A generic view at
the Dantzig-Wolfe decomposition approach
in Mixed Integer Programming:
paving the way for a generic code

François Vanderbeck

www//math.u-bordeaux.fr/~fv

Laboratoire de Mathématique Appliquées de Bordeaux (MAB)
Université Bordeaux 1
France
Use MIP solver v.s. compete against Xpress, Cplex, OSL → push back their limitations
Use MIP solver v.s. compete against Xpress, Cplex, OSL → push back their limitations

DW decomposition = well suited tool to defer combinatorial explosion
Motivation

- Use MIP solver v.s. compete against Xpress, Cplex, OSL → push back their limitations
- DW decomposition = well suited tool to defer combinatorial explosion
- DW decomposition not use to its full power (f.i.: Lagrangian decomposition)
Motivation

- Use MIP solver v.s. compete against Xpress, Cplex, OSL → push back their limitations
- DW decomposition = well suited tool to defer combinatorial explosion
- DW decomposition not use to its full power (f.i.: Lagrangian decomposition)
- Need for a generic branch-and-price code: Minto (Savelsbergh), Abacus (Thienel), . . .
Motivation

6 Use MIP solver v.s. compete against Xpress, Cplex, OSL → push back their limitations
6 DW decomposition = well suited tool to defer combinatorial explosion
6 DW decomposition not use to its full power (f.i.: Lagrangian decomposition)
6 Need for a generic branch-and-price code: Minto (Savelsbergh), Abacus (Thienel),...
6 Generic code ← generic understanding of important ingredients for DW decomp.
6 Decomposition: Why

Divide and Conquer

\[ \text{complexity} \]

\[ \text{size} \]

\[ + \]

\[ + \ldots \]
Decomposition: Why

6 Divide and Conquer

Exploit the structure

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.4/76
Decomposition: When

\[ Z^{MIP} = \min \quad c(x, y) \]
\[ A(x, y) \geq a \]
\[ B(x, y) \geq b \]
\[ x \geq 0 \]
\[ y \in \mathbb{N}^p \]
Decomposition: When

\[ Z^{MIP} = \min c(x, y) \]

\[ A(x, y) \geq a \]

\[ B(x, y) \geq b \]

\[ x \geq 0 \]

\[ y \in \mathbb{N}^P \]

Difficult Constraints
Decomposition: When

\[ Z^{MIP} = \min \quad c(x, y) \]

\[ A(x, y) \geq a \]

\[ B(x, y) \geq b \]

\[ x \geq 0 \]

\[ y \in \mathbb{N}^p \]

Difficult Constraints

Linking Constraints
Example: Cutting Stock Problem

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.6/70
Example: Cutting Stock Problem

\[
\begin{align*}
\min & \quad \sum_{k} y_k \\
\sum_{k} x_{i,k} & \geq d_i \quad \forall i \\
\sum_{i} s_i x_{i,k} & \leq y_k \quad \forall k \\
(x, y) & \in \mathbb{N}^{I+K}
\end{align*}
\]

A generic view of the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.6/76
\[ Z^{MIP} = \min \ c(x, y) \]
\[ A(x, y) \geq a \]
\[ B(x, y) \geq b \]
\[ x \geq 0 \]
\[ y \in \mathbb{N}^P \]

- Difficult Constraints
- Linking Constraints
- Multiple Sub-Systems (variable splitting)
Example: Multi-Item Lot-Sizing
Example: Multi-Item Lot-Sizing

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.9/76
Example: Multi-Item Lot-Sizing

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.10/76
Example: Multi-Item Lot-Sizing

\[
\begin{align*}
\min & \sum_{i,t} \left( \frac{p_{it}}{2} x_{it}^A + \frac{f_{it}}{2} y_{it}^A \right) + \sum_{i,t} \left( \frac{p_{it}}{2} x_{it}^B + \frac{f_{it}}{2} y_{it}^B \right) \\
& x_{it}^A \geq x_{it}^B \forall i, t \\
& \sum_{i} (x_{it}^A + s_i y_{it}^A) \leq C_t \forall t \\
& x_{it}^A \leq c_{it} y_{it}^A \forall i, t \\
& x_{it} \geq 0, y_{it}^A \in \{0, 1\} \forall i, t
\end{align*}
\]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.11/76
Decomposition: When

\[ Z^{MIP} = \min \ c(x, y) \]
\[ A(x, y) \geq a \]
\[ B(x, y) \geq b \]
\[ x \geq 0 \]
\[ y \in \mathbb{N}^P \]

- Difficult Constraints
- Linking Constraints
- Multiple Sub-Systems (variable splitting)
Decomposition: How

\[
\min\{ c(x, y) : A(x, y) \geq a, B(x, y) \geq b, x \geq 0, y \in \mathbb{N} \}
\]

difficult

nice
Decomposition: How

\[
\min \{ c(x, y) : \begin{array}{l}
A(x, y) \geq a, \\
B(x, y) \geq b, x \geq 0, y \in \mathbb{N}
\end{array} \} 
\]

- Lagrangian relaxation (Lagr. Dual)

\[
L(\pi) = \min \{ c(x, y) + \pi (a - A(x, y)) : B(x, y) \geq b, y \in \mathbb{N}^p \}
\]
Decomposition: How

\[
\min \left\{ c(x, y) : A(x, y) \geq a, \quad B(x, y) \geq b, x \geq 0, y \in \mathbb{N} \right\}
\]

difficult       nice

Lagrangian relaxation (Lagr. Dual)

\[
L(\pi) = \min \left\{ c(x, y) + \pi(a - A(x, y)) : B(x, y) \geq b, y \in \mathbb{N}^p \right\}
\]

\[
LD = \max_{\pi} L(\pi)
\]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* -- p.13/76
Decomposition: How

\[ \min \{ c(x, y) : A(x, y) \geq a, B(x, y) \geq b, x \geq 0, y \in \mathbb{N} \} \]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)
Decomposition: How

\[ \min \{ c(x, y) : A(x, y) \geq a, B(x, y) \geq b, x \geq 0, y \in \mathbb{N} \} \]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)

\[ \{ B(x, y) \geq b, (x, y) \geq 0 \} \rightarrow \{ B(x, y) \geq b, \gamma(x, y) \geq \gamma_0 \} \]
Decomposition: How difficult

\[ \min \{ c(x, y) : A(x, y) \geq a, \quad B(x, y) \geq b, \quad x \geq 0, \quad y \in \mathbb{N} \} \]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)
- Reformulation (Variable Redefinition)
Decomposition: How

\[
\min \{ c(x, y) : A(x, y) \geq a, \quad B(x, y) \geq b, \quad x \geq 0, \quad y \in \mathbb{N} \}
\]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)
- Reformulation (Variable Redefinition)

\[
\{ B(x, y) \geq b, \quad x \geq 0, \quad y \in \mathbb{N}^p \} \rightarrow \{ G(w, z) \geq g, \quad z \in \mathbb{N} \}
\]
Decomposition: How difficult

\[ \min \{ c(x, y) : A(x, y) \geq a, \quad B(x, y) \geq b, \quad x \geq 0, \quad y \in \mathbb{N} \} \]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)
- Reformulation (Variable Redefinition)

**Best Dual Bound:**

\[ \equiv \min \{ c(x, y) : A(x, y) \geq a, \quad (x, y) \in \text{conv}(\{ B(x, y) \geq b, \quad y \in \mathbb{N}^p \}) \} \]
Decomposition: How

\[
\min \{ c(x, y) : A(x, y) \geq a, \quad B(x, y) \geq b, \quad x \geq 0, \quad y \in \mathbb{N} \}
\]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)
- Reformulation (Variable Redefinition)

**Best Dual Bound:**

\[
\equiv \min \{ c(x, y) : A(x, y) \geq a, \quad (x, y) \in \text{conv}( \{ B(x, y) \geq b, y \in \mathbb{N}^p \} ) \}
\]
Decomposition: How
difficult
nice

Lagrangian relaxation (Lagr. Dual)
Cut Generation (Separation Sub-Problem)
Reformulation (Variable Redefinition)

Best Dual Bound:
\[ \min \{ c(x, y) : A(x, y) \geq a, B(x, y) \geq b, x \geq 0, y \in \mathbb{N} \} \]

\[ \equiv \min \{ c(x, y) : A(x, y) \geq a, (x, y) \in \text{conv}(\{ B(x, y) \geq b, y \in \mathbb{N} \}) \} \]
Decomposition: How

\[ \min\{c(x,y) : A(x,y) \geq a, B(x,y) \geq b, x \geq 0, y \in \mathbb{N}\} \]

- Lagrangian relaxation (Lagr. Dual)
- Cut Generation (Separation Sub-Problem)
- Reformulation (Variable Redefinition)

**Best Dual Bound:**

\[ \equiv \min\{c(x,y) : A(x,y) \geq a, (x,y) \in \text{conv}\{B(x,y) \geq b, y \in \mathbb{N}^p\}\} \]
Dantzig-Wolfe Decomposition

reformulation whose LP value achieves the Lagrangian dual bound
Dantzig-Wolfe Decomposition

reformulation whose LP value achieves the Lagrangian dual bound

\[ X^B = \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{N}^p : B(x, y) \geq b\} \]

\( G^B \) is a finite generating set for \( X^B \)

\[ X^B \equiv \{ (x, y) = \sum_{g \in G^B} g \lambda_g : \sum_{g \in G^B} \lambda_g = 1, \text{ integer restr.} \} \]
**Dantzig-Wolfe Decomposition**

A reformulation whose LP value achieves the Lagrangian dual bound.

\[ X^B = \{ (x, y) \in \mathbb{R}_+^n \times \mathbb{N}^p : B (x, y) \geq b \} \]

\( G^B \) is a finite generating set for \( X^B \)

\[ X^B \equiv \{ (x, y) = \sum_{g \in G^B} g \lambda_g : \sum_{g \in G^B} \lambda_g = 1, \text{ integer restr.} \} \]

variable change

\[ \lambda \in \mathbb{R}^B \]

Re-formulation [MASTER]:

\[ \min \sum_{g \in G^B} (c g) \lambda_g \]

\[ \sum_{g \in G^B} (A g) \lambda_g \geq a \quad (\pi) \]

\[ \lambda \in \mathbb{R}^B \]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.21/76
Solving the Master LP

6 Column Generation: graphically

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.22/76
Solving the Master LP

Column Generation: graphically
Solving the Master LP

Column Generation: graphically
Solving the Master LP

6 Column Generation: graphically

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.24/76
Solving the Master LP

6 Column Generation: graphically

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.25/76
6 Column Generation

MASTER:

$$\min \sum_{g \in G^B_{\text{restricted}}} (c_g) \lambda_g$$

$$\sum_{g \in G^B_{\text{restricted}}} (A_g) \lambda_g \geq a$$

$$\lambda \in R^B_{LP}$$

SUB-PROBLEM: Check optimality: $\min\{(c - \pi A) g : g \in G^B\} < 0$ ?

$$\min\{(c - \pi A)(x, y) : B(x, y) \geq b, x \geq 0, y \in \mathbb{N}^p\}.$$
Column Generation = Kelley (in the dual space)

\[ \theta = \max_{(\pi, \eta)} \eta \]
\[ \eta + (A g - a) \pi \leq c g \quad g \in G^B \]
Column Generation = Kelley (in the dual space)

DUAL MASTER:

\[ \theta = \max_{(\pi, \eta)} \eta \]

\[ \eta + (A g - a) \pi \leq c g \quad \text{for } g \in G^B \]

\[ \eta = L(\pi) \]
Solving the Master LP

Column Generation = Kelley (in the dual space)

DUAL MASTER:

\[ \theta = \max_{(\pi, \eta)} \eta \]
\[ \eta + (A g - a) \pi \leq c g \quad g \in G^B \]

\[ \eta = L(\pi) \]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code — p.28/76
Column Generation = Kelley (in the dual space)

**DUAL MASTER:**

\[
\theta = \max_{(\pi, \eta)} \eta \\
\eta + (A g - a) \pi \leq c g \quad g \in G^B
\]

\[\eta = L(\pi)\]
Column Generation = Kelley (in the dual space)

DUAL MASTER:

\[ \theta = \max_{(\pi, \eta)} \eta \]

\[ \eta + (A g - a) \pi \leq c g \quad g \in G^B \]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.30/76
Column Generation = Kelley (in the dual space)

DUAL MASTER:

\[ \theta = \max_{(\pi, \eta)} \eta \]

\[ \eta + (A g - a) \pi \leq c g \quad g \in G^B \]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.31/76
Column Generation = Kelley (in the dual space)

DUAL MASTER:

\[ \theta = \max_{(\pi, \eta)} \eta \]

\[ \eta + (A g - a) \pi \leq c g \quad g \in G^B \]

\[ \eta = L(\pi) \]
Col. Gen. / Kelley: the issue of convergence

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.33/76
Col. Gen. / Kelley: the issue of convergence
Col. Gen. / Kelley: the issue of convergence

- restricted Master LP values
- intermediate Lagrangian bounds
- tailing-off effect

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.34/76
Solving the Master LP

Col. Gen. / Kelley: the issue of convergence

- tailing-off effect
- heading-in effect

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.34/76
Solving the Master LP

- Column Generation = Kelley
- Stabilization techniques:
Solving the Master LP

6 Column Generation = Kelley
6 Stabilization techniques: the basics
   △ warm start: initialize master f.i. with SP opt. sol
Solving the Master LP

6 Column Generation = Kelley

6 Stabilization techniques: the basics
   △ warm start: initialize master f.i. with SP opt. sol
   △ inequality constraint in the Master (→ \( \pi \geq 0 \))
Solving the Master LP

6 Column Generation = Kelley

6 Stabilization techniques: the basics
   ▲ warm start: initialize master f.i. with SP opt. sol
   ▲ inequality constraint in the Master ($\pi \geq 0$)
   ▲ include artificial missing columns
Column Generation = Kelley

Stabilization techniques: the basics

- **warm start**: initialize master f.i. with SP opt. sol
- **inequality** constraint in the Master ($\pi \geq 0$)
- include artificial **missing columns**

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.35/76
Solving the Master LP

- Column Generation = Kelley

- Stabilization techniques: the basics
  - warm start: initialize master f.i. with SP opt. sol
  - inequality constraint in the Master (\( \pi \geq 0 \))
  - include artificial missing columns
Solving the Master LP

- Column Generation = Kelley
- Stabilization techniques: the basics
  - **warm start**: initialize master f.i. with SP opt. sol
  - **inequality** constraint in the Master \((\rightarrow \pi \geq 0)\)
  - include artificial missing columns

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.37/76
Solving the Master LP

6 Column Generation = Kelley

6 Stabilization techniques: the basics
   △ warm start: initialize master f.i. with SP opt. sol
   △ inequality constraint in the Master (→ \( \pi \geq 0 \))
   △ include artificial missing columns
     (cost initially low → \( \pi \) interior point)
   △ use exchange vectors (→ dual cut)

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.38/76
Solving the Master LP

- Column Generation = Kelley
- Stabilization techniques: the basics
  - warm start: initialize master f.i. with SP opt. sol
  - inequality constraint in the Master ($\pi \geq 0$)
  - include artificial missing columns
    (cost initially low $\rightarrow \pi$ interior point)
  - use exchange vectors ($\rightarrow$ dual cut)
- Sub-gradient algorithm

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.38/76
Solving the Master LP

6 Column Generation = Kelley
6 Stabilization techniques: the basics
   ▲ warm start: initialize master f.i. with SP opt. sol
   ▲ inequality constraint in the Master (\( \pi \geq 0 \))
   ▲ include artificial missing columns
     (cost initially low \( \rightarrow \pi \) interior point)
   ▲ use exchange vectors (\( \rightarrow \) dual cut)

6 Sub-gradient algorithm
6 Bundle method (Lemaréchal)
Solving the Master LP

- Column Generation = Kelley
- Stabilization techniques: the basics
  - warm start: initialize master f.i. with SP opt. sol
  - inequality constraint in the Master \( \rightarrow \pi \geq 0 \)
  - include artificial missing columns
    (cost initially low \( \rightarrow \pi \) interior point)
  - use exchange vectors \( \rightarrow \) dual cut
- Sub-gradient algorithm
- Bundle method (Lemaréchal)
- ACCPM (Goffin and Vial)

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.38/76
Solving the Master LP

- Column Generation = Kelley
- Stabilization techniques: the basics
  - warm start: initialize master f.i. with SP opt. sol
  - inequality constraint in the Master ($\pi \geq 0$)
  - include artificial missing columns (cost initially low $\rightarrow \pi$ interior point)
  - use exchange vectors ($\rightarrow$ dual cut)

- Sub-gradient algorithm
- Bundle method (Lemaréchal)
- ACCPM (Goffin and Vial)
  Combine with branch-and-Bound

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.38/76
A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code — p.39/76.
1. Discretization of an unbounded IP
1. Discretization of an unbounded IP
1. Discretization of an unbounded IP
1. Discretization of an unbounded IP
1. Discretization of an unbounded IP

\[
\begin{align*}
\{y &= \sum_{p \in P} p \lambda_p + \sum_{r \in R} r \lambda_r : \sum_{p \in P} \lambda_p = 1, \lambda_p \in \{0, 1\} \forall p, \lambda_r \in \mathbb{N} \forall r}\end{align*}
\]
2. Discretization of an bounded IP

3. Convexification of a bounded IP
4. Discretization of an bounded IP

5. Convexification of a bounded IP
6. Discretization of an bounded IP

7. Convexification of a bounded IP
8. Discretization of a MIP

9. Its Projection in the IP variables
8. Discretization of a MIP

9. Its Projection in the IP variables

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.47/76
Formulating Integrality restrictions

6 Discretization approach:
true IP re-formulation

6 Convexification approach:
mere LP relaxation re-formulation
Formulating Integrality restrictions

6 Discretization approach: true IP re-formulation

pure IP: $\lambda_g \in \mathbb{N} \quad \forall \ g \in G_d$
MIP: $\sum_{g \in G(y)} \lambda_g \in \mathbb{N} \quad \forall \ y \in G_p$

6 Convexification approach: mere LP relaxation re-formulation
Formulating Integrality restrictions

Discretization approach:
true IP re-formulation

pure IP: \( \lambda_g \in \mathbb{IN} \quad \forall \ g \in G_d \)
MIP: \( \sum_{g \in G(y)} \lambda_g \in \mathbb{IN} \quad \forall \ y \in G_p \)

Convexification approach:
mere LP relaxation re-formulation

\( y = \sum_{g \in G} y^g \lambda_g \in \mathbb{IN} \)
Branching: example

Assume: pure IP, single sub-problem, \[ \sum_{g \in G} \lambda_g = 1 \]
Branching: example

\[ y_i \leq \lfloor \alpha_i \rfloor \quad \text{or} \quad y_i \geq \lceil \alpha_i \rceil \]
Branching: example

under convexification

\[ \sum_{g \in G_c} y_i^g \lambda_g \leq \lfloor \alpha_i \rfloor \quad \text{or} \quad \sum_{g \in G_c} y_i^g \lambda_g \geq \lceil \alpha_i \rceil \]
Branching: example

under discretization

\[ \sum_{g \in G_d: y^g_i \geq \lfloor \alpha_i \rfloor} \lambda_g \leq 0 \quad \text{or} \quad \sum_{g \in G_d: y^q_i \leq \lceil \alpha_i \rceil} \lambda_g \leq 0 \]
Branching: example

under discretization

Stronger Dual Bound

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.53/76
Branching: general case
Branching: general case

\[
\sum_{g \in \hat{G}} \lambda_g \leq \lfloor \alpha \rfloor \quad \text{or} \quad \sum_{g \in \hat{G}} \lambda_g \geq \lceil \alpha \rceil
\]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.54/76
Branching: general case

\[ \sum_{g \in \hat{G}} \lambda_g \leq \left\lfloor \alpha \right\rfloor = 0 \quad \text{or} \quad \sum_{g \in \hat{G}} \lambda_g \geq \left\lceil \alpha \right\rceil = U \]

\[ G := G \setminus \hat{G} \quad \text{or} \quad G := \hat{G} \]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.55/76
A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.56/76
Proper Columns

\[ Z^{IP} = \min \quad c^T y \]

\[ A^T y \geq a \]

\[ B^T y \geq b \]

\[ y \in \mathbb{N}^p \]

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.56/76
$$Z^{IP} = \min \quad c^T y$$

$$A y \geq a$$

$$B y \geq b$$

$$y \in \mathbb{N}^p$$

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.57/76
\[ Z^{IP} = \min \quad cy \]
\[ A y \geq a \]
\[ B y \geq b \]
\[ y \in N^p \]

Improved Dual Bound

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.58/76
Proper Columns: pseudo-definition

\[ Z^{IP} = \min \ c^T y \]
\[ A y \geq a \]
\[ B y \geq b \]
\[ y \in \mathbb{N}^p \]

\[ g \in G^B \text{ is proper if } l_A \leq g \leq u_A \]

where \( l_A \) and \( u_A \) are component bounds “implied by” master constraints \( A \).
Proper Columns: examples

1. Cutting Stock Problem

\[ x_i \leq d_i \]

(2-d knapsack sub-problem gets strongly NP-Hard)
1. Cutting Stock Problem

\[ x_i \leq d_i \]

(2-d knapsack sub-problem gets strongly NP-Hard)

2. Multi-Item Lot-Sizing Problem

\[ x_{i,t} \leq C_t \]

(Capacitated Lot-Sizing sub-problem is NP-Hard)
Strongly Proper Columns

push pre-processing further (what if questions)
Strongly Proper Columns

push pre-processing further (what if questions)

pseudo-definition:

\[ g \in G^B \text{ is strongly proper if compon. bounds are s.t.} \]

\[ \lambda_g = 1 \text{ yields a residual problem that is not infeasible} \]
Strongly Proper Columns

push pre-processing further (what if questions)

pseudo-definition:
\[ g \in G^B \text{ is strongly proper if compon. bounds are s.t. } \lambda_g = 1 \text{ yields a residual problem that is not infeasible} \]

Example: Multi-Item Lot-Sizing Problem

\[ x_{i,t} = C_t \Rightarrow x_{j,t} = 0 \quad \forall j \neq i \]
Strongly Proper Columns

push pre-processing further (what if questions)

pseudo-definition:
\[ g \in G^B \text{ is strongly proper if compon. bounds are s.t. } \lambda_g = 1 \text{ yields a residual problem that is not infeasible} \]

Example: Multi-Item Lot-Sizing Problem

\[ x_{i,t} = C_t \Rightarrow x_{j,t} = 0 \quad \forall j \neq i \]

(compute bounds that account for capacity requirement of other products)

important for rounding heuristic

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.61/76
State Space relaxation

*relax the definition of the generating set*

easier sub-problem → weaker dual bound
State Space relaxation

relax the definition of the generating set

easier sub-problem $\rightarrow$ weaker dual bound

different states $g$

of the Universe $G$

mapping $\rightarrow$

base-pattern

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.62/76
**Base-Patterns: Cutting Stock**

**Definition:** (Fekete and Schepers, 1998)

A *mapping* \( u: s \in \mathbb{R} \rightarrow u(s) \in \mathbb{R} \) is dual feasible for a CSP instance if

\[
\sum_{i} u(s_i) g_i \leq 1 \quad \forall g \in KNP
\]

where *KNP* is the set of feasible knapsack solution using the original sizes \( s_i \)'s.
**Base-Patterns: Cutting Stock**

**Definition:** (Fekete and Schepers, 1998)

A *mapping* $u: s \in \mathbb{R} \rightarrow u(s) \in \mathbb{R}$ is dual feasible for a CSP instance if

$$
\sum_{i} u(s_i) g_i \leq 1 \quad \forall g \in \text{KNP}
$$

where $\text{KNP}$ is the set of feasible knapsack solution using the original sizes $s_i$’s.

fewer different sizes \quad \text{smaller numbers} \quad \Rightarrow \quad \text{easier knapsack SP}

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.63/76
Continuous Single Item Lot-Sizing sub-problem

\[ \{ x \in \mathbb{R}_+^T, y \in \{0, 1\}^T : \sum_{\tau=1}^{t} x_{\tau} \geq d_{1t} \ \forall t, \ x_{t} \leq c_{it} \ y_{t} \ \forall t \} \]

\[ \downarrow \]

Discrete Single Item Lot-Sizing sub-problem

\[ \{ y \in \{0, 1\}^T : \sum_{\tau=1}^{t} c_{i\tau} \ y_{\tau} \geq d_{1t} \ \forall t \} \]
Uses of Base-Pattern Relaxation

- relaxed \( G \Rightarrow \) weaker dual bound
- cheaper B-a-P node comput. but larger B-a-P tree

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.65/76
Uses of Base-Pattern Relaxation

- relaxed $G \Rightarrow$ weaker dual bound
- cheaper B-a-P node comput. but larger B-a-P tree
- warm start for column generation:
  relaxed columns acts as artificial columns
Uses of Base-Pattern Relaxation

- relaxed $G \Rightarrow$ weaker dual bound
  cheaper B-a-P node comput. but larger B-a-P tree

- warm start for column generation:
  relaxed columns acts as artificial columns

- Scaling approach: iteratively refined mapping
Uses of Base-Pattern Relaxation

- relaxed $G \Rightarrow$ weaker dual bound
  cheaper B-a-P node comput. but larger B-a-P tree

- warm start for column generation:
  relaxed columns acts as artificial columns

- Scaling approach: iteratively refined mapping

- 2-stage SP optimization: primal-dual heuristic for SP

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.65/76
Uses of Base-Pattern Relaxation

- relaxed $G \Rightarrow$ weaker dual bound
cheaper B-a-P node comput. but larger B-a-P tree

- warm start for column generation:
relaxed columns acts as artificial columns

- Scaling approach: iteratively refined mapping

- 2-stage SP optimization: primal-dual heuristic for SP

- reduced cost re-optimization from base-pattern

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.65/76
Uses of Base-Pattern Relaxation

- relaxed $G \Rightarrow$ weaker dual bound
  cheaper B-a-P node comput. but larger B-a-P tree
- warm start for column generation:
  relaxed columns acts as artificial columns
- Scaling approach: iteratively refined mapping
- 2-stage SP optimization: primal-dual heuristic for SP
- reduced cost re-optimization from base-pattern
- include exchange vectors between columns sharing the same base-pattern
Analysis of the Generating Set

- Convexification v.s. Discretization
- Proper Columns and Strongly Proper Col.
- State Space-Relaxation and Base-pattern
- Dominant/Redundant Columns: lifting

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: paving the way for a generic code – p.66/76
Combining Col. Gen. with other techniques

6 Cutting planes
Combining Col. Gen. with other techniques

- Cutting planes
- Preprocessing, Variable fixing
  (master information passed onto sub-problem)
Combining Col. Gen. with other techniques

- Cutting planes
- Preprocessing, Variable fixing
  (master information passed onto sub-problem)
- Primal Heuristics:
  - greedy
  - local search
  - rounding

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.67/76
Combining Col. Gen. with other techniques

- Cutting planes
- Preprocessing, Variable fixing (master information passed onto sub-problem)
- Primal Heuristics:
  - greedy
  - local search
  - rounding
- Hybrid Algorithms:
  f.i. sub-gradient + col gen (Fischetti)
a generic Branch-And-Price Code: C++ subroutine library
a generic Branch-And-Price Code: C++ subroutine library

- implements:
  1. an automatic Dantzig-Wolfe re-formulation
  2. a Branch-And-Price(-and-Cut) algorithm (+ primal heuristics)
a generic Branch-And-Price Code: C++ subroutine library

- implements:
  1. an automatic Dantzig-Wolfe re-formulation
  2. a Branch-And-Price(-and-Cut) algorithm (+ primal heuristics)

- User input:
a generic Branch-And-Price Code: C++ subroutine library

- implements:
  1. an automatic Dantzig-Wolfe re-formulation
  2. a Branch-And-Price(-and-Cut) algorithm (+ primal heuristics)

- User input:
  1. Data definition and reading
a generic Branch-And-Price Code: C++ subroutine library

- implements:
  1. an automatic Dantzig-Wolfe re-formulation
  2. a Branch-And-Price(-and-Cut) algorithm (+ primal heuristics)

- User input:
  1. Data definition and reading
  2. Variable and Constraint C++ Classes
A generic Branch-And-Price Code: C++ subroutine library

- implements:
  1. an **automatic** Dantzig-Wolfe re-formulation
  2. a **Branch-And-Price**(-and-Cut) algorithm (+ primal heuristics)

- **User input:**
  1. Data definition and reading
  2. Variable and Constraint C++ Classes
  3. Call to constructors of Variables, Constraints, Master and Sub-problems
class Period
{
    double C_t; // capacity

    Period(const Double & capacity): C_t(capacity) {}
    ~Period(){};
};

class Item
{
    double s_i; // capacity consumed in a setup
    double f_i; // setup cost
    map< Period *, Double> p_it; // production cost
    map< Period *, Double> d_it; // demand
    map< Period *, Double> c_it; // production capacity

    Item(...) {...}
    ~Item(){};
};
class YitGenVar: public GenericVar
{
    YitGenVar(vector<Item *> & itemPts,
              vector<Period *> & periodPts,
              map<IndexCell, SpConf *> & spConfMap)
    {
        for (vector<Item *>::iterator itemPt = ...)
            for (vector<Period *>::iterator periodPt = ...)
                {
                    new InstanciatedVar(...);
                }
    }
    ~YitGenVar(){}
};

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.70/76
class YABitGenConstr: public GenericConstr
{
    YABitGenConstr(vector<Item*> ..., vector<Period*> ..., 
                    MasterConf* masterPtr)
    {... new InstanciatedConstr(masterPtr, ...); }

    const bool genericCoef(InstanciatedConstr * iconstrPtr, 
                            InstanciatedVar * ivarPtr)
    {
        YitGenVar* YitPtr = dynamic_cast<YitGenVar*>(ivarPtr->genVarConstrPtr());
        if (YitPtr == NULL) return 0;
        if (iconstrPtr->id() != ivarPtr->id()) return 0;
        SpConf* SpConfPtr = dynamic_cast<SpConf*>(ivarPtr->probConfigPtr());
        if (_itemSpConfPts.count(SpConfPtr)) return 1; else return -1;
    }
};
readData();
_masterPtr = new MasterConf();
for (vector<Item *> ...) new SpConf(Item..., _masterPtr, NULL, 1);
for (vector<Period *>...) new SpConf(Period..., _masterPtr, 1, NULL);

new XitGenVar(itemPts, periodPts, itemSpConfPts);
new XitGenVar(itemPts, periodPts, periodSpConfPts);
new YitGenVar(itemPts, periodPts, itemSpConfPts);
new YitGenVar(itemPts, periodPts, periodSpConfPts);
new XABitGenConstr(itemPts, periodPts,_masterPtr);
new YABitGenConstr(itemPts, periodPts,_masterPtr);
new DemCOVitGenConstr(itemPts, periodPts, itemSpConfPts);
new XitUbGenConstr(itemPts, periodPts, itemSpConfPts);
new CAPtGenConstr(periodPts, periodSpConfPts);
new XitUbGenConstr(itemPts, periodPts, periodSpConfPts);

_masterPtr->prepareProbConfig();
_masterPtr->solve();
## Bapcod: MICLS Preliminary Results

<table>
<thead>
<tr>
<th>Instance</th>
<th>i6-t15</th>
<th>i6-t30</th>
<th>i12-t15</th>
<th>i12-t30</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP relaxation*</td>
<td>11019.8</td>
<td>14145.5</td>
<td>16786.0</td>
<td>18124.2</td>
</tr>
<tr>
<td>Cut. Plane*</td>
<td>37213.3</td>
<td>60963.2</td>
<td>73848.0</td>
<td>130177.0</td>
</tr>
<tr>
<td>DW decomp ULS</td>
<td>37202.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW decomp CLS</td>
<td>37220.0</td>
<td>60947.0</td>
<td>73848.0</td>
<td>130180.0</td>
</tr>
<tr>
<td>DW decomp, strongly proper col</td>
<td>37227.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lagrangian decomp</td>
<td>37223.0</td>
<td>60947.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incumbent IP sol</td>
<td>*37721.0</td>
<td>*61765.0</td>
<td>*74634.0</td>
<td>131027.0</td>
</tr>
</tbody>
</table>

* Belvaux and Wolsey (instances from Trigeiro et al., 1989)

---

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code* – p.73/76
Selection of sub-systems A and B

- Tested computationally
- Compared theoretically: dual bound vs sub-prob. complexity (Thizy analyses MICLS)
- Consider duplicating constraints
- Consider adding implicit constraints
- Consider nested decomposition

\[
\begin{align*}
\text{DW decomposition} & \quad \rightarrow \quad \{ \bigotimes_{t=1}^{T} A_{L}^{t} \cap \bigotimes_{i=1}^{I} C(B^{i}) \} \\
\text{Lagrang. decomp.} & \quad \rightarrow \quad \{ \bigotimes_{t=1}^{T} C(A^{t}) \cap \bigotimes_{i=1}^{I} C(B^{i}) \} \\
\text{Nested decomp.} & \quad \rightarrow \quad \{ \bigotimes_{t=1}^{T} C(A^{t} \cap \bigotimes_{i=1}^{I} C(B^{i})) \cap \bigotimes_{i=1}^{I} C(B^{i}) \} 
\end{align*}
\]
BaPCod’s immediate future

1. See whether plain use of $BaPCod + Mip$ solver better than $Mip$-solver applied to original formulation.

2. Integrate “improvement techniques” and test their efficiency across ≠ applications.
## Bapcod: MICLS Preliminary Results

<table>
<thead>
<tr>
<th>Instance</th>
<th>i6-t15</th>
<th>i6-t30</th>
<th>i12-t15</th>
<th>i12-t30</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP relaxation*</td>
<td>11019.8</td>
<td>14145.5</td>
<td>16786.0</td>
<td>18124.2</td>
</tr>
<tr>
<td>Cut. Plane*</td>
<td>37213.3</td>
<td>60963.2</td>
<td>73848.0</td>
<td>130177.0</td>
</tr>
<tr>
<td>DW decomp ULS</td>
<td>37202.0</td>
<td>60947.0</td>
<td>73848.0</td>
<td>130180.0</td>
</tr>
<tr>
<td>DW decomp CLS</td>
<td>37220.0</td>
<td>60947.0</td>
<td>73848.0</td>
<td>130180.0</td>
</tr>
<tr>
<td>DW decomp, strongly proper col</td>
<td>37227.0</td>
<td>60947.0</td>
<td>73848.0</td>
<td>130180.0</td>
</tr>
<tr>
<td>Lagrangian decomp</td>
<td>37223.0</td>
<td>60947.0</td>
<td>73854.0</td>
<td>130180.0</td>
</tr>
<tr>
<td>Incumbent IP sol</td>
<td>*37721.0</td>
<td>*61765.0</td>
<td>*74634.0</td>
<td>131027.0</td>
</tr>
</tbody>
</table>

A generic view at the Dantzig-Wolfe decomposition approach in Mixed Integer Programming: *paving the way for a generic code*. — p.76/76