Separable Lagrangian decomposition for the Knapsack Relaxation of Multicommodity Network Design

Antonio Frangioni\textsuperscript{1} \hspace{0.5cm} Bernard Gendron\textsuperscript{2} \hspace{0.5cm} Enrico Gorgone\textsuperscript{3}

\textsuperscript{1} Dipartimento di Informatica, Università di Pisa
\textsuperscript{2} Centre Interuniversitaire de Recherche sur les Réseaux d’Entreprise, la Logistique et le Transport (CIRRELT), and Department of Computer Science and Operations Research Université de Montréal
\textsuperscript{3} Dipartimento di Matematica e Informatica, Università di Cagliari

\textit{7\textsuperscript{th} International Workshop on Freight Optimization and Logistics Odysseus 2018}
Cagliari, June 6, 2018
Multicommodity Flows & Decomposition

Making decomposition work

A new master problem

Computational results

The software issue

Conclusions and (a Lot of) Future Work
A generic Multicommodity flow model

- Graph $G = (N, A)$, a generic Multicommodity flow model
  \[
  \min \sum_{k \in K} \sum_{(i, j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i, j) \in A} f_{ij} y_{ij} \tag{1}
  \]
  \[
  \sum_{(i, j) \in A} x_{ij}^k - \sum_{(j, i) \in A} x_{ji}^k = b_i^k \quad \text{i.e.,} \quad i \in N, \ k \in K \tag{2}
  \]
  \[
  \sum_{k \in K} x_{ij}^k \leq u_{ij} y_{ij} \quad \text{(i, j) \in A} \tag{3}
  \]
  \[
  0 \leq x_{ij}^k \leq u_{ij}^k y_{ij} \quad \text{(i, j) \in A, k \in K} \tag{4}
  \]
  \[
  y \in Y \tag{5}
  \]

- Often $b_i^k \equiv (s^k, t^k, d^k)$, i.e., commodities $K \equiv$ O-D pairs, possibly with $x_{ij} \rightarrow d^k x_{ij}$, $x_{ij} \in \{0, 1\}$ (unsplittable routing)

- Countless many relevant special cases:
  - different $Y$ (often, but not always $\subseteq \{0, 1\}^{|A|}$) $\Rightarrow$ almost all graph design problems
  - bipartite graph $\Rightarrow$ facility location
  - multiple node/arc capacities by graph transformations . . .

- Countless many generalizations (extra constraints, nonlinearities, . . .)
Multicommodity flow applications

- Pervasive structure in logistic and transportation, often very large (time-space $\Rightarrow$ acyclic) $G$, “few” commodities
- Common in many other areas (telecommunications, energy, ...), possibly “small” (undirected) $G$, “many” commodities
- Interesting links with many hard problems (e.g. Max-Cut)
- Hard to solve in general: many (difficult) problems in one
- Even continuous versions “hard”: very-large-scale LPs
- Many sources of structure $\Rightarrow$ the paradise of decomposition$^{1,2}$

---

(Very) Classical decomposition approaches

- **Lagrangian relaxation**\(^3\) of linking constraints:
  - (3) + (4): \(\implies\) flow (shortest path) relaxation
  - (2): \(\implies\) knapsack relaxation
  - others possible (cf. Bernard’s talk)

- **Benders’ decomposition**\(^4\) of linking variables:
  - design \((y)\) variables are “naturally” linking
  - Benders’ cuts are metric inequalities defining the multiflow feasibility
  - Linking variables can be artificially added (resource decomposition)\(^5\)

\[
x_{ij}^k \leq u_{ij}^k \quad , \quad \sum_{k \in K} u_{ij}^k \leq u_{ij}
\]

- This talk about Lagrange, but many ideas can be applied to Benders\(^6\)

---

\(^3\) Geoffrion “Lagrangean relaxation for integer programming” *Math. Prog. Study*, 1974

\(^4\) Benders “Partitioning procedures for solving mixed-variables programming problems” *Num. Math.*, 1962


\(^6\) van Ackooij, F., de Oliveira “Inexact Stabilized Benders’ Decomposition Approaches, with Application […]” *CO&A*, 2016
Outline

1. Multicommodity Flows & Decomposition
2. Making decomposition work
3. A new master problem
4. Computational results
5. The software issue
6. Conclusions and (a Lot of) Future Work
Decomposition 101

- Simplifying the notation:
  \[
  (\Pi) \quad \max \left\{ \begin{array}{l}
  cx : \quad Ax = b, \; x \in X
  \end{array} \right. \]
  
  \(Ax = b\) “complicating” \(\equiv\) optimizing upon \(X\) “easy”

- Almost always \(X = \bigotimes_{h \in K} X^h (K \neq K) \equiv Ax = b\) linking constraints

- The best possible (convex = solvable) relaxation
  \[
  (\bar{\Pi}) \quad \max \{ \begin{array}{l}
  cx : \quad Ax = b, \; x \in \text{conv}(X)
  \end{array} \} \tag{6}
  \]

- All our \(X\) compact, represent \(\text{conv}(X)\) by vertices

  \[
  \text{conv}(X) = \left\{ x = \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} : \sum_{\bar{x} \in X} \theta_{\bar{x}} = 1, \; \theta_{\bar{x}} \geq 0, \; \bar{x} \in X \right\}
  \]

  \(\Rightarrow\) Dantzig-Wolfe reformulation\(^2\) of \((\bar{\Pi})\):

  \[
  (\tilde{\Pi}) \quad \left\{ \begin{array}{l}
  \max \quad c \left( \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} \right) \\
  A \left( \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} \right) = b \\
  \sum_{\bar{x} \in X} \theta_{\bar{x}} = 1, \; \theta_{\bar{x}} \geq 0, \; \bar{x} \in X
  \end{array} \right. \]
Dantzig-Wolfe decomposition \( \equiv \) Lagrangian relaxation

- \( B \subset X \) (small), solve master problem restricted to \( B \)
  \[(\Pi_B) \max \{ cx : Ax = b, x \in \text{conv}(B) \} \]

  feed (partial) dual optimal solution \( \lambda^* \) (of \( Ax = b \)) to pricing problem
  \[(\Pi_{\lambda^*}) \max \{ (c - \lambda^* A)x : x \in X \} \quad [ + \lambda^* b ] \]

  (Lagrangian relaxation), optimal solution \( \bar{x} \) of \( \Pi_{\lambda^*} \to B \)

  - Dual: \( (\Delta_B) \min \{ f_B(\lambda) = \max \{ cx + \lambda(b - Ax) : x \in B \} \} \)
  - \( f_B = \) lower approximation of “true” Lagrangian function
    \[ f(\lambda) = \max \{ cx + \lambda(b - Ax) : x \in X \} \]

  \( \implies (\Delta_B) \) outer approximation of Lagrangian dual \( \equiv (\tilde{\Pi}) \equiv (\tilde{\Pi}) \)

  \( (\Delta) \min \{ f(\lambda) = \max \{ cx + \lambda(b - Ax) : x \in X \} \} \tag{7} \)

- Dantzig-Wolfe decomposition \( \equiv \) Cutting Plane approach to \( (\Delta) \)

---

Issue with the approach: instability

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

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

- More in general: $\{\lambda^*_k\}$ is unstable, has no locality properties $\equiv$ convergence speed does not improve near the optimum

- The solution is pretty obvious: stabilize it

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

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$1e+4$</th>
<th>$1e+2$</th>
<th>$1e+0$</th>
<th>$1e-2$</th>
<th>$1e-4$</th>
<th>$1e-5$</th>
<th>$1e-6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>r.it.</td>
<td>1.07</td>
<td>1.12</td>
<td>0.86</td>
<td>0.77</td>
<td>0.56</td>
<td>0.19</td>
<td>0.04</td>
</tr>
</tbody>
</table>

(relative iterations to $\delta = \infty$)

- Would work wonders . . .
Issue with the approach: instability

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

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

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

- The solution is pretty obvious: stabilize it

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

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>1e+4</th>
<th>1e+2</th>
<th>1e+0</th>
<th>1e−2</th>
<th>1e−4</th>
<th>1e−5</th>
<th>1e−6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r.it.</td>
<td>1.07</td>
<td>1.12</td>
<td>0.86</td>
<td>0.77</td>
<td>0.56</td>
<td>0.19</td>
<td>0.04</td>
</tr>
</tbody>
</table>

(relative iterations to $\delta = \infty$)

- Would work wonders . . . if only we knew the dual optimum
Stabilizing DW

- Current point $\bar{\lambda}$, box of size $t > 0$ around it

- Stabilized dual master problem

$$\min \left\{ f_B(\bar{\lambda} + d) : \|d\|_\infty \leq t \right\}$$

- Corresponding stabilized primal master problem

$$\max \left\{ cx + \bar{\lambda}z - t\|z\|_1 : z = b - Ax, \; x \in \text{conv}(B) \right\}$$

i.e., just Dantzig-Wolfe with slacks

- When stuck and $z^* = b - Ax^* \neq 0$, either move $\bar{\lambda}$ or enlarge $t$

- Uses just LP tools, relatively minor modifications

- How should one choose $t$?

---

8 Marsten, Hogan, Blankenship “The Boxstep Method for Large-scale Optimization” OR, 1975
Choosing $t$

- Left/right = distance from dual optimum/relative gap
- Stabilized with (fixed) different $t$, un-stabilized ($t = \infty$)
- One can clearly over-stabilize
More general

- Perhaps a different stabilizing term would help? Why not

\[
(\Delta_{B,\bar{\lambda},t}) \quad \min \left\{ \ f_B(\bar{\lambda} + d) + \frac{1}{2t} \| d \|^2 \right\}
\]

- More general: stabilizing term \( D \), stabilized master problems

\[
(\Delta_{B,\bar{\lambda},D}) \quad \min \left\{ \ f_B(\bar{\lambda} + d) + D(d) \right\} \tag{10}
\]

\[
(\Pi_{B,\bar{\lambda},D}) \quad \max \left\{ \ cx + \bar{\lambda}(b - Ax) - D^*(Ax - b) : x \in \text{conv}(B) \right\}
\]

("*" = Fenchel’s conjugate): a generalized augmented Lagrangian

- Change \( \bar{\lambda} \) when \( f(\bar{\lambda} + d^*) \ll f(\bar{\lambda}) \), appropriate \( D \Rightarrow \) converges

- Nifty aggregation trick: still converges with “poorman bundle”

\[
B = \{ \ x^* \} \quad \text{(although rather slowly} \approx \text{volume}\)
\]

---

In practice?

- Either \( D = \frac{1}{2t} \| \cdot \|_2 \equiv D^* = \frac{1}{2} t \| \cdot \|_2 \), with specialized solvers\(^{13}\)
- Or its piecewise-linear approximations\(^{14}\)

\[
\begin{align*}
\text{max} & \quad c \left( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} \right) - \lambda (s^- + w^- - w^+ - s^+) \\
& \quad + \gamma^- s^- + \delta^- w^- + \delta^+ w^+ + \gamma^+ s^+ \\
& \quad A \left( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} \right) + s^+ + w^- - w^+ - s^+ = b \\
& \quad \sum_{\bar{x} \in B} \theta_{\bar{x}} = 1, \quad \theta_{\bar{x}} \geq 0, \quad \bar{x} \in B \\
& \quad 0 \leq s^- \leq \zeta^-, \quad 0 \leq s^+ \leq \zeta^+ \\
& \quad 0 \leq w^- \leq \varepsilon^-, \quad 0 \leq w^+ \leq \varepsilon^+
\end{align*}
\]

same constraints as \((\Pi_B)\) + some slack variables

- Can be made to work efficiently despite the complex master problem

\(^{13}\) F. “Solving semidefinite quadratic problems within nonsmooth optimization algorithms” *Computers & O.R.*, 1996

Disaggregate master problem

- Exploit separability: $X = X^1 \times X^2 \times \ldots \times X^{|K|} \implies \conv(X) = \conv(X^1) \times \conv(X^2) \times \ldots \times \conv(X^{|K|}) \implies$
  
  $$\max \quad \sum_{k \in K} c^k \left( \sum_{x^k \in X^k} \bar{x}^k \theta^k \right)$$
  $$\sum_{k \in K} A^k \left( \sum_{x^k \in X^k} \bar{x}^k \theta^k \right) = b$$
  $$\sum_{x^k \in X^k} \theta^k = 1 \quad , \quad \theta^k \geq 0 \quad k \in K$$

- Aggregated case: $\theta^k = \theta^h, h \neq k$ (rather innatural)

- (Many) more columns but sparser, more rows

- More efficient than aggregated formulation\(^{15}\)

- Master problem size $\approx$ time increases, but convergence speed increases a lot $\equiv$ consistent improvement

- It still has to be stabilized (most of the times)

---

Structed problem with “easy variables”:

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

\(X^1\) arbitrary, \(X^2\) has compact convex formulation

Example: \(y \in \{0, 1\}^{|A|}\) (Fixed-Charge MMCF)

Lagrangian function \(f(\lambda) = f^1(\lambda) + f^2(\lambda)(-\lambda b)\), two components
Stabilized decomposition with “easy components”

- Structured problem with “easy variables”:

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

$X^1$ arbitrary, $X^2$ has compact convex formulation

- Example: $y \in \{0, 1\}^{|A|}$ (Fixed-Charge MMCF)

- Lagrangian function $f(\lambda) = f^1(\lambda) + f^2(\lambda)(-\lambda b)$, two components

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

- Structured problem with “easy variables”:
  \[(\Pi) \quad \max \left\{ c_1 x_1 + c_2(x_2) : x_1 \in X^1, \ G(x_2) \leq g, \ A_1 x_1 + A_2 x_2 = b \right\}\]

  \(X^1\) arbitrary, \(X^2\) has compact convex formulation

- Example: \(y \in \{0, 1\}^{|A|}\) (Fixed-Charge MMCF)

- Lagrangian function \(f(\lambda) = f^1(\lambda) + f^2(\lambda)(-\lambda b)\), two components

- 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

\[(\Delta_{\mathcal{B},\bar{\lambda},\mathcal{D}}) \quad \min \left\{ b(\bar{\lambda} + d) + f^1_B(\bar{\lambda} + d) + f^2(\bar{x} + d) + \mathcal{D}(d) \right\} \]

- Primal master problem: abstract form

\[(\Pi_{\mathcal{B},\bar{\lambda},\mathcal{D}}) \quad \max \left\{ \begin{array}{l} c_1 x_1 + c_2(x_2) + \bar{\lambda}z - \mathcal{D}^*(-z) \\ z = b - A_1 x_1 - A_2 x_2 \\ x_1 \in \text{conv}(\mathcal{B}), \quad x_2 \in \mathcal{X}^2 \end{array} \right\} \]

and implementable form

\[(\Pi_{\mathcal{B},\bar{y},\mathcal{D}}) \quad \max \left\{ \begin{array}{l} c_1 \left( \sum_{\bar{x}_1 \in \mathcal{B}} \bar{x}_1 \theta_{\bar{x}_1} \right) + c_2(x_2) + \bar{\lambda}z - \mathcal{D}^*(-z) \\ z = b - A_1 \left( \sum_{\bar{x}_1 \in \mathcal{B}} \bar{x}_1 \theta_{\bar{x}_1} \right) - A_2 x_2 \\ \sum_{\bar{x}_1 \in \mathcal{B}} \theta_{\bar{x}_1} = 1, \quad G(x_2) \leq g \end{array} \right\} \quad (11) \]

- Barring some details (do not translate $f^1_B$), everything works\(^{16}\)

---

\(^{16}\) F., Gorgone “Bundle methods for sum-functions with “easy” components […]” *Math. Prog.*, 2014
A taste of computational results

- Flow relaxation of FC-MMCF ($Y = \{0, 1\}^{\lvert A \rvert}$)

- Several possible options:
  - fully aggregated (FA)
  - partly disaggregated with easy $y$ (PDE)
  - disaggregated with difficult $y$ (DD)
  - disaggregated with easy $y$ (DE)

- Stabilizing terms: $\| \cdot \|_{\infty}, \| \cdot \|_2^2$ only for (FA) (exploiting 13)

- Many forcing constraints (4) $\implies$ dynamic generation needed\textsuperscript{17,18}

<table>
<thead>
<tr>
<th></th>
<th>Cplex static</th>
<th>Cplex dynamic</th>
<th>DE static</th>
<th>DE dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>54</td>
<td>10</td>
<td>44</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>315</td>
<td>54</td>
<td>233</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>1539</td>
<td>112</td>
<td>1234</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>2789</td>
<td>458</td>
<td>2227</td>
<td>65</td>
</tr>
</tbody>
</table>

\textsuperscript{17} F., Lodi, Rinaldi “New approaches for optimizing over the semimetric polytope” Math. Prog., 2005

\textsuperscript{18} Belloni, Sagastizábal “Dynamic Bundle Methods” Math. Prog., 2009
Computational results: you have to do it all right

<table>
<thead>
<tr>
<th>DE</th>
<th>PDE</th>
<th>DD</th>
<th>FA-1</th>
<th>FA-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>iter</td>
<td>gap</td>
<td>time</td>
<td>iter</td>
</tr>
<tr>
<td>32</td>
<td>77</td>
<td>1e-7</td>
<td>1000</td>
<td>2980</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>3e-7</td>
<td>3000</td>
<td>2896</td>
</tr>
<tr>
<td>29</td>
<td>24</td>
<td>2e-7</td>
<td>9000</td>
<td>8370</td>
</tr>
<tr>
<td>65</td>
<td>20</td>
<td>3e-8</td>
<td>27000</td>
<td>5618</td>
</tr>
<tr>
<td>410</td>
<td>14880</td>
<td>9e-7</td>
<td>1855</td>
<td>11141</td>
</tr>
</tbody>
</table>

- Most (stabilized) decompositions simply too slow to converge
- To be efficient, you have to let information accumulate!
- Optimal setting: maximum $|B| = 50 \cdot |K|$, constraints violation checked at every iteration, constraints never removed

| opt | 20 $\cdot |K|$ | Rmv = 20 | Sep = 10 |
|-----|---------------|-----------|-----------|
| time | it | gap | time | it | gap | time | it | gap |
| 31.69 | 77 | 1e-7 | 289.41 | 841 | 7e-7 | 104.60 | 218 | 2e-7 |
| 47.53 | 30 | 3e-7 | 3000.76 | 1585 | 3e-4 | 1564.82 | 803 | 4e-5 |
| 28.98 | 24 | 2e-7 | 1125.93 | 726 | 4e-7 | 2585.05 | 796 | 1e-6 |
| 65.31 | 20 | 3e-8 | 81.33 | 20 | 3e-8 | 17415.68 | 2121 | 8e-5 |
| 72.96 | 194 | 1e-6 | 363.67 | 159 | 3e-7 | 141.61 | 65 | 1e-6 |
| 669.34 | 78 | 5e-7 | 669.34 | 78 | 5e-7 | 669.34 | 78 | 5e-7 |

trying to save on master problem cost a bad idea
Once you do it all right

<table>
<thead>
<tr>
<th>primal</th>
<th>dual</th>
<th>net.</th>
<th>barr.</th>
<th>DE 1e-6</th>
<th>1e-12</th>
<th>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 9e-7</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 3e-6</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 1e-6</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 1e-6</td>
</tr>
<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 1e-6</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 1e-6</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 1e-6</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 1e-6</td>
</tr>
<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 1e-6</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 2e-5</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 3e-6</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 1e-6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time</th>
<th>f add</th>
<th>it</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.6</td>
<td>0.5</td>
<td>875</td>
</tr>
<tr>
<td>46</td>
<td>0.9</td>
<td>1.1</td>
<td>871</td>
</tr>
<tr>
<td>51</td>
<td>8.6</td>
<td>10.6</td>
<td>760</td>
</tr>
<tr>
<td>12</td>
<td>2.3</td>
<td>2.2</td>
<td>796</td>
</tr>
<tr>
<td>26</td>
<td>5.1</td>
<td>5.0</td>
<td>760</td>
</tr>
<tr>
<td>6</td>
<td>0.9</td>
<td>1.1</td>
<td>871</td>
</tr>
<tr>
<td>12</td>
<td>2.1</td>
<td>2.4</td>
<td>831</td>
</tr>
<tr>
<td>26</td>
<td>4.5</td>
<td>5.4</td>
<td>794</td>
</tr>
<tr>
<td>51</td>
<td>8.6</td>
<td>10.6</td>
<td>760</td>
</tr>
<tr>
<td>12</td>
<td>2.0</td>
<td>2.3</td>
<td>827</td>
</tr>
<tr>
<td>29</td>
<td>5.0</td>
<td>6.1</td>
<td>869</td>
</tr>
<tr>
<td>58</td>
<td>9.2</td>
<td>13.0</td>
<td>817</td>
</tr>
<tr>
<td>2706</td>
<td>16.7</td>
<td>24.1</td>
<td>765</td>
</tr>
</tbody>
</table>

- **Fa–V**: a FA with volume algorithm, **quick but too coarse**
- More than an order of magnitude to **Cplex** as $|A|$ and/or $|K|$ grows
- Can be extended to **dynamic easy components**\(^{19}\)

Outline

1. Multicommodity Flows & Decomposition
2. Making decomposition work
3. A new master problem
4. Computational results
5. The software issue
6. Conclusions and (a Lot of) Future Work
Motivation: knapsack decomposition

- Relax the flow conservation constraints (2)

\[
\begin{align*}
\min & \quad \sum_{(i,j) \in A} \left( \sum_{k \in K} (c_{ij}^k - \pi_i^k + \pi_j^k)x_{ij}^k + f_{ij}y_{ij} \right) \\
\sum_{k \in K} d_k x_{ij}^k & \leq u_{ij} y_{ij} \\
0 & \leq x_{ij}^k \leq u_{ij}^k y_{ij} \\
y & \in Y
\end{align*}
\]

- Decomposes by arc if \(Y = \{0, 1\}\)\(|A|\), easy \((\approx \text{continuous knapsack})\) but \textbf{no integrality property} \(\Rightarrow\) \textbf{better bound}

- Still solvable with (appropriate) \(Y \subset \{0, 1\}\)\(|A|\): optimal \(x_{ij}^*(\pi)\) gives cost \(f_{ij}^*(\pi)\), then \(\min \left\{ \sum_{(i,j) \in A} f_{ij}^*(\pi)y_{ij} : y \in Y \right\}\)

- However, Lagrangian function no longer separable: goodbye
disaggregate master problem, easy components, and all the rest

- Still, the Lagrangian problem is \textbf{somewhat separable}

- We want to “show this quasi-separability to the master problem”
General setting: quasi-separable problems

- Set of $N$ quasi-continuous (vector) variables $x_i$ governed by $y_i$

  \[
  \begin{align*}
  \text{max } & \quad dy + \sum_{i \in N} c_i x_i \\
  \text{subject to } & \quad Dy + \sum_{i \in N} C_i x_i = b \\
  & \quad A_i x_i \leq b_i y_i \quad i \in N \\
  & \quad x_i \in X_i \quad i \in N \\
  & \quad y \in Y
  \end{align*}
  \]

- $m$ linking constraints (13): Lagrangian relaxation

  \[
  \phi(\lambda) = \lambda b + \max \left\{ (d - \lambda D)y + \sum_{i \in N}(c_i - \lambda C_i)x_i : (14), (15), (16) \right\}
  \]

  solved with above two-stage procedure:

  \[
  \phi_i(\lambda) = \max \left\{ (c_i - \lambda C_i)x_i : x_i \in X_i \right\} \quad i \in N
  \]

  \[
  \phi(\lambda) = \lambda b + \max \left\{ \sum_{i \in N}(d_i - \lambda D^i + \phi_i(\lambda))y_i : y \in Y \right\}
  \]
Making it separable: the dumb way

- (Un-stabilized) master problem is not disaggregate:

\[
\max \sum_{(\bar{y}, \bar{x}) \in YX} (d\bar{y} + \sum_{i \in N} c_i \bar{x}_i) \theta(\bar{y}, \bar{x})
\]

\[
\sum_{(\bar{y}, \bar{x}) \in YX} (Dy + \sum_{i \in N} C_i x_i) \theta(\bar{y}, \bar{x}) = b
\]

\[
\sum_{(\bar{y}, \bar{x}) \in YX} \theta(\bar{y}, \bar{x}) = 1, \quad \theta(\bar{y}, \bar{x}) \geq 0 \quad (\bar{y}, \bar{x}) \in YX
\]

- To make it so also relax (14) with multipliers \( \mu = [\mu_i]_{i \in N} \geq 0 \)

\[
\phi(\lambda, \mu) = \lambda b + \psi(\lambda, \mu) + \sum_{i \in N} \psi_i(\lambda, \mu_i) \quad \text{with}
\]

\[
\psi_i(\lambda, \mu_i) = \max \left\{ (c_i - \lambda C_i - \mu_i A_i)x_i : x_i \in X_i \right\}
\]

\[
\psi(\lambda, \mu) = \max \left\{ \sum_{i \in N} (d_i - \lambda D^i - \mu_i b_i)y_i : y \in Y \right\}
\]

- Many more multipliers (\(|K| |A| \) in FC-MMCF), can easily destroy any advantage due to separability
Making it separable: the better way

- "Easy component" version: $X_i = \text{convex combination, original } Y$
  
  \[
  \max dy + \sum_{i \in N} \sum_{\bar{x}_i \in \bar{X}_i} (c_i \bar{x}_i) \theta_{\bar{x}_i} \\
  Dy + \sum_{i \in N} \sum_{\bar{x}_i \in \bar{X}_i} (C_i \bar{x}_i) \theta_{\bar{x}_i} = b \\
  \sum_{\bar{x}_i \in \bar{X}_i} (A_i \bar{x}_i) \theta_{\bar{x}_i} \leq y_i \quad i \in N \\
  \sum_{\bar{x}_i \in \bar{X}_i} \theta_{\bar{x}_i} \leq 1 \\
  \sum_{\bar{x}_i \in \bar{X}_i} \theta_{\bar{x}_i} \leq 1 \\
  y \in Y \ , \ \theta_{\bar{x}_i} \geq 0 \quad \bar{x}_i \in \bar{X}_i \ , \ i \in N \\
  \] (assuming $0 \in \bar{X}_i$, but generalizes)

- Nifty idea: replace (27)–(28) with
  \[
  \sum_{\bar{x}_i \in \bar{X}_i} \theta_{\bar{x}_i} \leq y_i \quad i \in N \\
  \] (29)

  then relax (29) with multipliers $\mu = [\mu_i]_{i \in N} \geq 0$ (much fewer now)

- Multipliers are from master problem constraints (which they are . . . )

- Non-easy component version obvious
Outline

1. Multicommodity Flows & Decomposition
2. Making decomposition work
3. A new master problem
4. Computational results
5. The software issue
6. Conclusions and (a Lot of) Future Work
Computational results

Er... I said it'd be quick...

No, seriously, we still don't have them.

We believe they will be good because a similar approach has been used for CFL.

We haven't had the time to test this yet.

It may be interesting to discuss a bit why...

...apart from the fact that we are lazy Italians (and Quebecois), of course.

---

Computational results

- Er . . . I said it’d be quick . . .

---

Computational results

- Er . . . I said it’d be quick . . .
- No, seriously, we still don’t have them

---

Computational results

- Er . . . I said it’d be quick . . .
- No, seriously, we still don’t have them
- We believe they will be good because a similar approach has been used for CFL\textsuperscript{20}
- We haven’t had the time to test this yet
- It may be interesting to discuss a bit why

\textsuperscript{20}Klose, Görtz “A branch-and-price algorithm for the capacitated facility location problem” \textit{EJOR}, 2007
Computational results

- Er . . . I said it’d be quick . . .

- No, seriously, we still don’t have them

- We believe they will be good because a similar approach has been used for CFL\textsuperscript{20}

- We haven’t had the time to test this yet

- It may be interesting to discuss a bit why . . . apart from the fact that we are lazy Italians (and Quebecois), of course

\textsuperscript{20} Klose, Görtz “A branch-and-price algorithm for the capacitated facility location problem” \textit{EJOR}, 2007
Outline

1 Multicommodity Flows & Decomposition

2 Making decomposition work

3 A new master problem

4 Computational results

5 The software issue

6 Conclusions and (a Lot of) Future Work
Putting all this in practice

- ...easier said than done
- Specialized implementations for one application “relatively easy”
- General implementations for all problems with same structure harder: it took \(\approx 10\) years from idea to paper for easy components on top of existing, nicely structured C++ bundle code
- Issue: extracting structure from problems
- Issue: really using this in a B&C approach
  \(\approx 20\) years doing this well for Multicommodity Network Design
- Especially hard: multiple nested forms of structure, reformulation
- Current modelling/solving tools just don’t do it
- So we are building our own under the auspices of plan4res
  https://www.plan4res.eu/
What We Want

A modelling language/system which:

- explicitly supports the notion of block ≡ nested structure
- separately provides “semantic” information from “syntactic” details (list of constraints/variables)
- allows exploiting specialised solvers on blocks with specific structure
- caters all needs of complex methods: dynamic generation of constraints/variables, modifications in the data, reoptimization

C++ library: set of “core” classes, easily extendable

Why C++? A number of reasons:

- all serious solvers are written in C/C++
- we all love it (especially C++11/14)
- tried with Julia/JuMP, but could not handle well C++ interface
The Core SMS++
**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., `Block::BinKnapsackBlock` = a 0/1 knapsack problem)

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

- **Abstract representation** of a Block:
  - one (for now) `ObjectiveFunction`
  - any # of groups of (pointers) to (static) `Variable`
  - any # of groups of `std::list` of (pointers) to (dynamic) `Variable`
  - any # of groups of (pointers) to (static) `Constraint`
  - any # of groups of `std::list` of (pointers) to (dynamic) `Constraint`
  - groups of `Variable`/`Constraint` can be single (`std::list`) or `std::vector` (…) or `boost::multi_array` thanks to `boost::any`

- Any # of sub-Blocks (recursively), possibly of specific type (e.g., `Block::MMCFBlock` can have $k$ `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)
- Keeps the set of Constraint/ObjectiveFunction it influences

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

- Modification::VariableModification (fix/unfix)
Constraint

- Abstract concept, thought to be extended (any algebraic constraint, a matrix constraint, a PDE constraint, bilevel program, ...)
- Keeps the set of Variables it is influenced from
- Either satisfied or not by the current value of the Variables
- 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
- Modification::ConstraintModification (relax/enforce)
ObjectiveFunction

- Abstract concept, perhaps to be extended (vector-valued ...)
- Either minimized or maximized
- Keeps the set of Variables it depends from
- Can be evaluated w.r.t. the current value of the Variables (but its value depends on the specific form)
- ObjectiveFunction::RealObjectiveFunction 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 ObjectiveFunction)
- Modification::OFModification (change verse)
Any # of Solvers attached to a Block to solve it

Solver:: for a specific Block:: can use the physical representation
\[ \Rightarrow \text{no need for explicit Constraints} \]
\[ \Rightarrow \text{abstract representation of Block only constructed on demand} \]

However, Variables are always present (interface with Solver)

A general-purpose Solver uses the abstract representation

Dynamic Variable/Constraints can be generated on demand
(user cuts/lazy constraints/column generation)

For a Solver attached to a Block:

- Variables not belonging to the Block are constants
- Constraints not belonging to the Block are ignored
(belonging = declared there or in any sub-Block recursively)

ObjectiveFunction of sub-Blocks summed to that of father Block
if has same verse, but min/max supported
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 Solvers can be attached to the same Block
- Solutions are written directly into the Variables of the Block
- Individual Solvers can be attached to sub-Blocks 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 Modifications

- Heavily slanted towards RealObjectiveFunction (optimality guarantees being upper and lower bounds)
- Derived CDASolver is “Convex Duality Aware”: bounds are associated to dual solutions (possibly, multiple)
- Something relevant may be missing, asynchronous calls not clear yet
Block and Modification

- Most Block components can change, but not all:
  - set of sub-Blocks
  - number and shape of groups of Variables/Constraints

- 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 Solvers concerned
  - abstract Modification, only Solvers using it concerned

- Abstract Modification on Variable/Constraint must always be issued, even if no Solver, to keep both representations in sync

- A single change may trigger more than one Modification

- A Solver will disregard a Modification it does not understand (there must always be another one it understands)

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

- Almost empty base class, then everything has its own derived ones
- Each change to Block/Variable/Constraint ... produces a Modification, and a smart pointer is passed to the Block
- The Block funnels it to the interested Solvers (above, if any)
- Heavy stuff can be attached to a Modification (e.g., added/deleted dynamic Variable/Constraints)
- Each Solver has the responsibility of cleaning up its list of Modifications (smart pointers → memory will finally be released)
- Modifications processed in the arrival order to ensure consistency
- Solvers are supposed to reoptimize to improve efficiency, which is easier if you can see all list of changes at once (lazy update)
- A Solver may optimize the changes (Modifications may cancel each outer out ...), but its responsibility
Solution and Configuration

- Block produces one Solution, possibly using its sub-Blocks’
- A 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 the primal/dual solution
- Block, Solution are tree-structured complex objects
- Configuration for them a (possibly) tree-structured complex object but also Configuration::SimpleConfiguration (an int)
- Configuration::BlockConfiguration sets (recursively):
  - which dynamic Variable/Constraints are generated, how (Solver, time limit ...)
  - which Solvers attached to each sub-Block
  - which Solution is produced ...
R³ Block

- 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 R³ Blocks Block:::-specific, a Configuration needed

- R³ Block completely independent (new Variable/Constraints), useful for algorithmic purposes (branch, fix, solve, . . .)

- Solution of R³ Block useful to Solvers for original Block: `map_back_solution()` (best effort in case of dynamic Variables)

- Sometimes keeping R³ Block in sync with original necessary: `map_forward_modifications()`, task of original Block

- `map_forward_solution()` and `map_back_modifications()` useful, e.g., dynamic generation of Variable/Constraints in the R³ Block

- Block:: is in charge of all this, thus decides what it supports
First Basic Implementations

- Variable::ColVariable implements “value = one single real”, possibly restricted to $\mathbb{Z}$, with (possibly infinite) bounds

- Modification::ColVariableModification (change bounds, type)

- Constraint::RowConstraint implements “$l \leq \text{a real} \leq u$”

- Has dual variable attached to it (single real)

- Modification::RowConstraintModification (change $l$, $u$)

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

- RealObjectiveFunction::FRealObjectiveFunction: “value” given by a Function
Function

- Function only deals with (real) values
- Approximate computation supported in a quite general way\(^{21}\)
- Asynchronous evaluation still not defined
- Handles set of Variables upon which it depends
- FunctionModification[Variables] for “easy” changes \(\implies\) reoptimization (shift, adding/removing “quasi separable” Variables)

\(^{21}\) van Ackooij, F. “Incremental bundle methods using upper models” *SIOPT*, 2018
C05Function/C15Function deal with 1st/2nd 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 → reoptimization (linearizations shift, some linearizations entries changing in simple ways)

C15Function supports Hessians, unclear how much reoptimization possible/useful
LagrangianFunction

- C05Function::LagrangianFunction has one isolated Block + set of (so far) LinearFunction to define Lagrangian term
- evaluate() = Block.get_registered_solvers()[i].solve(): asynchronous Solver $\implies$ asynchronous Function
- Solutions extracted from Block $\equiv$ linearizations
- Solver provides local pool
- LagrangianFunction handles global pool
- All changes lead to reoptimization-friendly Modification
- BendersFunction should be quite similar
Other useful stuff

- `un_any_thin()` template functions/macros to extract
  (std::vector or boost::multi_array of) (std::list of) Variable/Constraints out of a boost_any and work on that

- Solution::ColVariableSolution 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

- Solution::RowConstraintSolution uses the abstract representation of any Block that only have (...) RowConstraints to read/write the dual solution

- Of course, Solution::CVFRSolution ...

- Solver::MILPSolver solves with Cplex any Block that only has (...) ColVariables, FRowConstraints and FRealObjectiveFunction with LinearFunctions (uses the abstract representation)
Application to Multicommodity flows

- Different reformulations from same basic Block
- Streamlined interface with decomposition solvers
- General decomposition-based B&B now (perhaps) possible
Application to Multicommodity flows

- Different reformulations from same basic Block
- Streamlined interface with decomposition solvers
- General decomposition-based B&B now (perhaps) possible
Outline

1. Multicommodity Flows & Decomposition
2. Making decomposition work
3. A new master problem
4. Computational results
5. The software issue
6. Conclusions and (a Lot of) Future Work
Decomposition for Multicommodity flows is a very old idea, yet a lot of work required to make it efficient.

Crucial aspect: large, structured master problems.

Our proposal: yet another large, structured master problem.

Huge challenge: make these techniques mainstream (at least, less desperately bleeding-edge).

A new hope: structured modelling system.

Alpha version, not all the features you have seen are complete.

Design principles have kept evolving, new ideas continue to crop up.

Core nicely general, but only success in applications validate it.

Overhead still largely unknown (although C++ efficient).

Asynchronous still to be figured out (but very relevant).

Not for the faint of heart, but we are trying. Someone cares to join?
Acknowledgements

Copyright © PLAN4RES Partners 2018, 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.

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