Progress Update on COIN/DIP
Decomposition Methods for Integer Programming

Matthew Galati\textsuperscript{1}  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

INFORMS Annual Meeting 2010
Austin, Texas
Executive Summary

- Most software/theoretical frameworks for decomposition work in some **reformulated space**.
  - Burden on user to derive methods for each new algorithm or choice of decomposition.
- **DIP** works in the **compact space** (typically, the **original** formulation).
  - For many MILP practitioners, this is most familiar and preferable.
  - Opens the door for using decomposition without the learning curve or development burden.
Outline

1. DIP Framework

2. Applications
   - Multi-Choice Multi-Dimensional Knapsack Problem
   - ATM Cash Management Problem
   - Generic Black-box Solver for Block-Angular MILP

3. Other Projects Using DIP
Outline

1. DIP Framework

2. Applications
   - Multi-Choice Multi-Dimensional Knapsack Problem
   - ATM Cash Management Problem
   - Generic Black-box Solver for Block-Angular MILP

3. Other Projects Using DIP
Decomposition Bounding Methods - Common Threads

- **LP bound** – intersection of two explicitly defined polyhedra

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

- **Decomposition bound** – intersection of one explicitly and one implicitly defined polyhedron

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

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

- **Integrated decomposition methods** – intersection of two implicitly defined polyhedra
  - Price-and-Cut (PC)
  - Relax-and-Cut (RC)
  - Decompose-and-Cut (DC)
Decomposition Bounding Methods - Common Threads

- **LP bound** – intersection of two explicitly defined polyhedra

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

- **Decomposition bound** – intersection of one explicitly and one implicitly defined polyhedron

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

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

- Integrated decomposition methods – intersection of two implicitly defined polyhedra
  - Price-and-Cut (PC)
  - Relax-and-Cut (RC)
  - Decompose-and-Cut (DC)
Decomposition Bounding Methods - Common Threads

- **LP bound** – intersection of two explicitly defined polyhedra

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

- **Decomposition bound** – intersection of one explicitly and one implicitly defined polyhedron

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

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

- **Integrated decomposition methods** – intersection of two implicitly defined polyhedra
  - **Price-and-Cut** (PC)
  - **Relax-and-Cut** (RC)
  - **Decompose-and-Cut** (DC)
COmputational INfrastructure for Operations REsearch

Have some DIP with your CHiPPs?

DIP was built around data structures and interfaces provided by COIN-OR

- The DIP framework, written in C++, is accessed through two user interfaces:
  - Applications Interface: DecompApp
  - Algorithms Interface: DecompAlgo

- DIP provides the bounding method for branch and bound
- ALPS (Abstract Library for Parallel Search) provides the framework for tree search
  - AlpsDecompModel : public AlpsModel
    - a wrapper class that calls (data access) methods from DecompApp
  - AlpsDecompTreeNode : public AlpsTreeNode
    - a wrapper class that calls (algorithmic) methods from DecompAlgo
DIP was built around data structures and interfaces provided by COIN-OR

The DIP framework, written in C++, is accessed through two user interfaces:

- **Applications Interface**: DecomApp
- **Algorithms Interface**: DecomAlgo

DIP provides the bounding method for branch and bound

ALPS (Abstract Library for Parallel Search) provides the framework for tree search

- AlpsDecompModel : public AlpsModel
  - a wrapper class that calls (data access) methods from DecomApp

- AlpsDecompTreeNode : public AlpsTreeNode
  - a wrapper class that calls (algorithmic) methods from DecomAlgo
**DIP – Implementation in COIN**

Co**mputational IN**frastructure for **O**perations **R**esearch

*Have some DIP with your CHiPPs?

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

DecompApp

- Interface for user to define **application-specific** components *in compact space*
  
  Define the model(s)
  
  - `setModelObjective(double * c)`: define \( c \)
  
  - `setModelCore(DecompConstraintSet * model)`: define \( Q'' \)
  
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define \( Q' \) [optional]
  
  - `solveRelaxed()`: define a method for \( \text{OPT}(P', c) \) [optional, if \( Q', \text{CBC} \) is built-in]
  
  - `generateCuts()`: define a method for \( \text{SEP}(P', x) \) [optional, \text{CGL} \) is built-in]
  
  - `isUserFeasible()`: is \( \hat{x} \in P \)? [optional, if \( P = \text{conv}(P' \cap Q'' \cap \mathbb{Z}) \)]

- All other methods have appropriate defaults but are virtual and may be overridden

DecompAlgo

- Provides **algorithm shell** (init / master / subproblem / update).

- Each of the methods described has derived default implementations
  
  - `DecompAlgoC/RC/PC` : public `DecompAlgo`

- New, hybrid or extended methods can be easily derived from base object
  
  - e.g., `DecompAlgoDC` : public `DecompAlgoPC`
DIP – Creating a User Application

DecompApp

- Interface for user to define **application-specific** components *in compact space*
- Define the model(s)
  - `setModelObjective(double * c)`: define \( c \)
  - `setModelCore(DecompConstraintSet * model)`: define \( Q'' \)
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define \( Q' \) [optional]
- `solveRelaxed()`: define a method for \( \text{OPT}(P', c) \) [optional, if \( Q', \text{CBC} \) is built-in]
- `generateCuts()`: define a method for \( \text{SEP}(P', x) \) [optional, \( \text{CGL} \) is built-in]
- `isUserFeasible()`: is \( \hat{x} \in P \)? [optional, if \( P = \text{conv}(P' \cap Q'' \cap \mathbb{Z}) \)]
- All other methods have appropriate defaults but are virtual and may be overridden

DecompAlgo

- Provides **algorithm shell** (init / master / subproblem / update).
- Each of the methods described has derived default implementations
  - `DecompAlgoC/RC/PC` : public `DecompAlgo`
- New, hybrid or extended methods can be easily derived from base object
  - e.g., `DecompAlgoDC` : public `DecompAlgoPC`
DIP – Creating a User Application

DecompApp

- Interface for user to define **application-specific** components *in compact space*
- Define the model(s)
  - `setModelObjective(double * c)`: define `c`
  - `setModelCore(DecompConstraintSet * model)`: define `Q''`
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define `Q'` [optional]
- `solveRelaxed()`: define a method for `OPT(P', c)` [optional, if `Q', CBC` is built-in]
- `generateCuts()`: define a method for `SEP(P', x)` [optional, `CGL` is built-in]
- `isUserFeasible()`: is \( \hat{x} \in P \)? [optional, if \( P = \text{conv}(P' \cap Q'' \cap \mathbb{Z}) \)]

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

DecompAlgo

- Provides **algorithm shell** (init / master / subproblem / update).
- Each of the methods described has derived default implementations
  - `DecompAlgoC/RC/PC` : public `DecompAlgo`
- New, hybrid or extended methods can be easily derived from base object
  - e.g., `DecompAlgoDC` : public `DecompAlgoPC`
DecompApp

- Interface for user to define application-specific components in compact space
- Define the model(s)
  - `setModelObjective(double * c)`: define $c$
  - `setModelCore(DecompConstraintSet * model)`: define $Q''$
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define $Q'$ [optional]
- `solveRelaxed()`: define a method for $\text{OPT}(\mathcal{P}', c)$ [optional, if $Q'$, CBC is built-in]
- `generateCuts()`: define a method for $\text{SEP}(\mathcal{P}', x)$ [optional, CGL is built-in]
- `isUserFeasible()`: is $\hat{x} \in \mathcal{P}$? [optional, if $\mathcal{P} = \text{conv}(\mathcal{P}' \cap Q'' \cap \mathbb{Z})$]
- All other methods have appropriate defaults but are virtual and may be overridden

DecompAlgo

- Provides algorithm shell (init / master / subproblem / update).
- Each of the methods described has derived default implementations
  - `DecompAlgoC/RC/PC` : public `DecompAlgo`
- New, hybrid or extended methods can be easily derived from base object
  - e.g., `DecompAlgoDC` : public `DecompAlgoPC`
DIP – Creating a User Application

DecompApp
- Interface for user to define **application-specific** components *in compact space*
- Define the model(s)
  - `setModelObjective(double * c)`: define \( c \)
  - `setModelCore(DecompConstraintSet * model)`: define \( Q'' \)
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define \( Q' \) [optional]
- `solveRelaxed()`: define a method for \( \text{OPT}(P', c) \) [optional, if \( Q', \text{CBC} \) is built-in]
- `generateCuts()`: define a method for \( \text{SEP}(P', x) \) [optional, \( \text{CGL} \) is built-in]
- `isUserFeasible()`: is \( \hat{x} \in P \)? [optional, if \( P = \text{conv}(P' \cap Q'' \cap \mathbb{Z}) \)]
- All other methods have appropriate defaults but are virtual and may be overridden

DecompAlgo
- Provides **algorithm shell** (init / master / subproblem / update).
- Each of the methods described has derived default implementations
  - `DecompAlgoC/RC/PC`: public `DecompAlgo`
- New, hybrid or extended methods can be easily derived from base object
  - e.g., `DecompAlgoDC`: public `DecompAlgoPC`
DIP – Creating a User Application

DecompApp

- Interface for user to define **application-specific** components *in compact space*
- Define the model(s)
  - `setModelObjective(double * c)`: define \( c \)
  - `setModelCore(DecompConstraintSet * model)`: define \( Q'' \)
  - `setModelRelaxed(DecompConstraintSet * model, int block)`: define \( Q' \) [optional]
- `solveRelaxed()`: define a method for \( \text{OPT}(\mathcal{P}', c) \) [optional, if \( Q', \text{CBC} \) is built-in]
- `generateCuts()`: define a method for \( \text{SEP}(\mathcal{P}', x) \) [optional, \text{CGL} is built-in]
- `isUserFeasible()`: is \( \hat{x} \in \mathcal{P} \)? [optional, if \( \mathcal{P} = \text{conv}(\mathcal{P}' \cap Q'' \cap \mathbb{Z}) \)]

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

DecompAlgo

- Provides **algorithm shell** (init / master / subproblem / update).
- Each of the methods described has derived default implementations
  - `DecompAlgoC/RC/PC`: public `DecompAlgo`
- New, hybrid or extended methods can be easily derived from base object
  - e.g., `DecompAlgoDC`: public `DecompAlgoPC`
DIP – Example Main

```c
int main(int argc, char ** argv){
    // create the utility class for parsing parameters
    UtilParameters utilParam (argc, argv);
    bool doCut = utilParam.GetSetting("doCut", true);
    bool doPriceCut = utilParam.GetSetting("doPriceCut", false);
    bool doRelaxCut = utilParam.GetSetting("doRelaxCut", false);

    // create the user application (a DecompApp)
    SILP_DecomppApp sip (utilParam);

    // create the CPM/PC/RC algorithm objects (a DecompAlgo)
    DecompAlgo * algo = NULL;
    if (doCut) algo = new DecompAlgoC (&sip, &utilParam);
    if (doPriceCut) algo = new DecompAlgoPC (&sip, &utilParam);
    if (doRelaxCut) algo = new DecompAlgoRC (&sip, &utilParam);

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

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

- **BCP**: The user must derive methods for almost all of the *algorithmic components*: (master reformulation, expansion of rows and columns, branching in reformulated space, calculation of pricing mechanisms like reduced cost, etc).

- **DIP**: Must exist a *compact formulation* which forms the basis of the model attributes.

Design

- **BCP**: The user defines the model attributes and algorithmic components based on one predefined solution *method* (i.e., PC or CPM).

- **DIP**: The user defines the model attributes and algorithmic components based on one predefined compact *formulation*. Different algorithmic details managed by framework.

Parallelism

- **BCP**: Designed for shared or distributed memory for branch-and-bound search.

- **DIP**: Threaded for block-angular shared memory processing.

- **DIP**: Built on top of ALPS – potential for fully distributed branch-and-bound (*future*).
DIP – Compare and Contrast to COIN/BCP

Limitations

- **BCP**: The user must derive methods for almost all of the algorithmic components: (master reformulation, expansion of rows and columns, branching in reformulated space, calculation of pricing mechanisms like reduced cost, etc).
- **DIP**: Must exist a compact formulation which forms the basis of the model attributes.

Design

- **BCP**: The user defines the model attributes and algorithmic components based on one predefined solution *method* (i.e., PC or CPM).
- **DIP**: The user defines the model attributes and algorithmic components based on one predefined compact *formulation*. Different algorithmic details managed by framework.

Parallelism

- **BCP**: Designed for shared or distributed memory for branch-and-bound search.
- **DIP**: Threaded for block-angular shared memory processing.
- **DIP**: Built on top of ALPS – potential for fully distributed branch-and-bound (*future*).
DIP – Compare and Contrast to COIN/BCP

Limitations

- **BCP**: The user must derive methods for almost all of the algorithmic components: (master reformulation, expansion of rows and columns, branching in reformulated space, calculation of pricing mechanisms like reduced cost, etc).

- **DIP**: Must exist a compact formulation which forms the basis of the model attributes.

Design

- **BCP**: The user defines the model attributes and algorithmic components based on one predefined solution method (i.e., PC or CPM).

- **DIP**: The user defines the model attributes and algorithmic components based on one predefined compact formulation. Different algorithmic details managed by framework.

Parallelism

- **BCP**: Designed for shared or distributed memory for branch-and-bound search.

- **DIP**: Threaded for block-angular shared memory processing.

- **DIP**: Built on top of ALPS – potential for fully distributed branch-and-bound (future).
**Integration of generic MILP cuts**

- Using the mapping \( \hat{x} = \sum_{s \in \mathcal{E}} s\hat{\lambda}_s \) we can use generic MILP cuts in RC/PC context.

**Initial columns**

- Solve \( \text{OPT}(P', c + r) \) for random perturbations.
- Run several iterations of LD or DC collecting extreme points.

**Price-and-branch heuristic**

- For block-angular case, at end of each node, solve with \( \lambda \in \mathbb{Z} \).
- Used in *root node* by Barahona and Jensen (’98), we extend to tree.

**Choice of master LP solver**

- Dual simplex after adding rows or adjusting bounds (warm-start dual feasible).
- Primal simplex after adding columns (warm-start primal feasible).
- Interior-point methods might help with stabilization vs extremal duals.

**Compression of master LP and object pools**

- Reduce size of master LP, improve efficiency of subproblem processing.
Some Algorithmic Considerations for PC/RC

Integration of generic MILP cuts
- Using the mapping \( \hat{x} = \sum_{s \in E} s \hat{\lambda}_s \) we can use generic MILP cuts in RC/PC context

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

Price-and-branch heuristic
- For block-angular case, at end of each node, solve with \( \lambda \in \mathbb{Z} \)
- Used in root node by Barahona and Jensen ('98), we extend to tree

Choice of master LP solver
- Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
- Primal simplex after adding columns (warm-start primal feasible)
- Interior-point methods might help with stabilization vs extremal duals

Compression of master LP and object pools
- Reduce size of master LP, improve efficiency of subproblem processing
Some Algorithmic Considerations for PC/RC

Integration of generic MILP cuts
- Using the mapping $\hat{x} = \sum_{s \in E} s \hat{\lambda}_s$ we can use generic MILP cuts in RC/PC context

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

Price-and-branch heuristic
- For block-angular case, at end of each node, solve with $\lambda \in \mathbb{Z}$
- Used in root node by Barahona and Jensen ('98), we extend to tree

Choice of master LP solver
- Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
- Primal simplex after adding columns (warm-start primal feasible)
- Interior-point methods might help with stabilization vs extremal duals

Compression of master LP and object pools
- Reduce size of master LP, improve efficiency of subproblem processing
Some Algorithmic Considerations for PC/RC

Integration of generic MILP cuts
- Using the mapping \( \hat{x} = \sum_{s \in \mathcal{E}} s \hat{\lambda}_s \) we can use generic MILP cuts in RC/PC context

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

Price-and-branch heuristic
- For block-angular case, at end of each node, solve with \( \lambda \in \mathbb{Z} \)
- Used in root node by Barahona and Jensen ('98), we extend to tree

Choice of master LP solver
- Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
- Primal simplex after adding columns (warm-start primal feasible)
- Interior-point methods might help with stabilization vs extremal duals

Compression of master LP and object pools
- Reduce size of master LP, improve efficiency of subproblem processing
Some Algorithmic Considerations for PC/RC

Integration of generic MILP cuts
- Using the mapping \( \hat{x} = \sum_{s \in E} s \hat{\lambda}_s \) we can use generic MILP cuts in RC/PC context

Initial columns
- Solve \( \text{OPT}(P', c + r) \) for random perturbations
- Run several iterations of LD or DC collecting extreme points

Price-and-branch heuristic
- For block-angular case, at end of each node, solve with \( \lambda \in \mathbb{Z} \)
- Used in root node by Barahona and Jensen (’98), we extend to tree

Choice of master LP solver
- Dual simplex after adding rows or adjusting bounds (warm-start dual feasible)
- Primal simplex after adding columns (warm-start primal feasible)
- Interior-point methods might help with stabilization vs extremal duals

Compression of master LP and object pools
- Reduce size of master LP, improve efficiency of subproblem processing
Some Algorithmic Considerations for PC/RC (Cont.)

Branching for Inner Methods (PC)

- Add column bounds to \([A''', b''']\) and map back to the compact space \(\hat{x} = \sum_{s \in E} s \hat{\lambda}_s\)
- Variable branching in the compact space is constraint branching in the extended space
- **Current Limitation:** Identical subproblems are currently treated like non-identical

Branching for Inner Methods (RC)

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

Nested Polyhedra

- Outer methods use various approximations to improve the bound (template paradigm)
- Idea: generate inner points from multiple (nested) polyhedra
- For any polyhedron \(\mathcal{P}'_N \subset \mathcal{P}'\), we can also heuristically solve \(\text{OPT}(\mathcal{P}'_N, c^\top - u^\top A''')\)
- The more diverse the pool of columns, the better the chance to find good incumbents
Some Algorithmic Considerations for PC/RC (Cont.)

Branching for Inner Methods (PC)
- Add column bounds to \([A'', b'']\) and map back to the compact space \(\hat{x} = \sum_{s \in E} s\hat{\lambda}_s\)
- Variable branching in the compact space is constraint branching in the extended space
- **Current Limitation:** Identical subproblems are currently treated like non-identical

Branching for Inner Methods (RC)
- In general, Lagrangian methods do *not* provide a primal solution \(\lambda\)
- Let \(B\) define the extreme points found in solving subproblems for \(z_{LD}\)
- Build an inner approximation using \(B\), then proceed as PC (related to *bundle methods*)

Nested Polyhedra
- Outer methods use various approximations to improve the bound (template paradigm)
- Idea: generate inner points from multiple (nested) polyhedra
- For any polyhedron \(\mathcal{P}_N' \subset \mathcal{P}'\), we can also heuristically solve \(\text{OPT} (\mathcal{P}_N', c^\top - u^\top A'')\)
- The more diverse the pool of columns, the better the chance to find good incumbents
Some Algorithmic Considerations for PC/RC (Cont.)

Branching for Inner Methods (PC)
- Add column bounds to $[A'', b'']$ and map back to the compact space $\hat{x} = \sum_{s \in E} s \hat{\lambda}_s$
- Variable branching in the compact space is constraint branching in the extended space
- **Current Limitation:** Identical subproblems are currently treated like non-identical

Branching for Inner Methods (RC)
- In general, Lagrangian methods do not provide a primal solution $\lambda$
- Let $B$ define the extreme points found in solving subproblems for $z_{LD}$
- Build an inner approximation using $B$, then proceed as PC (related to *bundle methods*)

Nested Polyhedra
- Outer methods use various approximations to improve the bound (template paradigm)
  - **Idea:** generate inner points from multiple (nested) polyhedra
- For any polyhedron $P'_N \subset P'$, we can also *heuristically* solve $\text{OPT} \left( P'_N, c^\top - u^\top A'' \right)$
- The more *diverse* the pool of columns, the better the chance to find good incumbents
Outline

1 DIP Framework

2 Applications
   - Multi-Choice Multi-Dimensional Knapsack Problem
   - ATM Cash Management Problem
   - Generic Black-box Solver for Block-Angular MILP

3 Other Projects Using DIP
### DIP – Example Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>( P' )</th>
<th>OPT(c)</th>
<th>SEP(x)</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP3</td>
<td>3-index assignment</td>
<td>AP</td>
<td>Jonker</td>
<td>user</td>
<td>user</td>
</tr>
<tr>
<td>ATM</td>
<td>cash management (SAS COE)</td>
<td>MILP(s)</td>
<td>CBC</td>
<td>CGL</td>
<td>user</td>
</tr>
<tr>
<td>GAP</td>
<td>generalized assignment</td>
<td>KP(s)</td>
<td>Pisinger</td>
<td>CGL</td>
<td>user</td>
</tr>
<tr>
<td>MAD</td>
<td>matrix decomposition</td>
<td>MaxClique</td>
<td>Cliquer</td>
<td>CGL</td>
<td>user</td>
</tr>
<tr>
<td>MILP</td>
<td>random partition into ( A', A'' )</td>
<td>MILP</td>
<td>CBC</td>
<td>CGL</td>
<td>mps, block</td>
</tr>
<tr>
<td>MILPBlock</td>
<td>user-defined blocks for ( A' )</td>
<td>MILP(s)</td>
<td>CBC</td>
<td>CGL</td>
<td>mps, block</td>
</tr>
<tr>
<td>MMKP</td>
<td>multi-dim/choice knapsack</td>
<td>MCKP</td>
<td>Pisinger</td>
<td>CGL</td>
<td>user</td>
</tr>
<tr>
<td>SILP</td>
<td>intro example, tiny IP</td>
<td>MILP</td>
<td>CBC</td>
<td>CGL</td>
<td>user</td>
</tr>
<tr>
<td>TSP</td>
<td>traveling salesman problem</td>
<td>1-Tree</td>
<td>Boost</td>
<td>Concorde</td>
<td>user</td>
</tr>
<tr>
<td>VRP</td>
<td>vehicle routing problem</td>
<td>( k )-TSP</td>
<td>Concorde</td>
<td>CVRPSEP</td>
<td>user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( b )-Match</td>
<td>CBC</td>
<td>CVRPSEP</td>
<td>user</td>
</tr>
</tbody>
</table>
Multi-Choice Multi-Dimensional Knapsack Problem (MMKP)

SAS Marketing Optimization – improve ROI for marketing campaign offers by targeting higher response rates, improving channel effectiveness, and reduce spending.

\[
\begin{align*}
\text{max} & \quad \sum_{i \in N} \sum_{j \in L_i} v_{ij} x_{ij} \\
& \quad \sum_{i \in N} \sum_{j \in L_i} r_{kij} x_{ij} \leq b_k \quad \forall k \in M \\
& \quad \sum_{j \in L_i} x_{ij} = 1 \quad \forall i \in N \\
x_{ij} & \quad \in \{0, 1\} \quad \forall i \in N, j \in L_i
\end{align*}
\]

- Relaxation - Multi-Choice Knapsack Problem (MCKP)
  - solver \textit{mcknap} by Pisinger a DP-based branch-and-bound
Multi-Choice Multi-Dimensional Knapsack Problem (MMKP)

SAS Marketing Optimization – improve ROI for marketing campaign offers by targeting higher response rates, improving channel effectiveness, and reduce spending.

\[
\begin{align*}
\text{max} & \quad \sum_{i \in N} \sum_{j \in L_i} v_{ij} x_{ij} \\
& \quad \sum_{i \in N} \sum_{j \in L_i} r_{kij} x_{ij} \leq b_k \quad \forall k \in M \\
& \quad \sum_{j \in L_i} x_{ij} = 1 \quad \forall i \in N \\
& \quad x_{ij} \in \{0, 1\} \quad \forall i \in N, j \in L_i
\end{align*}
\]

- Relaxation - Multi-Choice Knapsack Problem (MCKP)

  - solver \textit{mcknap} by Pisinger a DP-based branch-and-bound

\[
\begin{align*}
& \quad \sum_{i \in N} \sum_{j \in L_i} r_{mij} x_{ij} \leq b_m \\
& \quad \sum_{j \in L_i} x_{ij} = 1 \quad \forall i \in N \\
& \quad x_{ij} \in \{0, 1\} \quad \forall i \in N, j \in L_i
\end{align*}
\]
Nested Relaxations

- Multi-Choice 2-D Knapsack Problem (MC2KP): $\mathcal{P}_p^{MC2KP} \subset \mathcal{P}_{MCKP} \forall p \in M \setminus \{m\}$

\[
\sum_{i \in N} \sum_{j \in L_i} r_{pij} x_{ij} \leq b_p \\
\sum_{i \in N} \sum_{j \in L_i} r_{mij} x_{ij} \leq b_m \\
\sum_{j \in L_i} x_{ij} = 1 \quad \forall i \in N \\
x_{ij} \in \{0, 1\} \quad \forall i \in N, j \in L_i
\]
Nested Relaxations

- Multi-Choice 2-D Knapsack Problem (MC2KP): \( P_{\text{MC2KP}} \subset P_{\text{MCKP}} \forall p \in M \setminus \{m\} \)

\[
\begin{align*}
\sum_{i \in N} \sum_{j \in L_i} r_{pij} x_{ij} & \leq b_p \\
\sum_{i \in N} \sum_{j \in L_i} r_{mij} x_{ij} & \leq b_m \\
\sum_{j \in L_i} x_{ij} & = 1 \quad \forall i \in N \\
x_{ij} & \in \{0, 1\} \quad \forall i \in N, j \in L_i
\end{align*}
\]

- Multi-Choice Multi-Dimensional Knapsack Problem (MMKP): \( P \subset P_{\text{MCKP}} \)
### MMKP: PC vs PC Nested with MC2KP and MMKP

<table>
<thead>
<tr>
<th>Instance</th>
<th>DIP-PC</th>
<th>DIP-PC-M2</th>
<th>DIP-PC-MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>0.04 OPT</td>
<td>0.16 OPT</td>
<td>0.08 OPT</td>
</tr>
<tr>
<td>I10</td>
<td>T 11.86%</td>
<td>T 6.99%</td>
<td>T 0.63%</td>
</tr>
<tr>
<td>I11</td>
<td>T 12.25%</td>
<td>T 11.15%</td>
<td>T 0.60%</td>
</tr>
<tr>
<td>I12</td>
<td>T 7.93%</td>
<td>T 11.41%</td>
<td>T 0.79%</td>
</tr>
<tr>
<td>I13</td>
<td>T 11.89%</td>
<td>T 13.65%</td>
<td>T 0.52%</td>
</tr>
<tr>
<td>I2</td>
<td>0.05 OPT</td>
<td>0.45 OPT</td>
<td>0.14 OPT</td>
</tr>
<tr>
<td>I3</td>
<td>T 1.07%</td>
<td>T 1.18%</td>
<td>T 1.10%</td>
</tr>
<tr>
<td>I4</td>
<td>T 5.14%</td>
<td>T 3.18%</td>
<td>T 1.23%</td>
</tr>
<tr>
<td>I5</td>
<td>0.13 OPT</td>
<td>0.14 OPT</td>
<td>0.07 OPT</td>
</tr>
<tr>
<td>I6</td>
<td>T 0.28%</td>
<td>483.53 OPT</td>
<td>T 0.25%</td>
</tr>
<tr>
<td>I7</td>
<td>T 14.32%</td>
<td>T 4.85%</td>
<td>T 0.97%</td>
</tr>
<tr>
<td>I8</td>
<td>T 13.36%</td>
<td>T 9.79%</td>
<td>T 0.67%</td>
</tr>
<tr>
<td>I9</td>
<td>T 10.71%</td>
<td>T 10.57%</td>
<td>T 0.73%</td>
</tr>
<tr>
<td>INST01</td>
<td>T 9.99%</td>
<td>T 5.97%</td>
<td>T 1.86%</td>
</tr>
<tr>
<td>INST02</td>
<td>T 7.39%</td>
<td>T 7.29%</td>
<td>T 1.74%</td>
</tr>
<tr>
<td>INST03</td>
<td>T 3.83%</td>
<td>T 11.93%</td>
<td>T 1.61%</td>
</tr>
<tr>
<td>INST04</td>
<td>T 7.48%</td>
<td>T 7.04%</td>
<td>T 1.56%</td>
</tr>
<tr>
<td>INST05</td>
<td>T 10.23%</td>
<td>T 8.84%</td>
<td>T 1.11%</td>
</tr>
<tr>
<td>INST06</td>
<td>T 9.82%</td>
<td>T 9.77%</td>
<td>T 1.39%</td>
</tr>
<tr>
<td>INST07</td>
<td>T 15.75%</td>
<td>T 8.78%</td>
<td>T 1.23%</td>
</tr>
<tr>
<td>INST08</td>
<td>T 11.55%</td>
<td>T 8.50%</td>
<td>T 1.37%</td>
</tr>
<tr>
<td>INST09</td>
<td>T 15.24%</td>
<td>T 8.48%</td>
<td>T 0.89%</td>
</tr>
<tr>
<td>INST11</td>
<td>T 7.96%</td>
<td>T 8.72%</td>
<td>T 1.13%</td>
</tr>
<tr>
<td>INST12</td>
<td>T 7.90%</td>
<td>T 6.72%</td>
<td>T 1.03%</td>
</tr>
<tr>
<td>INST13</td>
<td>T 2.97%</td>
<td>T 3.06%</td>
<td>T 0.76%</td>
</tr>
<tr>
<td>INST14</td>
<td>T 3.89%</td>
<td>T 3.67%</td>
<td>T 0.52%</td>
</tr>
<tr>
<td>INST15</td>
<td>T 3.43%</td>
<td>T 2.81%</td>
<td>T 0.78%</td>
</tr>
<tr>
<td>INST16</td>
<td>T 2.19%</td>
<td>T 3.01%</td>
<td>T 0.50%</td>
</tr>
<tr>
<td>INST17</td>
<td>T 2.09%</td>
<td>T 2.16%</td>
<td>T 0.39%</td>
</tr>
<tr>
<td>INST18</td>
<td>T 4.43%</td>
<td>T 2.60%</td>
<td>T 0.41%</td>
</tr>
<tr>
<td>INST19</td>
<td>T 3.13%</td>
<td>T 3.97%</td>
<td>T 0.46%</td>
</tr>
<tr>
<td>INST20</td>
<td>T 3.05%</td>
<td>T 4.06%</td>
<td>T 0.94%</td>
</tr>
</tbody>
</table>

#### MMKP: Relative Gap

- **Optimal**: 3
- **≤ 1% Gap**: 4
- **≤ 10% Gap**: 22
**SAS Center of Excellence in Operations Research Applications (OR COE)**

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- Cash allocation plans finalized at beginning of month - deviations from plan are costly
- **Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals
- **Constraints:**
  - Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
  - For each ATM, limit on number of days *cash-out* based on predictive model (customer satisfaction)
SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches

- Cash allocation plans finalized at beginning of month - deviations from plan are costly
- Goal: Determine multipliers for fit to minimize mismatch based on predicted withdrawals
- Constraints:
  - Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
  - For each ATM, limit on number of days cash-out based on predictive model (customer satisfaction)
ATM Cash Management Problem - Business Problem

SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM

Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches

Cash allocation plans finalized at beginning of month - deviations from plan are costly

**Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals

**Constraints:**
- Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
- For each ATM, limit on number of days cash-out based on predictive model (customer satisfaction)
SAS Center of Excellence in Operations Research Applications (OR COE)

- Determine schedule for allocation of cash inventory at branch banks to service ATMs
- Define a polynomial fit for predicted cash flow need per day/ATM
- Predictive model factors include:
  - days of the week
  - weeks of the month
  - holidays
  - salary disbursement days
  - location of the branches
- Cash allocation plans finalized at beginning of month - deviations from plan are costly

**Goal:** Determine multipliers for fit to minimize mismatch based on predicted withdrawals

**Constraints:**
- Regulatory agencies enforce a minimum cash reserve ratio at branch banks (per day)
- For each ATM, limit on number of days *cash-out* based on predictive model (customer satisfaction)
ATM Cash Management Problem - MINLP Formulation

- Simple *looking* nonconvex quadratic integer NLP.
- Linearize the absolute value, add binaries for count constraints.
- So far, no MINLP solvers seem to be able to solve this (several die with numerical failures).

\[
\begin{align*}
\text{min } & \sum_{a \in A} \sum_{d \in D} |f_{ad}| \\
\text{s.t. } & c_{ad}^x x_a + c_{ad}^y y_a + c_{ad}^{xy} x_a y_a + c_{ad}^u u_a + c_{ad} - w_{ad} = f_{ad} \quad \forall a \in A, d \in D \\
& \sum_{a \in A} (f_{ad} + w_{ad}) \leq B_d \quad \forall d \in D \\
& |\{d \in D \mid f_{ad} < 0\}| \leq K_a \quad \forall a \in A \\
& x_a, y_a \in [0, 1] \quad \forall a \in A \\
& u_a \geq 0 \quad \forall a \in A \\
& f_{ad} \geq -w_{ad} \quad \forall a \in A, d \in D
\end{align*}
\]
Application - ATM Cash Management Problem - MILP Approx Formulation

- Discretization of $x$ domain $\{0, 0.1, 0.2, ..., 1.0\}$.
- Linearization of product of binary and continuous, and absolute value.

\[
\begin{align*}
\min & \sum_{a \in A} \sum_{d \in D} \left( f_{ad}^+ + f_{ad}^- \right) \\
\text{s.t.} & \sum_{t \in T} c_{ad}^x x_{at} + c_{ad}^y y_a + c_{ad}^{xy} \sum_{t \in T} c_t z_{at} + c_{ad}^u u_a - w_{ad} = f_{ad}^+ - f_{ad}^- & \forall a \in A, d \in D \\
& \sum_{t \in T} x_{at} \leq 1 & \forall a \in A \\
& z_{at} \leq x_{at} & \forall a \in A, t \in T \\
& z_{at} \leq y_a & \forall a \in A, t \in T \\
& z_{at} \geq x_{at} + y_a - 1 & \forall a \in A, t \in T \\
& f_{ad}^- \leq w_{ad} v_{ad} & \forall a \in A, d \in D \\
& \sum_{a \in A} (f_{ad}^+ - f_{ad}^- + w_{ad}) \leq B_d & \forall d \in D \\
& \sum_{d \in D} v_{ad} \leq K_a & \forall a \in A
\end{align*}
\]
ATM Cash Management Problem - MILP Approx Formulation

\[ x_{at} \quad \in \{0, 1\} \quad \forall a \in A, t \in T \]
\[ z_{at} \quad \geq 0 \quad \forall a \in A, t \in T \]
\[ v_{ad} \quad \in \{0, 1\} \quad \forall a \in A, d \in D \]
\[ y_a \quad \in [0, 1] \quad \forall a \in A \]
\[ u_a \quad \geq 0 \quad \forall a \in A \]
\[ f_{ad}^+, f_{ad}^- \quad \in [0, w_{ad}] \quad \forall a \in A, d \in D \]

- The MILP formulation has a natural block-angular structure.
  - Master constraints are just the budget constraint.
  - Subproblem constraints (the rest) - one block for each ATM.

- Submitted to MIPLIB2010
| $|A|$ | $|D|$ | s | CPX11 Time | Gap | Nodes | DIP-PC Time | Gap | Nodes | DIP-PC+ Time | Gap | Nodes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 25 | 1 | 0.76 | OPT | 467 | 1.62 | OPT | 6 | 1.96 | OPT | 6 |
| 5 | 25 | 2 | 1.41 | OPT | 804 | 1.95 | OPT | 9 | 1.57 | OPT | 7 |
| 5 | 25 | 3 | 0.42 | OPT | 147 | 7.38 | OPT | 32 | 8.03 | OPT | 32 |
| 5 | 25 | 4 | 1.49 | OPT | 714 | 2.74 | OPT | 14 | 2.45 | OPT | 13 |
| 5 | 25 | 5 | 0.16 | OPT | 32 | 0.98 | OPT | 7 | 0.95 | OPT | 6 |
| 5 | 50 | 1 | T | 0.10 | 1264574 | 162.74 | OPT | 127 | 164.46 | OPT | 131 |
| 5 | 50 | 2 | 87.96 | OPT | 38341 | 183.28 | OPT | 273 | 263.24 | OPT | 275 |
| 5 | 50 | 3 | 8.09 | OPT | 3576 | 17.58 | OPT | 36 | 22.28 | OPT | 35 |
| 5 | 50 | 4 | 4.13 | OPT | 1317 | 3.13 | OPT | 3 | 3.17 | OPT | 3 |
| 5 | 50 | 5 | 57.55 | OPT | 32443 | 91.30 | OPT | 145 | 141.29 | OPT | 147 |
| 10 | 50 | 1 | T | 0.76 | 998624 | 297.65 | OPT | 301 | 234.47 | OPT | 156 |
| 10 | 50 | 2 | 1507.84 | OPT | 351879 | 28.84 | OPT | 29 | 52.99 | OPT | 29 |
| 10 | 50 | 3 | T | 0.81 | 667371 | 64.72 | OPT | 64 | 49.20 | OPT | 47 |
| 10 | 50 | 4 | 1319.00 | OPT | 433155 | 7.97 | OPT | 1 | 5.00 | OPT | 1 |
| 10 | 50 | 5 | 365.51 | OPT | 181013 | 12.49 | OPT | 3 | 5.18 | OPT | 3 |
| 10 | 100 | 1 | T | ∞ | 128155 | T | ∞ | 20590 | T | 0.11 | 13190 |
| 10 | 100 | 2 | T | ∞ | 116522 | T | ∞ | 60554 | 2437.43 | OPT | 135 |
| 10 | 100 | 3 | T | ∞ | 118617 | T | ∞ | 52902 | T | 0.20 | 40793 |
| 10 | 100 | 4 | T | ∞ | 108899 | T | ∞ | 47931 | T | 1.51 | 59477 |
| 10 | 100 | 5 | T | ∞ | 167617 | T | ∞ | 40283 | T | 0.38 | 26490 |
| 20 | 100 | 1 | T | ∞ | 93519 | 379.75 | OPT | 9 | 544.49 | OPT | 9 |
| 20 | 100 | 2 | T | ∞ | 68863 | T | 16.44 | 14240 | T | 0.26 | 25756 |
| 20 | 100 | 3 | T | ∞ | 95981 | T | 15.37 | 41495 | T | 0.12 | 3834 |
| 20 | 100 | 4 | T | ∞ | 81836 | T | 0.39 | 7554 | T | 0.08 | 7918 |
| 20 | 100 | 5 | T | ∞ | 101917 | 635.59 | OPT | 21 | 608.68 | OPT | 19 |
| Optimal | | | | | | | | | | | |
| ≤ 1% Gap | 12 | | | | | | | | | | |
| ≤ 10% Gap | 15 | | | | | | | | | | |
ATM: CPX11 vs PC/PC+

ATM: Solution Quality Across Methods

ATM: Relative Gap

ATM: Time to Solve

Galati, Ralphs  Progress Update on COIN/DIP  23/28
Consulting work led to numerous MILPs that cannot be solved with generic (B&C) solvers.
Often consider a decomposition approach, since a common modeling paradigm is
independent departmental policies which are then coupled by some global constraints.
Development time was slow due to problem-specific implementations of methods.

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

MILPBlock provides a black-box solver for applying integrated methods to generic MILP.
This is the first framework to do this (to my knowledge).
Similar efforts now by F. Vanderbeck BaPCod (no cuts?) and M. Lübbecke SCIP.
Currently, the only input needed is MPS/LP and a block file.
Future work will attempt to embed automatic recognition of the block-angular structure
using packages from linear algebra like: MONET, hMETIS, Mondriaan.
Consulting work led to numerous MILPs that cannot be solved with generic (B&C) solvers.

Often consider a decomposition approach, since a common modeling paradigm is:

- independent departmental policies which are then coupled by some global constraints.

Development time was slow due to problem-specific implementations of methods.

\[
\begin{pmatrix}
A''_1 & A''_2 & \cdots & A''_\kappa \\
A'_1 \\
A'_2 \\
\vdots \\
A'_\kappa
\end{pmatrix}
\]

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

- This is the first framework to do this (to my knowledge).
- Similar efforts now by F. Vanderbeck BaPCod (no cuts?) and M. Lübbecke SCIP.

Currently, the only input needed is MPS/LP and a block file.

Future work will attempt to embed automatic recognition of the block-angular structure using packages from linear algebra like: MONET, hMETIS, Mondriaan.
Application - Block-Angular MILP (applied to Retail Optimization)

SAS Retail Optimization Solution

- Multi-tiered supply chain distribution problem where each block represents a store
- Prototype model developed in SAS/OR’s OPTMODEL (algebraic modeling language)

<table>
<thead>
<tr>
<th>Instance</th>
<th>CPX11</th>
<th>DIP-PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Gap</td>
</tr>
<tr>
<td>retail27</td>
<td>T</td>
<td>2.30%</td>
</tr>
<tr>
<td>retail31</td>
<td>T</td>
<td>0.49%</td>
</tr>
<tr>
<td>retail3</td>
<td>529.77</td>
<td>OPT</td>
</tr>
<tr>
<td>retail4</td>
<td>T</td>
<td>1.61%</td>
</tr>
<tr>
<td>retail6</td>
<td>1.12</td>
<td>OPT</td>
</tr>
</tbody>
</table>
MILPBlock – Recently Added Features

**Interface Features**
- User can select which block to **process next** (alternative to *all* or *round-robin*)
- Sparse definition of blocks - user just defines the **mapping** (memory!)
- Interface to provide an **initial dual vector**

**New Options**
- Branching can be auto enforced in subproblem or master (when oracle is MILP)
- Ability to stop subproblem calculation on gap/time and calculate LB (can branch early)
- For oracles that provide it, allow **multiple columns** for each subproblem call
- Management of **compression of columns** - once master gap is tight

**Performance**
- Detection and removal of columns that are close to **parallel**
- Added basic **dual stabilization** (Wentges smoothing)
MILPBlock – Recently Added Features

Interface Features
- User can select which block to process next (alternative to all or round-robin)
- Sparse definition of blocks - user just defines the mapping (memory!)
- Interface to provide an initial dual vector

New Options
- Branching can be auto enforced in subproblem or master (when oracle is MILP)
- Ability to stop subproblem calculation on gap/time and calculate LB (can branch early)
- For oracles that provide it, allow multiple columns for each subproblem call
- Management of compression of columns - once master gap is tight

Performance
- Detection and removal of columns that are close to parallel
- Added basic dual stabilization (Wentges smoothing)
MILPBlock – Recently Added Features

Interface Features
- User can select which block to process next (alternative to all or round-robin)
- Sparse definition of blocks - user just defines the mapping (memory!)
- Interface to provide an initial dual vector

New Options
- Branching can be auto enforced in subproblem or master (when oracle is MILP)
- Ability to stop subproblem calculation on gap/time and calculate LB (can branch early)
- For oracles that provide it, allow multiple columns for each subproblem call
- Management of compression of columns - once master gap is tight

Performance
- Detection and removal of columns that are close to parallel
- Added basic dual stabilization (Wentges smoothing)
Outline

1. DIP Framework

2. Applications
   - Multi-Choice Multi-Dimensional Knapsack Problem
   - ATM Cash Management Problem
   - Generic Black-box Solver for Block-Angular MILP

3. Other Projects Using DIP
Related Projects using DIP - Work in Progress

- **OSDip** - Optimization Services (OS) wraps DIP (in CoinBazaar)
  - University of Chicago - Kipp Martin

- **Dippy** - Python interface for DIP through PuLP
  - University of Auckland - Michael O’Sullivan

- **IBM** - National Workforce Management, Cross-Training and Scheduling Project
  - IBM Business Process Re-engineering - Alper Uygur

- **DIP@SAS** - surface MILPBlock-like solver for PROC OPTMODEL
  - SAS Institute - Matthew Galati

- **Jaidong Wang** - PhD student doing work on automating the identification of block angular structure (missing piece for black box MILP solver) and parallelism
  - Lehigh University - Jaidong Wang and Ted Ralphs