Two Relaxations

#### 1-Tree

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

#### 2-Matching

\[ x(\delta(u)) = 2 \quad \forall u \in V \]
\[ x_e \in \{0, 1\} \quad \forall e \in E \]
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 $P'$ by solving a cutting plane generation subproblem.

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

The **Lagrangian Method (LD)** iteratively solves a Lagrangian relaxation subproblem.
The *LP bound* is obtained by optimizing over the intersection of two explicitly defined polyhedra.

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

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

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

Traditional decomposition-based bounding methods contain two primary steps

- **Master Problem**: Update the primal/dual solution information.
- **Subproblem**: Update the approximation of \( P' \): \( SEP(x, \mathcal{P}') \) 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)
The **LP bound** is obtained by optimizing over the intersection of two explicitly defined polyhedra.

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

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

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

Traditional decomposition-based bounding methods contain two primary steps

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

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

- The **decomposition bound** is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.
  \[ z_{CP} = z_{DW} = z_{LD} = z_{D} = \min_{x \in \mathbb{R}^n} \{ c^\top x \mid x \in P' \cap Q'' \} \geq z_{LP} \]

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

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

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

- As in the cutting plane method, separate $\hat{x} = \sum_{s \in F'} s \hat{\lambda}_s$ from $P$ 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: 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 \( \hat{s} \in \arg\min_{s \in \mathcal{F}'} \{(c^\top - u^\top A''') s + u^\top b''\} \), 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 $D = \{ s \in F' \mid \hat{\lambda}_s > 0 \}$.
- As with RC, it is often much easier to separate a member of $F'$ from $P$ than an arbitrary real vector, such as $\hat{x}$.
- RC only gives us one member of $F'$ 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 $F'$ and apply the same technique.
- In case this decomposition fails, we still get a Farkas cut for free.
DECOMP Framework: Motivation

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 $P'$ 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 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 $P'$ 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 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 $P'$ 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.

Galati, Ralphs  A Framework for Decomposition in IP
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** 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 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/algorithms* 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/algorithms* 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).
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 \(OPT(c, P')\).
  - 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()`
- `...`
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 \(OPT(c, P')\).
  - 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()`
- ...

Galati, Ralphs  A Framework for Decomposition in IP
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()`
- ...
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 $OPT(c, P')$.
  - 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.
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 \(OPT(c, P')\).
  - 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()`
- `...`
The base class `DecompAlgo` provides the shell (init / master / subproblem / update).

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

```java
public DecompAlgo which are accessible by any application class, allowing full flexibility.
```

New, hybrid or extended methods can be easily derived by overriding the various subroutines, which are called from the base class. For example,

- Alternative methods for solving the master LP in DW, such as interior point methods or ACCPM.
- 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).
- ...

![Diagram of DECOMP Algorithms](image-url)
The base class `DecompAlgo` provides the shell (init / master / subproblem / update).

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

```java
public DecompAlgo which are accessible by any application class, allowing full flexibility.
```

New, hybrid or extended methods can be easily derived by overriding the various subroutines, which are called from the base class. For example,

- Alternative methods for solving the master LP in DW, such as interior point methods or ACCPM.
- 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).
- ...

![Diagram of DECOMP Algorithms]

- **DecompAlgoC**
- **DecompAlgoPC**
- **DecompAlgoRC**
- **DecompAlgoUSER**
**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).
  - ...

![Diagram of DECOMP Algorithms]

Galati, Ralphs  A Framework for Decomposition in IP
int main(int argc, char ** argv){
    // create the utility class for parsing parameters
    UtilParameters utilParam(argc, argv);

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

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

- Traditional Decomposition Methods approximate $\mathcal{P}$ as $\mathcal{P}' \cap Q''$.
  - $\mathcal{P}' \supset \mathcal{P}$ 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](http://www.coin-or.org).
- Related publications:
Summary

- Traditional Decomposition Methods approximate $\mathcal{P}$ as $\mathcal{P}' \cap Q''$.
  - $\mathcal{P}' \supset \mathcal{P}$ 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](http://www.coin-or.org).

- Related publications:
## 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](http://www.coin-or.org).

- **Related publications:**
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](http://www.coin-or.org).

- Related publications: