The Long Road to Practical Decomposition Methods
Part III: Many Twists and Turns
Part IV: A Useful Companion on the Road

Antonio Frangioni
Dipartimento di Informatica, Università di Pisa

AIRO PhD School 2021
&
5th AIRO Young Workshop

“Napoli” — February 9, 2021
Meta–Outline

- Part I: Why Leaving the Bed At All?
- Part II: The Long Journey Begins
- Part III: Many Twists and Turns
- Part IV: A Useful Companion on the Road
Outline – Parts III & IV

1. Stabilization
2. Dual-Optimal Cuts
3. Cuts Selection
4. Disaggregated Model
5. Easy Components
6. Structured Decomposition
7. Incremental, Inexact, Asynchronous
8. A Useful Companion on the Road
9. Conclusions (for good)
Part III:
Many Twists and Turns
Stabilization
Issue with the Cutting-Plane approach: instability

- $y_{k+1}^{*}$ can be very far from $y_k^{*}$, where $f_B$ is a “bad model” of $f$.
Issue with the Cutting-Plane approach: instability

- $y_{k+1}^*$ can be very far from $y_k^*$, where $f_B$ is a “bad model” of $f$

... as a matter of fact, infinitely far

- $(\Pi_B)$ empty $\equiv (\Delta_B)$ unbounded $\Rightarrow$ Phase 0 / Phase 1 approach

- More in general: $\{y_k^*\}$ is unstable, has no locality properties $\equiv$ convergence speed does not improve near the optimum
The effects of instability

- What does it mean?
  - a good (even perfect) estimate of dual optimum is useless!
  - frequent oscillations of dual values
  - “bad quality” of generated columns

→ tailing off, slow convergence
The effects of instability

- What does it mean?
  - a good (even perfect) estimate of dual optimum is useless!
  - frequent oscillations of dual values
  - “bad quality” of generated columns

⇒ tailing off, slow convergence

- The solution is pretty obvious: stabilize it

- Gedankenexperiment: starting from known dual optimum, constrain duals in a box of given width

<table>
<thead>
<tr>
<th>width</th>
<th>time</th>
<th>iter.</th>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>∞</td>
<td>4178.4%</td>
<td>509%</td>
<td>37579%</td>
</tr>
<tr>
<td>200.0</td>
<td>835.5</td>
<td>119</td>
<td>9368</td>
</tr>
<tr>
<td>20.0</td>
<td>117.9</td>
<td>35</td>
<td>2789</td>
</tr>
<tr>
<td>2.0</td>
<td>52.0</td>
<td>20</td>
<td>1430</td>
</tr>
<tr>
<td>0.2</td>
<td>47.5</td>
<td>19</td>
<td>1333</td>
</tr>
</tbody>
</table>

Works wonders! . . .
Stabilizing DW/Lagrange/CG

...if only we knew the dual optimum! (which we don’t)

- Current point $\bar{y}$, box of size $t > 0$ (how chosen??) around it

- Stabilized dual master problem $^{[34]}$

  \[
  (\Delta_B, \bar{y}, t) \quad \min \{ f_B(\bar{y} + d) : \|d\|_\infty \leq t \} \tag{1}
  \]

- Corresponding stabilized primal master problem

  \[
  (\Pi_B, \bar{y}, t) \quad \max \{ cx + \bar{y}s - t\|s\|_1 : s = b - Ax, \ x \in \text{conv}(B) \}
  \]

  i.e., just Dantzig-Wolfe with slacks $(s)$

- When $f(\bar{y} + d^*) \ll f(\bar{y})$, move $\bar{y} = \bar{y} + d^*$ (“serious step”)

- Uses just LP tools, relatively minor modifications to $(\Delta_B)$

- Does this really work?

---

Computational results of the boxstep method (pds7)

- Pure multicommodity flow instance (no design)
- Left = distance from final dual optimum
  right = relative gap with optimal value
- Stabilized with (fixed) different $t$, un-stabilized ($t = \infty$)
- One can clearly over-stabilize
Computational results of the boxstep method (pds18)

- All cases show a “combinatorial tail” where convergence is very quick.
- $t = 1e+3$: “smooth but slow” until the combinatorial tail kicks in, a short-step approach not unlike subgradient methods\cite{35}.
- $t = \infty$: apparently trashing along until some magic threshold is hit.
- “Intermediate” $t$ work best, but pattern not clear.

\cite{35} Camerini, Fratta, Maffioli “On Improving Relaxation Methods by Modified Gradient Techniques” Math. Prog. Study, 1975
Computational results of the boxstep method (pds30)

- $t = 1\text{e}+5$: initially even worse than $t = \infty$ but ends up faster
- Clearly, some on-line tuning of $t$ would be appropriate
- Perhaps a different stabilizing term would help? Why not\[36\]
  $$\left(\Delta_B, \bar{y}, t\right) = \min \left\{ f_B(\bar{y} + d) + \frac{1}{2t} \| d \|_2^2 \right\}$$
- "Because it’s not LP” $\implies$ a different duality need be used

Generalized proximal/trust region stabilization

- **General stabilizing term** $\mathcal{D}$, stabilized dual problem
  
  $$(\Delta \bar{y}, \mathcal{D}) \quad \phi_{\mathcal{D}}(\bar{y}) = \min \left\{ f(\bar{y} + d) + \mathcal{D}(d) \right\}$$

  with proper $\mathcal{D}$, $\phi_{\mathcal{D}}$ has same minima as $f$ but is “smoother”

- **Stabilized primal problem** = Fenchel’s dual of $(\Delta \bar{y}, \mathcal{D})$
  
  $$(\Pi \bar{y}, \mathcal{D}) \quad \min \left\{ f^*(s) - s\bar{y} + \mathcal{D}^*(-s) \right\}$$

  where $f^*(x) = \max_s \{ xs - f(s) \}$ the Fenchel’s conjugate of $f$

- For our dual $f$, a **generalized augmented Lagrangian**
  
  $$\max \left\{ cx + \bar{y}(b - Ax) - \mathcal{D}^*(Ax - b) : x \in \text{conv}(X) \right\}$$

- A “primal” exists even for a non-dual $f$: $v(\Pi) = -f^*(0) = v(\Delta)$ for
  
  $$\max \left\{ -f^*(s) : s = 0 \right\}$$

- General theory exist[37], but never mind

---

Classical stabilizing terms

\[ D = \frac{1}{2t} \| \cdot \|_2 \]
\[ D^* = \frac{1}{2} t \| \cdot \|_2 \]

\[ D = \frac{1}{t} \| \cdot \|_1 \]
\[ D^* = I_{B\infty}(1/t) \]

\[ D = I_{B\infty}(t) \]
\[ D^* = t \| \cdot \|_1 \]
Fancier stabilizing terms (very nonlinear)

- Smooth approximation of $\| \cdot \|_1$ \[^{[38]}\]

\[
\mathcal{D}^*(s) = \sum_i \Phi^*_\varepsilon(s_i) = \begin{cases} 
    s_i^2/(2\varepsilon) & \text{if } -\varepsilon \leq s_i \leq \varepsilon \\
    |s_i| - \frac{\varepsilon}{2} & \text{otherwise}
\end{cases}
\]

- Smooth approximation of $t\| \cdot \|_\infty$ \[^{[5]}\]

\[
\mathcal{D}^*(s) = \ln \sum_i e^{ts_i}
\]

- Bregman functions \[^{[39]}\]

\[
\mathcal{D}_{\bar{y}}(d) = \left( \psi(\bar{y} + d) - \psi(\bar{y}) - \nabla\psi(\bar{y})d \right)
\]

with $\psi$ fixed, strictly convex, differentiable, with compact level sets

- Others ($\varphi$-divergences, . . .), all “very nonlinear”

---

[^38]: Pinar, Zenios “Parallel Decomposition of Multicommodity [. . .] Using a Linear-Quadratic [. . .]” ORSA J. Comp., 1992

[^39]: Chen, Teboulle “Convergence Analysis of a Proximal-like Minimization Algorithm Using Bregman Functions” SIOPT, 1993
A 5-piecewise-linear function

Trust region on $\bar{y}$ + small penalty close + much larger penalty farther$^{[40]}$

\[ dK + \varepsilon^- + \varepsilon^+ + \Delta^- + \Delta^+ + \Gamma^- + \Gamma^+ \]

Slightly simplified version: only 3 pieces.

---

$^{[40]}$ Ben Amor, Desrosiers, F. “On the Choice of Explicit Stabilizing Terms in Column Generation” DAM, 2009
A 5-piecewise-linear master problem

\[ \begin{align*}
(\Pi_{B,\bar{y},D}) & \quad \text{max} \quad c \left( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} \right) - \bar{y} \left( s'_- + s''_+ - s''_- - s'_+ \right) \\
 & \quad + \gamma^- s'_- + \delta^- s''_+ + \delta^+ s''_- + \gamma^+ s'_+ \\
 & \quad + A \left( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} \right) + s'_- + s''_- - s''_+ - s'_+ = b \\
 & \quad \sum_{\bar{x} \in B} \theta_{\bar{x}} = 1 \ , \ \theta_{\bar{x}} \geq 0 \quad \bar{x} \in B \\
 & \quad 0 \leq s'_- \leq \zeta^- \ , \ 0 \leq s'_+ \leq \zeta^+ \\
 & \quad 0 \leq s''_- \leq \epsilon^- \ , \ 0 \leq s''_+ \leq \epsilon^+
\end{align*} \]
A 5-piecewise-linear master problem

\[ \max \left\{ c \left( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} \right) - \bar{y} \left( s'_- + s''_- - s'_{-} - s'_{+} \right) + \gamma^- s'_- + \delta^- s''_- + \delta^+ s'_{-} + \gamma^+ s'_{+} \right\} \]

\[ A \left( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} \right) + s'_- + s''_- - s''_{-} - s'_{+} = b \]

\[ \sum_{\bar{x} \in B} \theta_{\bar{x}} = 1, \quad \theta_{\bar{x}} \geq 0, \quad \bar{x} \in B \]

\[ 0 \leq s'_{-} \leq \zeta^- , \quad 0 \leq s'_{+} \leq \zeta^+ \]

\[ 0 \leq s''_{-} \leq \varepsilon^- , \quad 0 \leq s''_{+} \leq \varepsilon^+ \]

- Same constraints as \((\Pi_B)\), 4 slack variables for each constraint
- Many parameters: widths \(\Gamma^\pm\) and \(\Delta^\pm\), penalties \(\zeta^\pm\) and \(\varepsilon^\pm\), different roles for small and large penalties
- Large penalties \(\zeta^\pm\) easily make \((\Delta_B,\bar{y},D)\) bounded \(\implies\) no Phase 0
- 3-pieces: either large penalty \(\implies\) small moves, or small penalty \(\implies\) instability
On unboundedness and early termination

- A ray $\chi$ of $X$: $x \in X \implies x + \lambda \chi \in X$ for $\lambda \to \infty \implies (c - yA)\chi > 0 \implies f(y) = \infty \implies$ constraint $cr \leq y(A\chi)$ in the dual

- One might even hide the convexity constraint:
  - $A\bar{x} \rightarrow [A\bar{x}, 1], \ b \rightarrow [b, 1]$;
  - Ignoring the special role of $v$ (just another $y$)
  - Advantage: everything is a constraint
On unboundedness and early termination

- A ray $\chi$ of $X$: $x \in X \implies x + \lambda \chi \in X$ for $\lambda \to \infty \implies (c - yA)\chi > 0 \implies f(y) = \infty \implies$ constraint $cr \leq y(A\chi)$ in the dual

- One might even hide the convexity constraint:
  - $A\bar{x} \to [A\bar{x}, 1]$, $b \to [b, 1]$;
  - Ignoring the special role of $v$ (just another $y$)
  - Advantage: everything is a constraint

This is a bad idea!
On unboundedness and early termination

- A ray $\chi$ of $X$: $x \in X \implies x + \lambda \chi \in X$ for $\lambda \to \infty \implies (c - yA)\chi > 0 \implies f(y) = \infty \implies$ constraint $cr \leq y(A\chi)$ in the dual

- One might even hide the convexity constraint:
  - $A\bar{x} \to [A\bar{x}, 1], \quad b \to [b, 1]$;
  - Ignoring the special role of $v$ (just another $y$)
  - Advantage: everything is a constraint

\[\text{This is a bad idea!}\]

- Moving $\bar{y}$ requires testing for decrease in $f$-value, but when a ray is generated, $f(\bar{y} + d^*) = \infty$

- Ignoring convexity constraint $\implies$ Proximal Point: solve the problem exactly for $\bar{y}$ before moving it

- Convexity constraints are good: invent them if they are not there
A Glimpse to Computational Results

- **State-of-the-art GenCol code**, large-scale, difficult MDVS instances (only root relaxation times)

- 5-pieces better than 3-pieces, **5-then-3 even better**

- Quadratic more “stable”, but optimized 5-pieces always faster (quadratic has far less parameters, easier but less flexible)

- Comparing 5-piecewise with (BP) or without (PP) early termination

<table>
<thead>
<tr>
<th></th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>p8</th>
<th>p9</th>
<th>p10</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>CG</td>
<td>139</td>
<td>177</td>
<td>235</td>
<td>159</td>
<td>3138</td>
<td>3966</td>
<td>3704</td>
<td>1742</td>
<td>3685</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td>33</td>
<td>36</td>
<td>38</td>
<td>28</td>
<td>482</td>
<td>335</td>
<td>946</td>
<td>572</td>
<td>1065</td>
</tr>
<tr>
<td></td>
<td>BP</td>
<td>26</td>
<td>28</td>
<td>35</td>
<td>21</td>
<td>295</td>
<td>257</td>
<td>639</td>
<td>352</td>
<td>545</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>p8</th>
<th>p9</th>
<th>p10</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter</td>
<td>CG</td>
<td>117</td>
<td>149</td>
<td>200</td>
<td>165</td>
<td>408</td>
<td>524</td>
<td>296</td>
<td>186</td>
<td>246</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td>47</td>
<td>47</td>
<td>49</td>
<td>45</td>
<td>93</td>
<td>64</td>
<td>98</td>
<td>83</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>BP</td>
<td>37</td>
<td>43</td>
<td>44</td>
<td>36</td>
<td>57</td>
<td>53</td>
<td>59</td>
<td>49</td>
<td>51</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>p8</th>
<th>p9</th>
<th>p10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpt</td>
<td>CG</td>
<td>88</td>
<td>125</td>
<td>165</td>
<td>105</td>
<td>1679</td>
<td>2004</td>
<td>1955</td>
<td>925</td>
<td>1984</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td>13</td>
<td>16</td>
<td>17</td>
<td>10</td>
<td>189</td>
<td>128</td>
<td>428</td>
<td>257</td>
<td>542</td>
</tr>
<tr>
<td></td>
<td>BP</td>
<td>10</td>
<td>14</td>
<td>15</td>
<td>10</td>
<td>100</td>
<td>70</td>
<td>329</td>
<td>206</td>
<td>334</td>
</tr>
</tbody>
</table>

- Stabilization works well, approximate stabilization works better
Other Forms of Stabilization

- **Proximal level**\(^{[41]}\): closest point promising given amount of decrease

\[
(\Delta_B, \bar{y}, l) \min \left\{ \frac{1}{2t} \| d \|_2^2 : f_B(\bar{y} + d) \leq f(\bar{y}) - l \right\}
\]  \(2\)

- \(l\) somehow easier to manage than \(t\), easy rules available that allow to keep \(\bar{y}\) fixed (but possible in proximal, too)

- Trade blows in practice, but **doubly-stabilised** possible\(^{[42]}\)

- Different approach: aim for **center** (analytic\(^{[43]}\) or Chebychev\(^{[44]}\)) of localization set \(L = \{(y, v) : f_B(y) \leq v \leq f(\bar{y})\} \subset \mathbb{R}^{n+1}\)

- “Good” theoretical performances, but in practice a penalty term is still required\(^{[45]}\)

---

\(^{[43]}\) Gondzio, González-Brevis, Munari “New Developments in the Primal-Dual Column Generation Technique” *EJOR*, 2013  
From Minimally to Maximally Intrusive Stabilization

- Changing the master problem not strictly needed: In-Out approach\textsuperscript{[46]} computes un-stabilised $d^*$ but probes $f(\bar{y} + \alpha d^*)$, $\alpha \in (0, 1]$
- Simple to implement and can still work well in practice\textsuperscript{[47]}
- Other extreme: $\mathcal{D}(d) = d^T Q d$, $Q = \text{“approximation of } \nabla^2 f(\bar{y})\text{”}$ (?!?!?) a-la quasi-Newton
- Theory exists, superlinear convergence possible\textsuperscript{[48]}
- Hard to make work in practice, but simpler scalings seem to work\textsuperscript{[49]}
- Many nice ideas\textsuperscript{[50]} if you like the research line
- Do work in practice but parameters ($t$, $l$, $\alpha$, \ldots) tuning still an art more than a science

\textsuperscript{[47]} Pessoa, Sadykov, Uchoa, Vanderbeck “Automation [\ldots] of [\ldots] Stabilization [\ldots] in Column Generation” \textit{IJoC}, 2018
\textsuperscript{[49]} Helmberg, Pichler “Dynamic Scaling and Submodel Selection in Bundle Methods for Convex Optimization” \textit{OO}, 2017
\textsuperscript{[50]} F. “Standard Bundle Methods: Untrusted Models and Duality” \textit{Numerical Nonsmooth Optimization}, 2020
Stabilized Benders’ Decomposition

- Stabilized master problem easy to do: with trust region
  \((B_B, \bar{x}, t)\) \(\min \left\{ v_B(x) : \| x - \bar{x} \|_\infty \leq t, x \in X \right\}\)
  pretty identical to (1) (no dual, though)

- For \(X \subseteq \{0, 1\}^n\), local branching constraint
  \(\sum_{i : \bar{x}_i = 1}(1 - x_i) + \sum_{i : \bar{x}_i = 0} x_i \leq t\)

- However, \(x^* = \bar{x}\) only \(\implies\) \(\bar{x}\) local optimum (nonconvex)
  \(\implies\) have to increase \(t\) until \(t = n (\infty)\)

- Silver lining: reverse box \(\| x - \bar{x} \|_\infty \geq t\) (nonconvex) now easy

- Level stabilization a-la (2) also possible\(^{[51]}\), pros and cons:
  \((B_B, \bar{x}, l)\) can be solved inexact\(ly\) (but larger and more difficult),
  \(l\) easier to manage than \(t\) and need not go \(\infty\) (but no reverse box)

- All in all it does work\(^{[52]}\) (but nontrivial)

---

\(^{[51]}\) van Ackooij, F., de Oliveira “Inexact Stabilized Benders’ Decomposition Approaches […]” COAP, 2016

\(^{[52]}\) Baena, Castro, F. “Stabilized Benders Methods for Large-scale Combinatorial Optimization […]” Man. Sci., 2020
Dual-Optimal Cuts
Dual-Optimal Cuts

- Stabilizing $\equiv$ restricting the dual space

- The above approaches need stability center $\tilde{y}$, to be updated: it’d be nice if we could do without

- Simple observation: dual constraints $\equiv$ primal variables
  $\implies$ need to add even more variables to the primal

  ... in such a way that not all dual optimal solution are cut
Dual-Optimal Cuts

- Stabilizing = restricting the dual space

- The above approaches need stability center $\bar{y}$, to be updated: it’d be nice if we could do without

- Simple observation: dual constraints = primal variables
  $\implies$ need to add even more variables to the primal
  
  ... in such a way that not all dual optimal solution are cut

- Actually quite simple:
  the new variables must not add new primal solutions$^{[53]}$

---

Dual-Optimal Cuts for Multicommodity flows

- \( \mathcal{C} = \) directed circuits with one reversed arc (aggregated flow)

- Constraints become

\[
\sum_{p \in \mathcal{P} : (i,j) \in p} f_p + \sum_{c \in \mathcal{C} : (i,j) \in c} \pm f_c \leq u_{ij}
\]

where “−” if \((i,j)\) is reversed in \(c\); hence, one also needs

\[
0 \leq \sum_{p \in \mathcal{P} : (i,j) \in p} f_p + \sum_{c \in \mathcal{C} : (i,j) \in c} \pm f_c
\]

- Any feasible solution to the extended model can be converted into a feasible solution to the original model
Dual-Optimal Cuts for Multicommodity flows

- $\mathcal{C} =$ directed circuits with one reversed arc (aggregated flow)

- Constraints become

$$\sum_{p \in P : (i,j) \in p} f_p + \sum_{c \in \mathcal{C} : (i,j) \in c} \pm f_c \leq u_{ij}$$

where “−” if $(i,j)$ is reversed in $c$; hence, one also needs

$$0 \leq \sum_{p \in P : (i,j) \in p} f_p + \sum_{c \in \mathcal{C} : (i,j) \in c} \pm f_c$$

- Any feasible solution to the extended model can be converted into a feasible solution to the original model

- $|\mathcal{C}| \in O(n^2)$ if $G$ is planar, all-pairs SPT pricing otherwise

- Some good results, other applications (Cutting Stock, different cuts)
Cuts Selection
(Feasibility) Cuts Selection

- \( v(x) = -\infty \implies \text{any } \bar{w} \in W_\infty \text{ gives a cut: which one is "best"?} \)
- If LP solver chooses, can’t expect it to pick a “good one”
- \( (x^*, v^*) \) solution of \( B_B \): a cut does not \( \exists \iff v(x^*) = \max \{ ez : Ez \leq d - Dx \} \geq v^* \)
  \[ \equiv \max \{ 0z : ez^* \geq v^* , Ez^* \leq d - Dx^* \} = 0 \]
- Hence a cut \( \exists \iff \min \{ w(d - Dx^*) - w_0 v^* : wE = w_0 e , (w , w_0) \geq 0 \} = -\infty \]
  \[ \equiv \text{(homogeneity)} \]
  \[ 0 > \min w(d - Dx^*) - w_0 v^* \]
  \[ wE = w_0 e , w\beta + w_0\beta_0 = 1 , (w , w_0) \geq 0 \]
  however chosen \((\beta , \beta_0)\): a proper choice improves performances[54]

Disaggregated Model
The real decomposition case:

\[(\Pi) \max \left\{ \sum_{k \in K} c^k x^k : \sum_{k \in K} A^k x^k = b, \quad x^k \in X^k, \quad k \in K \right\} \]

i.e., \(\bar{x} = [\bar{x}^k]_{k \in K}\) (Cartesian product of individual solutions)

Disaggregated DW reformulation:

\[(\Pi)\left\{ \max \sum_{k \in K} c^k \left( \sum_{\bar{x}^k \in X^k} \bar{x}^k \theta^k_\bar{x} \right) \right.\]
\[
\left. \sum_{k \in K} A^k \left( \sum_{\bar{x}^k \in X^k} \bar{x}^k \theta^k_\bar{x} \right) = b \right.\]
\[
\sum_{\bar{x}^k \in X^k} \theta^k_\bar{x} = 1 \quad k \in K
\]
\[
\theta^k_\bar{x} \geq 0 \quad \bar{x}^k \in X^k, \quad k \in K
\]

i.e., \(X = X^1 \times X^2 \times \ldots \times X^{|K|} \implies\)

\(\text{conv}(X) = \text{conv}(X^1) \times \text{conv}(X^2) \times \ldots \times \text{conv}(X^{|K|})\)

A different multiplier \(\theta^k_\bar{x}\) for each \(k \in K\): aggregated is \(\theta^k_\bar{x} = \theta^h_\bar{x}\) for \(h \neq k \implies\) a restriction (less solutions \(\equiv\) bad)
Geometry of Disaggregated Models

Given \( X \),

\[
X = \bigcup_{i=1}^{n} X_i
\]

From the dual viewpoint

\[
f_B(y) = \sum_{k \in K} f_k B(y)
\]

the sum of individual models is better than the model of the sum.
Geometry of Disaggregated Models

Given $X$, taking the convex hull of Cartesian products is smaller (bad) than first making convex hulls and then taking the Cartesian product.

From the dual viewpoint $f_B(y) = \sum_{k \in K} f_k B(y)$, the sum of individual models is better than the model of the sum.

- Given $X$, taking the convex hull of Cartesian products
Geometry of Disaggregated Models

- Given $X$, taking the convex hull of Cartesian products is smaller (bad) than first making convex hulls.
Geometry of Disaggregated Models

- Given $X$, taking the convex hull of Cartesian products is smaller (bad) than first making convex hulls and then taking the Cartesian product.
Given $X$, taking the convex hull of Cartesian products is smaller (bad) than first making convex hulls and then taking the Cartesian product.

From the dual viewpoint

$$f_B(y) = \sum_{k \in K} f^k_B(y)$$

the sum of individual models is better than the model of the sum.
Disaggregated Dantzig-Wolfe and Multicommodity Flows

- **Aggregated DW:** \( S = \{ \text{(extreme) flows } s = [\bar{x}^1, s, \ldots, \bar{x}^K, s] \} \)

  \[
  \min \sum_{s \in S} \left( \sum_{k \in K} \sum_{(i, j) \in A} c_{ij}^k \bar{x}_{ij}^k, s \right) \theta_s \\
  \sum_{s \in S} \left( \sum_{k \in K} \bar{x}_{ij}^k, s - u_{ij} \right) \theta_s \leq 0 \quad (i, j) \in A \\
  \sum_{s \in S} \theta_s = 1, \quad \theta_s \geq 0 \quad s \in S
  \]

- **Disaggregated + scaling ≡ arc-path formulation:**

  \( p \in \mathcal{P}^k = \{ s^k - t^k \text{ paths } \}, \ c_p \text{ cost, } f_p (= d^k \theta_s^k) \text{ flow, } \mathcal{P} = \bigcup_{k \in K} \mathcal{P}^k \)

  \[
  \min \sum_{p \in \mathcal{P}} c_p f_p \\
  \sum_{p \in \mathcal{P}} : (i, j) \in p \ f_p \leq u_{ij} \quad (i, j) \in A \\
  \sum_{p \in \mathcal{P}^k} f_p = d^k \quad k \in K \\
  f_p \geq 0 \quad p \in \mathcal{P}
  \]

- **More columns but sparser, (a few) more rows, much more efficient**\(^{[55]}\)

- **Master problem size ≈ time increases, but convergence speed more so**

---

Disaggregated decomposition

- Easily extended to any decomposable $X^{[15]}$
- Stabilized versions immediate
More or Less Disaggregated?

- That was $\approx 30$ years ago with $|K| \approx 10$, still true if $|K| \approx 10000$?

- Aggregation is arbitrary, then why “all or nothing”?

- Partition $C = (C_1, C_2, \ldots, C_h)$ of $K$, partially aggregated model $f^C_B$ with $h$ components $f^i_B$, each the sum over one $C_i$

- Basically, $\theta^k_s = \theta^h_s$ only for each $(h, k) \in C_i \times C_i$

- Exploring the trade-off between master problem size $\Rightarrow$ time and iterations, subproblems remain the same

- How to choose the $C_i$? In general open problem

- Aggregation can be dynamic\[56\], even more open problem, but it can work\[49\]

---

\[56\] van Ackooij, F. “Incremental Bundle Methods Using Upper Models” *SIOPT*, 2018
Easy Components
Multicommodity flow + arc design costs \( f_{ij} (z_{ij} \in \{0, 1\}) \)

\( S = \) extreme points of \( z \) (\( 2^{|A|} \) vertices of the unitary hypercube):

\[
\begin{align*}
\text{min} & \quad \sum_{p \in P} c_p f_p + \sum_{s \in S} \left( \sum_{(i,j) \in A} f_{ij} \bar{z}_{ij}^s \right) \theta_s \\
\sum_{p \in P : (i,j) \in p} f_p & \leq u_{ij} \sum_{s \in S} \bar{z}_{ij}^s \theta_s \quad (i,j) \in A \\
\sum_{p \in P^k} f_p &= d^k \quad k \in K \\
f_p & \geq 0 \quad p \in P \\
\sum_{s \in S} \theta_s &= 1, \quad \theta_s \geq 0 \quad s \in S
\end{align*}
\]

Are you sure you’re sane? Arguably not:

replacing a 2\( n \) formulation with a 2\( n \) one!

... and with very long, dense rows

A. Frangioni (DI — UniPi) Practical Decomposition Methods III&IV “Napoli” 2021 27 / 65
The unitary hypercube is a cartesian product: why not $S^{ij} = \{0, 1\}$?

$z_{ij} \rightarrow 0 \cdot \theta^{ij,0} + 1 \cdot \theta^{ij,1}$, $\theta^{ij,0} + \theta^{ij,1} = 1$, $\theta^{ij,0} \geq 0$, $\theta^{ij,1} \geq 0$.

$z_{ij} \in [0, 1]$
The unitary hypercube is a cartesian product: why not $S^{ij} = \{0, 1\}$?

$z_{ij} \rightarrow 0 \cdot \theta^{ij,0} + 1 \cdot \theta^{ij,1}$, $\theta^{ij,0} + \theta^{ij,1} = 1$, $\theta^{ij,0} \geq 0$, $\theta^{ij,1} \geq 0$.

$z_{ij} \in [0, 1]$ (no, ... really?!)

Arc-path formulation with original arc design variables

$$\begin{align*}
\min & \quad \sum_{p \in \mathcal{P}} c_p f_p + \sum_{(i,j) \in A} f_{ij} z_{ij} \\
\text{s.t.} & \quad \sum_{p \in \mathcal{P}} : (i,j) \in p f_p \leq u_{ij} z_{ij} \quad (i,j) \in A \\
& \quad \sum_{p \in \mathcal{P}^k} f_p = d^k \quad k \in K \\
& \quad f_p \geq 0 \quad p \in \mathcal{P} \\
& \quad z_{ij} \in [0, 1] \quad (i,j) \in A
\end{align*}$$

Only generate the right variables
Is it always this easy?

- **No:** what if one had, say,

\[ \sum_{(i,j) \in A} z_{ij} \leq r \quad ? \]

- Design subproblem can no longer be disaggregated

- But, one **could write the arc-path formulation** in that case, too

- And **could add that constraint to the master problem**

- Can this be stabilized? Of course it can\(^{[57]}\)

---

Stabilized decomposition with “easy components”

- \( f \) Lagrangian function of structured optimization problem

\[
(\Pi) \quad \max \left\{ c_1 x_1 + c_2(x_2) : x_1 \in X^1, \ G(x_2) \leq g, \ A_1 x_1 + A_2 x_2 = b \right\}
\]

i.e., \( f(y) = f^1(y) + f^2(y)(-yb) \) where

\[
f^1(\bar{y}) = \max \left\{ (c_1 - \bar{y}A_1)x_1 : x_1 \in X^1 \right\}
\]

“easy for some reason” (efficient but “totally obscure” black box)

\[
f^2(\bar{y}) = \max \left\{ c_2(x_2) - (\bar{y}A_2)x_2 : G(x_2) \leq g \right\}
\]

“easy because a compact convex formulation is known”
Stabilized decomposition with “easy components”

- $f$ Lagrangian function of structured optimization problem

\[(\Pi) \quad \max \left\{ c_1 x_1 + c_2(x_2) : x_1 \in X^1, \ G(x_2) \leq g, \ A_1 x_1 + A_2 x_2 = b \right\}\]

i.e., $f(y) = f^1(y) + f^2(y)(-yb)$ where

$$f^1(\bar{y}) = \max \left\{ (c_1 - \bar{y}A_1)x_1 : x_1 \in X^1 \right\}$$

“easy for some reason” (efficient but “totally obscure” black box)

$$f^2(\bar{y}) = \max \left\{ c_2(x_2) - (\bar{y}A_2)x_2 : G(x_2) \leq g \right\}$$

“easy because a compact convex formulation is known”

- Usual approach: disregard differences
  Better idea: treat “easy” components specially

- In practice: insert “full” description of $f^2$ in the master problem

- Master problem size may increase (at the beginning), but
  “perfect” information is known
“Easy components” in formulæ

- Dual master problem: abstract form

\[(\Delta_{\mathcal{B}, \bar{y}, \mathcal{D}}) \min \{ b(\bar{y} + d) + f^{1}_{\mathcal{B}}(\bar{y} + d) + f^{2}(\bar{x} + d) + \mathcal{D}(d) \} \]

- Primal master problem: abstract and implementable form

\[(\Pi_{\mathcal{B}, \bar{y}, \mathcal{D}}) \begin{cases} 
    c_{1}x_{1} + c_{2}(x_{2}) + \bar{y}s - \mathcal{D}^*(-s) \\
    s = b - A_{1}x_{1} - A_{2}x_{2} \\
    x_{1} \in \text{conv}(\mathcal{B}), \quad x_{2} \in X^{2} \\
    c_{1} \left( \sum_{\bar{x}_{1} \in \mathcal{B}} \bar{x}_{1} \theta_{\bar{x}_{1}} \right) + c_{2}(x_{2}) + \bar{y}s - \mathcal{D}^*(-s) \\
    s = b - A_{1} \left( \sum_{\bar{x}_{1} \in \mathcal{B}} \bar{x}_{1} \theta_{\bar{x}_{1}} \right) - A_{2}x_{2} \\
    \sum_{\bar{x}_{1} \in \mathcal{B}} \theta_{\bar{x}_{1}} = 1, \quad G(x_{2}) \leq g
\end{cases} \]

- Barring some details (do not translate \(f^{1}_{\mathcal{B}}\)), everything works

- Performances can improve dramatically (not hard to see why)
## A Glimpse to Computational Results

<table>
<thead>
<tr>
<th>primal</th>
<th>Cplex dual</th>
<th>net.</th>
<th>barr.</th>
<th>DE 1e-6</th>
<th>DE 1e-12</th>
<th>FA-2 time</th>
<th>f add</th>
<th>it</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>10</td>
<td>11</td>
<td>15</td>
<td>32</td>
<td>64</td>
<td>410</td>
<td>12</td>
<td>7</td>
<td>14880</td>
</tr>
<tr>
<td>64</td>
<td>53</td>
<td>61</td>
<td>71</td>
<td>48</td>
<td>51</td>
<td>1855</td>
<td>19</td>
<td>16</td>
<td>11141</td>
</tr>
<tr>
<td>139</td>
<td>114</td>
<td>132</td>
<td>157</td>
<td>29</td>
<td>29</td>
<td>1254</td>
<td>32</td>
<td>20</td>
<td>9035</td>
</tr>
<tr>
<td>559</td>
<td>456</td>
<td>531</td>
<td>587</td>
<td>65</td>
<td>66</td>
<td>1732</td>
<td>100</td>
<td>67</td>
<td>12940</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>primal</th>
<th>Cplex dual</th>
<th>net.</th>
<th>barr.</th>
<th>DE 1e-6</th>
<th>DE 1e-12</th>
<th>FA-2 time</th>
<th>f add</th>
<th>it</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>39</td>
<td>43</td>
<td>60</td>
<td>26</td>
<td>32</td>
<td>322</td>
<td>12</td>
<td>10</td>
<td>10320</td>
</tr>
<tr>
<td>147</td>
<td>132</td>
<td>144</td>
<td>209</td>
<td>28</td>
<td>56</td>
<td>294</td>
<td>15</td>
<td>9</td>
<td>5300</td>
</tr>
<tr>
<td>509</td>
<td>301</td>
<td>478</td>
<td>648</td>
<td>21</td>
<td>26</td>
<td>5033</td>
<td>169</td>
<td>155</td>
<td>27231</td>
</tr>
<tr>
<td>2329</td>
<td>1930</td>
<td>2302</td>
<td>2590</td>
<td>133</td>
<td>133</td>
<td>3122</td>
<td>192</td>
<td>169</td>
<td>14547</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>primal</th>
<th>Cplex dual</th>
<th>net.</th>
<th>barr.</th>
<th>DE 1e-6</th>
<th>DE 1e-12</th>
<th>FA-2 time</th>
<th>f add</th>
<th>it</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>196</td>
<td>131</td>
<td>156</td>
<td>304</td>
<td>2</td>
<td>3</td>
<td>344</td>
<td>20</td>
<td>12</td>
<td>7169</td>
</tr>
<tr>
<td>926</td>
<td>708</td>
<td>862</td>
<td>1174</td>
<td>246</td>
<td>337</td>
<td>2256</td>
<td>111</td>
<td>118</td>
<td>17034</td>
</tr>
<tr>
<td>2706</td>
<td>2167</td>
<td>2542</td>
<td>3272</td>
<td>284</td>
<td>508</td>
<td>5475</td>
<td>192</td>
<td>249</td>
<td>15061</td>
</tr>
<tr>
<td>11156</td>
<td>8908</td>
<td>11675</td>
<td>11683</td>
<td>242</td>
<td>253</td>
<td>11863</td>
<td>349</td>
<td>413</td>
<td>13953</td>
</tr>
</tbody>
</table>

- **Fa-V** = subgradient, **FA-2** = aggregated, ad-hoc \((\Delta_{B,\bar{y},t})\) solver\(^{[58]}\)
- **Tuning not easy**, a lot of pieces have to click\(^{[57]}\)
- **Much faster than Cplex and anything else** as \(|A|\) and/or \(|K|\) grows

The Easy Component Need Not Be Linear

- **Nonlinear** multicommodity routing:

\[
\min \left\{ \sum_{(i,j) \in A} \frac{z_{ij}}{1-z_{ij}} : \langle \text{multicommodity flow} \rangle, \ z \in [0, 1]^{|A|} \right\}
\]

with classical (convex) **Kleinrock delay function**

- Decomposes into \(|K|\) flows + \(|A|\) simple convex subproblems

- **Specialized models** of \(|A|\) convex functions using the conjugate

- **Specialized treatment** of these “easy” \(C^2\) functions with **Newton model instead of the cutting-plane model**[59]

- Substantially improved performances

[59] Lemaréchal, Ororou, Petrou “A Bundle-type Algorithm for Routing in Telecommunication Data Networks” *COAP*, 2009
Structured Decomposition
The Structured Dantzig-Wolfe Idea

- **Assumption 1**: Alternative Formulation of "easy" set

\[
\text{conv}(X) = \{ \ x = C\theta : \Gamma\theta \leq \gamma \ \}
\]
The Structured Dantzig-Wolfe Idea

- **Assumption 1**: Alternative Formulation of "easy" set
  \[
  \text{conv}(X) = \{ x = C\theta : \Gamma\theta \leq \gamma \}
  \]

- **Assumption 2**: padding with zeroes
  \[
  \Gamma_B\tilde{\theta}_B \leq \gamma_B \Rightarrow \Gamma[\tilde{\theta}_B, 0] \leq \gamma
  \]
  \[
  \Rightarrow X_B = \{ x = C_B\theta_B : \Gamma_B\theta_B \leq \gamma_B \} \subseteq \text{conv}(X)
  \]
The Structured Dantzig-Wolfe Idea

- **Assumption 1**: Alternative Formulation of “easy” set
  \[ \text{conv}(X) = \{ x = C\theta : \Gamma\theta \leq \gamma \} \]

- **Assumption 2**: padding with zeroes
  \[ \Gamma_{B} \bar{\theta}_{B} \leq \gamma_{B} \Rightarrow \Gamma[\bar{\theta}_{B}, 0] \leq \gamma \]
  \[ \Rightarrow X_{B} = \{ x = C_{B}\theta_{B} : \Gamma_{B}\theta_{B} \leq \gamma_{B} \} \subseteq \text{conv}(X) \]

- **Assumption 3**: easy update of rows and columns
  Given \( B, \bar{x} \in \text{conv}(X), \bar{x} \notin X_{B} \), it is “easy” to find \( B' \supset B \)
  \[ (\Rightarrow \Gamma_{B'}, \gamma_{B'}) \text{ such that } \exists B'' \supseteq B' \text{ such that } \bar{x} \in X_{B''}. \]
The Structured Dantzig-Wolfe Idea

- **Assumption 1**: Alternative Formulation of “easy” set
  \[
  \text{conv}(X) = \{ x = C\theta : \Gamma\theta \leq \gamma \}
  \]

- **Assumption 2**: padding with zeroes
  \[
  \Gamma_B\bar{\theta}_B \leq \gamma_B \Rightarrow \Gamma[\bar{\theta}_B, 0] \leq \gamma
  \]
  \[
  \Rightarrow X_B = \{ x = C_B\theta_B : \Gamma_B\theta_B \leq \gamma_B \} \subseteq \text{conv}(X)
  \]

- **Assumption 3**: easy update of rows and columns
  Given \( \mathcal{B}, \bar{x} \in \text{conv}(X), \bar{x} \notin X_B, \) it is “easy” to find \( \mathcal{B}' \supset \mathcal{B} \)
  \( (\Rightarrow \Gamma_{B'}, \gamma_{B'}) \) such that \( \exists \mathcal{B}'' \supset \mathcal{B}' \) such that \( \bar{x} \in X_{B''} \).

- **Structured master problem**
  \[
  (\Pi_B) \quad \max \{ c x : A x = b, \ x = C_B\theta_B, \ \Gamma_B\theta_B \leq \gamma_B \} \quad (3)
  \]
  \( \equiv \) structured model
  \[
  f_B(y) = \max\{ (c - yA)x + xb : x = C_B\theta_B, \ \Gamma_B\theta_B \leq \gamma_B \} \quad (4)
  \]
The Structured Dantzig-Wolfe Algorithm

\[ \langle \text{initialize } B \rangle; \]
repeat
\[ \langle \text{solve } (\Pi_B) \text{ for } x^*, y^* \text{ (duals of } Ax = b); \quad v^* = cx^* \rangle; \]
\[ \bar{x} = \text{argmin} \{ (c - y^* A)x : x \in X \}; \]
\[ \langle \text{update } B \text{ as in Assumption 3 } \rangle; \]
until \( v^* < c\bar{x} + y^*(b - A\bar{x}) \)
The Structured Dantzig-Wolfe Algorithm

\[
\begin{align*}
\langle \text{initialize } B \rangle; \\
\text{repeat} \\
\langle \text{solve } (\Pi_B) \text{ for } x^*, y^* \text{ (duals of } Ax = b); \nu^* = cx^* \rangle; \\
\bar{x} = \text{argmin} \{ (c - y^* A)x : x \in X \}; \\
\langle \text{update } B \text{ as in Assumption 3} \rangle; \\
\text{until } \nu^* < c\bar{x} + y^*(b - A\bar{x})
\end{align*}
\]

- Easy\cite{60} to prove that:
  - finitely terminates with an optimal solution of \((\Pi)\)
  - \ldots even if (proper) removal from \(B\) is allowed (when \(cx^*\) increases)
  - \ldots even if \(X\) is non compact and \(B = \emptyset\) at start (Phase 0)
The Structured Dantzig-Wolfe Algorithm

\[
\langle \text{initialize } \mathcal{B} \rangle;
\]
repeat
\[
\langle \text{solve } (\Pi_\mathcal{B}) \text{ for } x^*, y^* \text{ (duals of } Ax = b); v^* = cx^* \rangle;
\]
\[
\bar{x} = \arg\min \{ (c - y^* A)x : x \in X \};
\]
\[
\langle \text{update } \mathcal{B} \text{ as in Assumption 3} \rangle;
\]
until \( v^* < c\bar{x} + y^*(b - A\bar{x}) \)

- Easy\(^{[60]}\) to prove that:
  - finitely terminates with an optimal solution of (\(\Pi\))
  - . . . even if (proper) removal from \(\mathcal{B}\) is allowed (when \(cx^*\) increases)
  - . . . even if \(X\) is non compact and \(\mathcal{B} = \emptyset\) at start (Phase 0)
- The subproblem to be solved is identical to that of DW
- Requires (\(\implies\) exploits) extra information on the structure
- Master problem with any structure, possibly much larger

\(^{[60]}\) F., Gendron “0-1 reformulations of the multicommodity capacitated network design problem” DAM, 2009
Stabilizing the Structured Dantzig-Wolfe Algorithm

- Exactly the same as stabilizing DW: stabilized master problem
  \[(\Delta_B, \bar{y}, D) \quad \min \{ f_B(\bar{y} + d) + D(d) \}\]
  except \(f_B\) is a different model of \(f\) (not the cutting plane one)

- Even simpler from the primal viewpoint:
  \[
  \max \{ cx + \bar{y}s - D^*(-s) : s = b - Ax, x = C_B\theta_B, \Gamma_B\theta_B \leq \gamma_B \}
  \]

- With proper choice of \(D\), still a Linear Program; e.g.
  \[
  \max \ldots - (\Delta^- + \Gamma^-)s'' - \Delta^- s' - \Delta^+ s' - (\Delta^+ + \Gamma^+)s''
  
  s'' + s' - s' + s'' = b - Ax, \ldots
  
  s'' \geq 0, \varepsilon^+ \geq s' \geq 0, \varepsilon^- \geq s'' \geq 0, \quad s'' \geq 0
  
  dual optimal variables of “\(s = b - Ax\)” still give \(d^*\), \ldots
  \]

- Move \(\bar{y}\), handle \(t\), handle \(B\): as in [37] (or simpler, \(B\) is finite)

- Even better computational results in the right application[61]

Incremental, Inexact, Asynchronous
Incremental Computation of Subproblems

- (Partial) aggregation can contribute to reducing master problem cost but subproblem cost remains the same.
- Subproblem cost high if $|K|$ large and/or subproblems hard, trade-off very application-dependent (you get to meet all sorts).
- Clearly interesting to avoid “unnecessary” subproblems computations.
- In fact quite easy to understand early on if $f(\bar{y} + d^*) \ll f(\bar{y})$ “null steps” can be declared without computing all subproblems\(^{[62]}\).
- Early declaring “serious steps” harder, but possible\(^{[56]}\) provided you can estimate the Lipchitz constant (nontrivial).
- Trade-off still all to explore.

Inexact Computation of Subproblems

- Turns out **incremental special case of inexact**: 
  \[ f(\bar{y} + d^*) \] only approximately computed
- Powerful general theory well-understood for proximal\(^{[63]}\) and level\(^{[64]}\)
- May require “**noise reduction steps**”: \(t/l\) changed without oracle calls (exploit stabilization to sample the space, like “curved search”\(^{[65]}\))
- Different noise reductions depending on oracle “unfaithfulness”\(^{[56]}\)
- **Explicitly provide upper/lower bounds and accuracy** to oracle\(^{[56]}\)
- Can significantly improve total running time, but:
  - details depend on stabilization employed
  - trade-off with number of iterations nontrivial


\(^{[64]}\) de Oliveira, Sagastizábal “Level Bundle Methods for Oracles With On-Demand Accuracy” *OM&S*, 2014

\(^{[65]}\) Schramm, Zowe.“A Version of the Bundle Idea for Minimizing a Nonsmooth Function […]” *SIOPT*, 1992
Asynchronous Computation of Subproblems

- Clear avenue to reduce wall-clock time: parallelize subproblems
- Master-slave version “obvious”\cite{ref1}, popular for stochastic programs\cite{ref2}
- Runs afoul of \textit{Amdahl’s Law}: speedup limited by master problem cost and large master problems is what works best (most often)
- May use specialised algorithms\cite{ref3} and hardware\cite{ref4}, but issue remains
- \textbf{Completely asynchronous versions possible}\cite{ref5}
- Still to be completed (proximal? multiple masters?), general efficient implementations highly nontrivial
- Interesting variants for “loosely coupled subproblems”\cite{ref6}

\begin{thebibliography}{9}
\bibitem{ref1} Lubin, Martin, Petra, Sandıkçı “On Parallelizing Dual Decomposition in Stochastic Integer Programming” \textit{O.R. Lett.}, 2013
\bibitem{ref2} Iutzeler, Malick, de Oliveira “Asynchronous Level Bundle Methods” \textit{Math. Prog.}, 2020
\bibitem{ref3} Fischer, Helmberg “A Parallel Bundle Framework for Asynchronous Subspace Optimisation [\ldots]” \textit{SIOPT}, 2014
\end{thebibliography}
Part IV:
A Useful Companion on the Road
Decomposition in Practice

- Decomposition is complex, but so is any Branch-and-X
- Need general-purpose efficient decomposition software:
  - Cplex does Benders’, structure automatic or user hints
  - SCIP\textsuperscript{[30]} does B&C&P (one-level D-W), pricing & reformulation up to the user (plugins)
  - GCG\textsuperscript{[30]} extends SCIP with automatic and user-defined (one-level) D-W and recently also a generic (one-level) Benders’ approach\textsuperscript{[69]}
  - D-W approaches for two-stage stochastic programs are implemented in DDSIP\textsuperscript{[70]} and PIPS\textsuperscript{[71]}, the latter interfaced with StructJuMP\textsuperscript{[72]}
  - The BaPCoD B&C&P code has been used to develop Coluna.jl\textsuperscript{[73]}, doing one-level D-W and (alpha) Benders’, multi-level planned

- 4 years ago there was no multi-level, nor C++, so we started one

---

\textsuperscript{[69]} Maher “Implementing the Branch-and-Cut approach for a general purpose Benders’ decomposition framework” \textit{EJOR}, 2021
\textsuperscript{[70]} \url{https://github.com/RalfGollmer/ddsip}
\textsuperscript{[71]} \url{https://github.com/Argonne-National-Laboratory/PIPS}
\textsuperscript{[72]} \url{https://github.com/StructJuMP/StructJuMP.jl}
\textsuperscript{[73]} \url{https://github.com/atoptima/Coluna.jl}
https://gitlab.com/smspp/smspp-project

Open source (LGPL3), public as of yesterday!
What SMS++ is

- A core set of C++-17 classes implementing a modelling system that:
  - explicitly supports the notion of Block ≡ nested structure
  - separately provides “semantic” information from “syntactic” details (list of constraints/variables ≡ one specific formulation among many)
  - allows exploiting specialised Solver on Block with specific structure
  - manages any dynamic change in the Block beyond “just” generation of constraints/variables
  - supports reformulation/restriction/relaxation of Block
  - has built-in parallel processing capabilities
  - should be able to deal with almost anything (bilevel, PDE, …)

- An hopefully growing set of specialized Block and Solver

- In perspective an ecosystem fostering collaboration and code sharing
What SMS++ is not

- **An algebraic modelling language**: Block / Solver are C++ code (although it provides some modelling-language-like functionalities)

- **For the faint of heart**: primarily written for algorithmic experts (although users may benefit from having many pre-defined Block)

- **Stable**: only version 0.4, lots of further development ahead, significant changes in interfaces not ruled out, actually expected (although current Block / Solver very thoroughly tested)

- **Interfaced with many solvers**: only Cplex, SCIP, MCFClass, StOpt (although the list should hopefully grow)
A Crude Schematic

Objective

Constraint

Variable

Modification

Solver

{ Modification }
**Block**

- **Block** = abstract class representing the general concept of “a (part of a) mathematical model with a well-understood identity”

- Each **:Block** a model with specific structure (e.g., MCFBlock::Block = a Min-Cost Flow problem)

- **Physical representation** of a Block: whatever data structure is required to describe the instance (e.g., $G$, $b$, $c$, $u$)

- **Possibly alternative abstract representation(s)** of a Block:
  - one Objective (but possibly vector-valued)
  - any # of groups of (static) Variable
  - any # of groups of std::list of (dynamic) Variable
  - any # of groups of (static) Constraint
  - any # of groups of std::list of (dynamic) Constraint
  - groups of Variable/Constraint can be single (std::list) or std::vector (...) or *boost::multi_array*

- **Any # of sub-Blocks** (recursively), possibly of specific type (e.g., Block::MMCFBlock has $k$ Block::MCFBlock inside)
Variable

- Abstract concept, thought to be extended (a matrix, a function, ...)
- Does not even have a value
- Knows which Block it belongs to
- Can be fixed and unfixed to/from its current value (whatever that is)
- Influences a set of Constraint/Objective/Function (actually, a set of ThinVarDepInterface)
- Fundamental design decision: “name” of a Variable = its memory address \(\implies\) copying a Variable makes a different Variable \(\implies\) dynamic Variables always live in std::lists
- VariableModification:Modification (fix/unfix)
Constraint

- Abstract concept, thought to be extended (any algebraic constraint, a matrix constraint, a PDE constraint, bilevel program, ...)

- **Depends from** a set of Variable (:ThinVarDepInterface)

- Either **satisfied** or not by the current value of the Variable, checking it possibly costly (:ThinComputeInterface)

- Knows which Block it belongs to

- Can be **relaxed** and **enforced**

- **Fundamental design decision:** “name” of a Constraint = its memory address $\implies$ copying a Constraint makes a different Constraint $\implies$ dynamic Constraints always live in std::lists

- **ConstraintModification**:Modification (relax/enforce)
Objective

- Abstract concept, does not specify its return value (vector, set, ...)
- Either minimized or maximized
- Depends from a set of Variable (:ThinVarDepInterface)
- Must be evaluated w.r.t. the current value of the Variable, possibly a costly operation (:ThinComputeInterface)
- RealObjective: Objective implements “value is an extended real”
- Knows which Block it belongs to
- Same fundamental design decision ...
  (but there is no such thing as a dynamic Objective)
- ObjectiveModification: Modification (change verse)
Function

- Real-valued Function
- Depends from a set of Variable (:ThinVarDepInterface)
- Must be evaluated w.r.t. the current value of the Variable, possibly a costly operation (:ThinComputeInterface)
- Approximate computation supported in a quite general way[^56] (since :ThinComputeInterface, and that does)
- FunctionModification[Variables] for “easy” changes → reoptimization (shift, adding/removing “quasi separable” Variable)
C05Function and C15Function

- C05Function/C15Function deal with 1\textsuperscript{st}/2\textsuperscript{nd} order information (not necessarily continuous)

- General concept of “linearization” (gradient, convex/concave subgradient, Clarke subgradient, . . .)

- Multiple linearizations produced at each evaluation (local pool)

- **Global pool of linearizations for reoptimization:**
  - convex combination of linearizations
  - “important linearization” (at optimality)

- C05FunctionModification[Variables/LinearizationShift] for “easy” changes $\rightarrow$ reoptimization (linearizations shift, some linearizations entries changing in simple ways)

- C15Function supports (partial) Hessians

- Arbitrary hierarchy of :Function possible/envisioned, any one that makes sense for application and/or solution method
Closer to the ground

- **ColVariable**: “value = one single real” (possibly $\in \mathbb{Z}$)

- **RowConstraint**: “$l \leq a \text{ real} \leq u$” $\implies$
  has dual variable (single real) attached to it

- **OneVarConstraint**: “a real” $=$
  a single **ColVariable** $\equiv$ bound constraints

- **FRowConstraint**: “a real” given by a **Function**

- **FRealObjective**: “value” given by a **Function**

- **LinearFunction**: a linear form in **ColVariable**

- **DQuadFunction**: a separable quadratic form

- **Many things missing** (AlgebraicFunction, DenseLinearFunction, Matrix/VectorVariable, ...)

A. Frangioni (DI — UniPi)  
Practical Decomposition Methods III&IV  
“Napoli” 2021  
50 / 65
Block and Solver

- Any # of Solver attached to a Block to solve it
- Solver for a specific :Block can use the physical representation → no need for explicit Constraint
  → abstract representation of Block only constructed on demand
- However, Variable are always present to interface with Solver (this may change thanks to methods factory)
- A general-purpose Solver uses the abstract representation
- Dynamic Variable/Constraint can be generated on demand (user cuts/lazy constraints/column generation)
- For a Solver attached to a Block:
  - Variable not belonging to the Block are constants
  - Constraint not belonging to the Block are ignored
  (belonging = declared there or in any sub-Block recursively)
- Objective of sub-Blocks summed to that of father Block if has same verse, otherwise min/max
Solver

- **Solver** = interface between a Block and algorithms solving it
- Each Solver attached to a single Block, from which it picks all the data, but any # of Solver can be attached to the same Block
- **Solutions are written directly into the Variable of the Block**
- Individual Solver can be attached to sub-Block of a Block
- Tries to cater for all the important needs:
  - optimal and sub-optimal solutions, provably unbounded/unfeasible
  - time/resource limits for solutions, but **restarts** (reoptimization)
  - any # of **multiple solutions** produced on demand
  - lazily reacts to changes in the data of the Block via **Modification**
- Slanted towards **RealObjective** (≈optimality = up/low bounds)
- **CDASolver:** Solver is “Convex Duality Aware”: **bounds are associated to dual solutions** (possibly, multiple)
- Provides **general events mechanism** (ThinComputeInterface does)
Block and Modification

- Most Block components can change, but not all:
  - set of sub-Block
  - # and shape of groups of Variable/Constraint

- Any change is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification object

- anyone_there() \(\equiv\) \(\exists\) interested Solver (Modification needed)

- However, two different kinds of Modification (what changes):
  - physical Modification, only specialized Solver concerned
  - abstract Modification, only Solver using it concerned

- Abstract Modification used to keep both representations in sync
  \(\implies\) a single change may trigger more than one Modification
  \(\implies\) concerns Block() mechanism to avoid this to repeat
  \(\implies\) parameter in changing methods to avoid useless Modification

- Specialized Solver disregard abstract Modification and vice-versa

- A Block may refuse to support some changes (explicitly declaring it)
Modification

- Almost empty base class, then everything has its own derived ones

- **Heavy stuff** can be attached to a Modification (e.g., added/deleted dynamic Variable/Constraint)

- Each Solver has the **responsibility** of cleaning up its list of Modification (smart pointers $\rightarrow$ memory eventually released)

- Solver supposedly **reoptimize** to improve efficiency, which is easier if you can see all list of changes at once (lazy update)

- **GroupModification** to (recursively) pack many Modification together $\Longrightarrow$ different “channels” in Block

- Modification **processed in the arrival order** to ensure consistency

- A Solver may optimize the changes (Modifications may cancel each outer out . . .), but its responsibility
Support to (coarse-grained) Parallel Computation

- Block can be (r/w) lock()-ed and read_lock()-ed
- lock()-ing a Block automatically lock()s all inner Block
- lock() (but not read_lock()) sets an owner and records its std::thread::id; other lock() from the same thread fail (std::mutex would not work there)
- Similar mechanism for read_lock(), any # of concurrent reads
- Write starvation not handled yet
- A Solver can be “lent an ID” (solving an inner Block)
- The list of Modification of Solver is under an “active guard” (std::atomic)
- Distributed computation under development, can exploit general serialize/deserialize Block capabilities, Cray/HPE “Fugu” framework
Solution

- Block produces Solution object, possibly using its sub-Blocks’
- Solution can read() its own Block and write() itself back
- Solution is Block-specific rather than Solver-specific
- Solution may save dual information
- Solution may save only a specific subset of primal/dual information
- **Linear combination** of Solution supported $\Rightarrow$ “less general”
  Solution may (automatically) convert in “more general” ones
- Like Block, Solution are tree-structured complex objects
- ColVariableSolution: Solution uses the abstract representation of any Block that only have (std::vector or boost::multi_array of) (std::list of) ColVariables to read/write the solution
- RowConstraintSolution: Solution same for dual information (RowConstraint), ColRowSolution for both
Configuration

- Block a tree-structured complex object \( \Rightarrow \) Configuration for them a (possibly) tree-structured complex object

- But also SimpleConfiguration\( \langle T \rangle : \text{Configuration} \)
  
  \( T \) an int, a double, a std::pair\( <> \), ...

- [C/O/R] BlockConfiguration:Configuration set [recursively]:
  - which dynamic Variable/Constraint are generated, how (Solver, time limit, parameters ...)
  - which Solution is produced (what is saved)
  - the ComputeConfiguration:Configuration of any Constraint/Objective that needs one
    - a bunch of other Block parameters

- [R] BlockSolverConfiguration:Configuration set [recursively] which Solver are attached to the Block and their ComputeConfiguration:Configuration

- Can be clear()-ed for cleanup
R³Block

- Often reformulation crucial, but also relaxation or restriction: get_R3_Block() produces one, possibly using sub-Blocks’

- Obvious special case: copy (clone) should always work

- Available R³Blocks :Block-specific, a :Configuration needed

- R³Block completely independent (new Variable/Constraint), useful for algorithmic purposes (branch, fix, solve, ...)

- Solution of R³Block useful to Solver for original Block: map_back_solution() (best effort in case of dynamic Variable)

- Sometimes keeping R³Block in sync with original necessary: map_forward_modification(), task of original Block

- map_forward_solution() and map_back_modification() useful, e.g., dynamic generation of Variable/Constraint in the R³Block

- :Block is in charge of all this, thus decides what it supports
A lot of other support stuff

- All **tree-structured complex objects** (Block, Configuration, ...) and Solver have an (almost) automatic **factory**

- All **tree-structured complex objects** (....) have methods to **serialize/deserialize** themselves to netCDF files

- A **methods factory** for changing the physical representation without knowing of which :Block it exactly is (standardised interface)

- AbstractBlock for constructing a model a-la algebraic language, can be derived for “general Block + specific part”

- PolyhedralFunction[Block], very useful for decomposition

- AbstractPath for indexing any Constraint/Variable in a Block

- FakeSolver:Solver stashes away all Modification, UpdateSolver:Solver immediately forwards/R^3 Bs them

- ...
Main Existing Blocks:

- **MCFBlock/MMCFBlock**: single/multicommodity flow (p.o.c.)

- **UCBlock** for UC, abstract UnitBlock with several concrete (ThermalUnitBlock, HydroUnitBlock, ...), abstract NetworkBlock with a few concrete (DCNetworkBlock)

- **LagBFunction**: \{C05Function, Block\} transforms any Block (with appropriate Objective) into its dual function

- **BendersBFunction**: \{C05Function, Block\} transforms any Block (with appropriate Constraint) into its value function

- **StochasticBlock** implements realizations of scenarios into any Block (using methods factory)

- **SDDPBlock** represents multi-stage stochastic programs suitable for Stochastic Dual Dynamic Programming
Main “Basic” Solver

- **MCFSolver**: templated p.o.c. wrapper to MCFClass\(^{[74]}\) for MCFBlock
- **DPSolver** for ThermalUnitBlock\(^{[12]}\) (still needs serious work)
- **MILPSolver**: constructs matrix-based representation of any “LP” Block: ColVariable, FRowConstraint, FRealObjective with LinearFunction or DQuadFunction
- **CPXMILPSolver**: MILPSolver and **SCIPMILPSolver**: MILPSolver wrappers for Cplex and SCIP (to be improved)
- **BundleSolver**: CDASolver: SMS++-native version of\(^{[75]}\) (still shares some code, dependency to be removed), optimizes any (sum of) C05Function, most (not all) state-of-the-art tricks
- **SDDPSolver**: wrapper for SDDP solver St0pt\(^{[76]}\) using StochasticBlock, BendersBFunction and PolyhedralFunction
- **SDDPGreedySolver**: greedy forward simulator for SDDPBlock

---

\(^{[74]}\) [https://github.com/frangio68/Min-Cost-Flow-Class](https://github.com/frangio68/Min-Cost-Flow-Class)

\(^{[75]}\) [https://gitlab.com/frangio68/ndosolver_fioracle_project](https://gitlab.com/frangio68/ndosolver_fioracle_project)

\(^{[76]}\) [https://gitlab.com/stochastic-control/St0pt](https://gitlab.com/stochastic-control/St0pt)
Our Masterpiece: LaGrangianDualSolver

- Works for any Block with natural block-diagonal structure: no Objective or Variable, all Constraint linking the inner Block
- Using LagBFunction stealthily constructs the LaGrangian Dual w.r.t. linking Constraint, $R^3B$-ing or “stealing” the inner Block
- Solves the LaGrangian Dual with appropriate CDASolver (e.g., but not necessarily, BundleSolver), provides dual and “convexified” solution in original Block
- Can attach LaGrangianDualSolver and (say) :MILPSolver to same Block, solve in parallel!
- Weeks of work in days/hours (if Block of the right form already)
- Hopefully soon BendersDecompositionSolver (crucial component BendersBFunctor existing and tested)
- Multilevel nested parallel heterogeneous decomposition by design (but I’ll believe it when I’ll see it running)
The many things that we do not have (yet)

- A relaxation-agnostic Branch-and-X Solver (could recycle OORB)
- Many other forms of Variable (Vector/MatrixVariable, FunctionVariable, ...), Constraint (AlgebraicFunction, BilevelConstraint, EquilibriumConstraint, PDEConstraint, ...) and/or Objective (RealVectorObjective, ...)
- Interfaces with many other solvers (OSISolverInterface, Couenne, OR-tools CP-SAT Solver, ...)
- Many many more :Block and their specialised :Solver

A. Frangioni (DI — UniPi)  Practical Decomposition Methods III&IV  “Napoli” 2021  63 / 65
The many things that we do not have (yet)

- A relaxation-agnostic Branch-and-X Solver (could recycle 00BB)
- Many other forms of Variable (Vector/MatrixVariable, FunctionVariable, ...), Constraint (AlgebraicFunction, BilevelConstraint, EquilibriumConstraint, PDEConstraint, ...) and/or Objective (RealVectorObjective, ...)
- Interfaces with many other solvers (OSISolverInterface, Couenne, OR-tools CP-SAT Solver, ...)
- Many many more: Block and their specialised: Solver
- Achieving critical mass crucial, decomposition not the only objective:
  - improve collaboration and code reuse, reduce huge code waste (I ♥ coding, breaks my ♥)
  - significantly increase the addressable market of decomposition
  - a much-needed step towards higher uptake of parallel methods
  - the missing marketplace for specialised solution methods
  - a step towards a reformulation-aware modelling system[77]

Conclusions
(for good, this time)
Conclusions and (a lot of) future work

- Decomposition methods (D-W, Benders’) old ideas, well-understood, but by-the-book decomposition often not effective enough
- Many nontrivial ideas to improve on the standard approaches
Conclusions and (a lot of) future work

- Decomposition methods (D-W, Benders’) old ideas, well-understood, but by-the-book decomposition often not effective enough
- Many nontrivial ideas to improve on the standard approaches
- Reduce iterations count: “large” master problems to quickly get the “combinatorial tail” \(\implies\)
  - large master problem time go against Amdhal’s Law
  - “unstructured” master problems \(\implies\) can’t use “easy” specialised methods\(^{[58]}\) (but there may be ways\(^{[78]}\), some structure is there)
- Reduce subproblems cost: incremental/inexact/asynchronous solution + all possible reoptimization
- In all cases, complex multilevel parallel implementation
Conclusions and (a lot of) future work

- Decomposition methods (D-W, Benders’) old ideas, well-understood, but by-the-book decomposition often not effective enough
- Many nontrivial ideas to improve on the standard approaches
- Reduce iterations count: “large” master problems to quickly get the “combinatorial tail” ➞
  - large master problem time go against Amdhal’s Law
  - “unstructured” master problems ➞ can’t use “easy” specialised methods\(^{[58]}\) (but there may be ways\(^{[78]}\), some structure is there)
- Reduce subproblems cost: incremental/inexact/asynchronous solution + all possible reoptimization
- In all cases, complex multilevel parallel implementation
- Some support slowly starting to show up
- SMS++ aiming for a slice of this cake

Conclusions and (a lot of) future work

- Decomposition methods (D-W, Benders’) old ideas, well-understood, but by-the-book decomposition often not effective enough
- Many nontrivial ideas to improve on the standard approaches
- Reduce iterations count: “large” master problems to quickly get the “combinatorial tail”
  - large master problem time go against Amdhal’s Law
  - “unstructured” master problems can’t use “easy” specialised methods[^58] (but there may be ways[^78], some structure is there)
- Reduce subproblems cost: incremental/inexact/asynchronous solution + all possible reoptimization
- In all cases, complex multilevel parallel implementation
- Some support slowly starting to show up
- SMS++ aiming for a slice of this cake (not a lie), and more

Conclusions and (a lot of) future work

- Decomposition methods (D-W, Benders’) old ideas, well-understood, but by-the-book decomposition often not effective enough

- Many nontrivial ideas to improve on the standard approaches

- Reduce iterations count: “large” master problems to quickly get the “combinatorial tail”\[
\implies
\]

  - large master problem time go against Amdhal’s Law
  - “unstructured” master problems \(\implies\) can’t use “easy” specialised methods\[^{58}\] (but there may be ways\[^{78}\], some structure is there)

- Reduce subproblems cost: incremental/inexact/asynchronous solution + all possible reoptimization

- In all cases, complex multilevel parallel implementation

- Some support slowly starting to show up

- SMS++ aiming for a slice of this cake (not a lie), and more

- Lots of fun to be had, all contributions welcome

Acknowledgements

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 773897

Copyright © Università di Pisa 2021, all rights reserved.

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the PLAN4RES Consortium. In addition, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

This document may change without notice.

The content of this document only reflects the author’s views. The European Commission / Innovation and Networks Executive Agency is not responsible for any use that may be made of the information it contains.