Separable Lagrangian Decomposition for Quasi-Separable Problems

(with application to Multicommodity Network Design)

Antonio Frangioni[1] Bernard Gendron[2] Enrico Gorgone[3]

1. Dipartimento di Informatica, Università di Pisa
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
3. Dipartimento di Matematica e Informatica, Università di Cagliari

6^{th} International Symposium on Combinatorial Optimization

ISCO 2020

Montreal, Canada, May 6, 2020

Outline

A generic Multicommodity flow model

- Graph $G = (N, A)$, a generic Multicommodity flow model

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c^k_{ij} x^k_{ij} + \sum_{(i,j) \in A} f_{ij} y_{ij} \qquad (1)$$

$$\sum_{(i,j) \in A} x^k_{ij} - \sum_{(j,i) \in A} x^k_{ji} = b^k_i \qquad i \in N \,,\ k \in K \qquad (2)$$

$$\sum_{k \in K} x^k_{ij} \leq u_{ij} y_{ij} \qquad (i,j) \in A \qquad (3)$$

$$0 \leq x^k_{ij} \leq u^k_{ij} y_{ij} \qquad (i,j) \in A \,,\ k \in K \qquad (4)$$

$$y \in Y \qquad (5)$$

- Often $b^k_i \equiv (s^k \,,\ t^k \,,\ d^k)$, 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 $\subseteq \{\, 0 \,,\ 1 \,\}^{|A|}$) $\Longrightarrow$ almost all graph design problems
  - bipartite graph $\Longrightarrow$ 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 $\Longrightarrow$ 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 $\Longrightarrow$ the paradise of decomposition[1,2]

---

1 Ford, Fulkerson “A Suggested Computation for Maximal Multicommodity Network Flows” *Man. Sci.*, 1958

2 Dantzig, Wolfe “The Decomposition Principle for Linear Programs” *Op. Res.*, 1960

(Very) Classical decomposition approaches

- Lagrangian relaxation[3] of linking constraints:
  - (3) + (4): $\Longrightarrow$ flow (shortest path) relaxation
  - (2): $\Longrightarrow$ knapsack relaxation
  - others possible[4]
- Benders' decomposition[5] 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)[6]

$$x^k_{ij} \leq u^k_{ij} \qquad , \qquad \sum_{k \in K} u^k_{ij} \leq u_{ij}$$

- This talk about Lagrange, but many ideas can be applied to Benders[7]

---

[3] Geoffrion "Lagrangean relaxation for integer programming" *Math. Prog. Study*, 1974

[4] Kazemzadeh, Bektas, Crainic, F., Gendron, Gorgone "Node-Based Lagrangian Relaxations for Multicommodity Capacitated Fixed-Charge Network Design" Technical Report CIRRELT-2019-21, 2019

[5] Benders "Partitioning procedures for solving mixed-variables programming problems" *Num. Math.*, 1962

[6] Kennington, Shalaby "An Effective Subgradient Procedure for Minimal Cost Multicomm. Flow Problems" *Man. Sci.* 1977

[7] van Ackooij, F., de Oliveira "Inexact Stabilized Benders' Decomposition Approaches, with Application [. . .]" *CO&A*, 2016

Decomposition 101

- Simplifying the notation:

$$(\Pi) \qquad \max \{ \, cx \; : \; Ax = b \; , \; x \in X \, \}$$

$Ax = b$ "complicating" $\equiv$ optimizing upon X "easy"

- Almost always $X = \bigotimes_{h \in \mathcal{K}} X^h$ ($\mathcal{K} \neq K$) $\equiv$ $Ax = b$ linking constraints
- The best possible (convex = solvable) relaxation

$$(\bar{\Pi}) \qquad \max \{ \, cx \; : \; Ax = b \; , \; x \in conv(X) \, \} \tag{6}$$

- All our X compact, represent $conv(X)$ by vertices

$$conv(X) = \{ \, x = \sum_{\bar{x} \in X} \bar{x}\theta_{\bar{x}} \; : \; \sum_{\bar{x} \in X} \theta_{\bar{x}} = 1 \; , \; \theta_{\bar{x}} \geq 0 \quad \bar{x} \in X \, \}$$

$\Longrightarrow$ Dantzig-Wolfe reformulation[2] of $(\bar{\Pi})$:

$$(\tilde{\Pi}) \qquad \begin{cases} \max \quad c \left(\sum_{\bar{x} \in X} \bar{x}\theta_{\bar{x}} \right) \\ \qquad\quad A \left(\sum_{\bar{x} \in X} \bar{x}\theta_{\bar{x}} \right) = b \\ \qquad\qquad \sum_{\bar{x} \in X} \theta_{\bar{x}} = 1 \quad , \quad \theta_{\bar{x}} \geq 0 \qquad \bar{x} \in X \end{cases}$$

D-W decomposition $\equiv$ Lagrangian relaxation

- $\mathcal{B} \subset X$ (small), solve master problem restricted to $\mathcal{B}$

$$(\Pi_{\mathcal{B}}) \qquad \max \{ \, cx \; : \; Ax = b \, , \; x \in conv(\mathcal{B}) \, \}$$

feed (partial) dual optimal solution λ^* (of $Ax = b$) to pricing problem

$$(\Pi_{\lambda^*}) \qquad \max \{ \, (c - \lambda^* A)x \; : \; x \in X \, \} \quad [+ \lambda^* b \,]$$

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

- Dual: $(\Delta_{\mathcal{B}})$ $\min \{ \, f_{\mathcal{B}}(\lambda) = \max \{ \, cx + \lambda(b - Ax) \; : \; x \in \mathcal{B} \, \} \, \}$
- $f_{\mathcal{B}}$ = lower approximation of "true" Lagrangian function

$$f(\lambda) = \max \{ \, cx + \lambda(b - Ax) \; : \; x \in X \, \}$$

$\Longrightarrow$ $(\Delta_{\mathcal{B}})$ outer approximation of Lagrangian dual $\equiv (\bar{\Pi})$

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

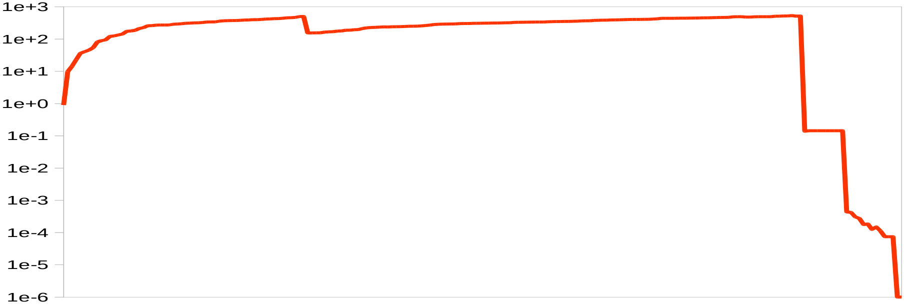
- Dantzig-Wolfe decomposition $\equiv$ Cutting Plane approach to (Δ)[8]

[8] Kelley "The Cutting-Plane Method for Solving Convex Programs" *Journal of the SIAM*, 1960

All well and nice, but does it work well?

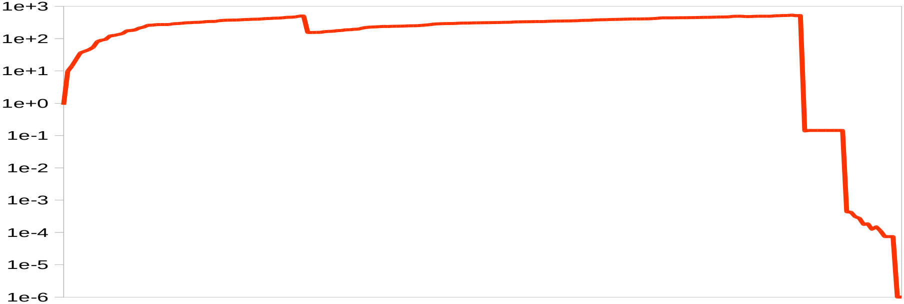
All well and nice, but does it work well?

- By-the-book? Not really



All well and nice, but does it work well?

- By-the-book? Not really



- λ^* immediately shoots much farther from optimum than initial point
  $\equiv$ having good initial point not much useful
- Apparently no improvement for a long time as information slowly accrues
- A mysterious threshold is hit and “real” convergence begins

Outline

How to deal with instability

- λ^*_{k+1} can be very far from λ^*_k, where $f_{\mathcal{B}}$ is a "bad model" of f
- If $\{\lambda^*_k\}$ is unstable, then stabilize it around Current point $\bar{\lambda}$
- Stabilizing term $\mathcal{D}_t$ with parameter t, stabilized master problems

$$\begin{array}{ll}(\Delta_{\mathcal{B},\bar{\lambda},\mathcal{D}_t}) & \min \{ f_{\mathcal{B}}(\bar{\lambda} + d) + \mathcal{D}_t(d) \} \\ (\Pi_{\mathcal{B},\bar{\lambda},\mathcal{D}_t}) & \max \{ cx + \bar{\lambda}(b - Ax) - \mathcal{D}^*_t(Ax - b) \; : \; x \in conv(\mathcal{B}) \}\end{array} \tag{8}$$

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

- Change $\bar{\lambda}$ when $f(\bar{\lambda} + d^*) \ll f(\bar{\lambda})$, appropriate $\mathcal{D} \Longrightarrow$ converges[9]
- Choosing t nontrivial
- Aggregation trick: right $\mathcal{D} \Longrightarrow$ still converges with "poorman bundle" $\mathcal{B} = \{ x^* \}$ (although rather slowly[10] $\approx$ volume[11] $\equiv$ subgradient)

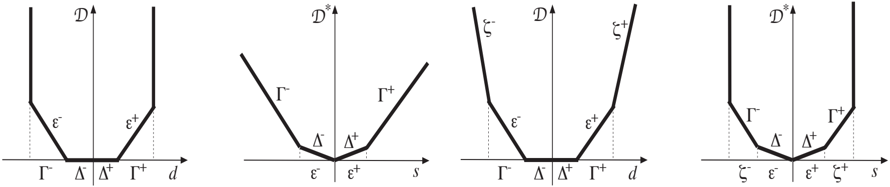
[9] F. "Generalized Bundle Methods" *SIOPT*, 2002

[10] Briant, Lemaréchal, et. al. "Comparison of bundle and classical column generation" *Math. Prog.*, 2006

[11] Bahiense, Maculan, Sagastizábal "The volume algorithm revisited: relation with bundle methods" *Math. Prog.*, 2002

What is an appropriate stabilization?

- Simplest: $\mathcal{D}_t \equiv \| d \|_\infty \leq t$, $\mathcal{D}^*_t = t\| \cdot \|_2^2$ ("boxstep")[12]
- Better[13]: $\mathcal{D}_t = \frac{1}{2t}\| \cdot \|_2^2$, $\mathcal{D}^*_t = \frac{1}{2}t\| \cdot \|_2^2$ (may use specialized QP solvers[14])
- Keep LP master: piecewise-linear approximations[15]



- Several other ideas[16] (level stabilization, centres, better "Hessian", ...)

---

[12] Marsten, Hogan, Blankenship "The Boxstep Method for Large-scale Optimization" *OR*, 1975

[13] Lemaréchal "Bundle Methods in Nonsmooth Optimization" in *Nonsmooth Optimization* vol. 3, 1978

[14] F. "Solving semidefinite quadratic problems within nonsmooth optimization algorithms" *Computers & O.R.*, 1996

[15] Ben Amor, Desrosiers, F. "On the choice of explicit stabilizing terms in column generation" *Disc. Appl. Math.*, 2009

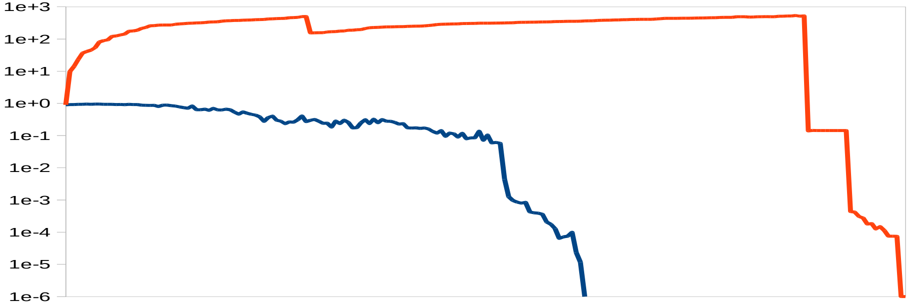
[16] F., "Standard Bundle Methods: Untrusted Models and Duality" in *Numerical Nonsmooth Optimization: ...*, 2020

All well and nice, but does it work well?

[17] Nemirovsky, Yudin "Problem Complexity and Method Efficiency in Optimization" Wiley, 1983

All well and nice, but does it work well?

- It depends on what "well" means, but surely better



- Black-box nonsmooth optimization is $\Omega(1/\varepsilon^2)$ in general[17]
- Convergence slow (but at lest some) until mysterious threshold hit
- At least, better information accrued sooner $\Longrightarrow$ "quick tail" starts sooner

[17] Nemirovsky, Yudin "Problem Complexity and Method Efficiency in Optimization" Wiley, 1983

Disaggregate master problem

- Exploit separability: $X = X^1 \times X^2 \times \ldots \times X^{|K|} \Longrightarrow$
  $conv(X) = conv(X^1) \times conv(X^2) \times \ldots \times conv(X^{|K|}) \Longrightarrow$
  $$(\Pi_{\mathcal{B}}) \quad \max \left\{ \sum_{k\in\mathcal{K}} c^k x^k \ : \ \sum_{k\in\mathcal{K}} A^k x^k = b \, , \ x^k \in conv(\mathcal{B}^k) \ \ k \in \mathcal{K} \right\}$$
  $$\equiv \quad \begin{array}{ll} \max & \sum_{k\in\mathcal{K}} c^k \left(\sum_{\bar{x}^k \in X^k} \bar{x}^k \theta^k_{\bar{x}} \right) \\ & \sum_{k\in\mathcal{K}} A^k \left(\sum_{\bar{x}^k \in X^k} \bar{x}^k \theta^k_{\bar{x}} \right) = b \\ & \sum_{\bar{x}^k \in X^k} \theta^k_{\bar{x}} = 1 \quad , \quad \theta^k \geq 0 \qquad k \in \mathcal{K} \end{array}$$
- Aggregated case: $\theta^k = \theta^h$, $h \neq k$ (rather innatural)
- (Many) more columns but sparser, more rows
- Can be seen as a reformulation trick in original space[18]
- Dual: $f(\lambda)$ is a sum-function, so $f_{\mathcal{B}}$ also should be
  $$(\Delta_{\mathcal{B}}) \quad \min \left\{ \lambda b + \sum_{k\in\mathcal{K}} f^k_{\mathcal{B}}(\lambda) = \max \left\{ (c^k - \lambda A^k) x^k \ : \ x^k \in \mathcal{B}^k \right\} \right\}$$

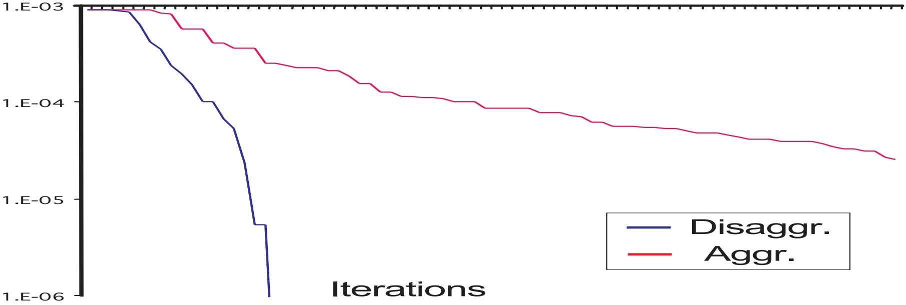
[18] Jones, Lustig, et. al. "Multicommodity Network Flows: The Impact of Formulation on Decomposition" *Math. Prog.*, 1993

All well and nice, but does it work well?

[19] Helmberg, Pichler "Dynamic Scaling and Submodel Selection in Bundle Methods [. . .]" Preprint 2017-04, TU Chemnitz, 2017

All well and nice, but does it work well?

- Has several trade-offs, but surely converges faster



- Master problem size ≈ time increases, but convergence speed increases a lot $\Longrightarrow$ most often better
- It still has to be stabilized (most of the times)
- Can play the partial aggregation trick[19] but details still rather unclear

[19] Helmberg, Pichler "Dynamic Scaling and Submodel Selection in Bundle Methods [...]" Preprint 2017-04, TU Chemnitz, 2017

- Separable subproblem with "easy component":

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

  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

"Easy components"

- Separable subproblem with "easy component":

$$(\Pi) \quad \max\{\, c_1x_1 + c_2(x_2) \;:\; x_1 \in X^1 \,,\; G(x_2) \leq g \,,\; A_1x_1 + A_2x_2 = b \,\}$$

  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

"Easy components"

- Separable subproblem with "easy component":

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

  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

The master problems

- Primal master problem:

$$(\Pi_{\mathcal{B}}) \quad \max \left\{ \begin{array}{l} c_1 x_1 + c_2(x_2) \\ A_1 x_1 - A_2 x_2 = b \\ x_1 \in conv(\mathcal{B}) \;\; , \;\; x_2 \in X^2 \end{array} \right.$$

$$\equiv \quad \max \left\{ \begin{array}{l} c_1 \left(\sum_{\bar{x}_1 \in \mathcal{B}} \bar{x}_1 \theta_{\bar{x}_1} \right) + c_2(x_2) \\ A_1 \left(\sum_{\bar{x}_1 \in \mathcal{B}} \bar{x}_1 \theta_{\bar{x}_1} \right) + A_2 x_2 = b \\ \sum_{\bar{x}_1 \in \mathcal{B}} \theta_{\bar{x}_1} = 1 \;\; , \;\; G(x_2) \leq g \end{array} \right. \tag{9}$$

"just use the easy set in the master problem"

- Dual master problem: $(\Delta_{\mathcal{B}}) \;\; \min \{ \; \lambda b + f^1_{\mathcal{B}}(\lambda) + f^2(\lambda) \; \}$
- Barring some details (do not translate $f^1_{\mathcal{B}}$), everything works[20]
- Of course, stabilization + multiple easy/hard components . . .

[20] F., Gorgone "Bundle methods for sum-functions with "easy" components [. . .]" *Math. Prog.*, 2014

All well and nice, but does it work well?

[21] F., Gendron "A Stabilized Structured Dantzig-Wolfe Decomposition Method" *Math. Prog.*, 2013

All well and nice, but does it work well?

- You have to do it right (let information accumulate)

[21] F., Gendron "A Stabilized Structured Dantzig-Wolfe Decomposition Method" *Math. Prog.*, 2013

All well and nice, but does it work well?

- You have to do it right (let information accumulate)

Cplex	easy		aggregate			volume		
dual	1e-6	1e-12	time	it	gap	time	it	gap
39	26	32	322	10320	1e-6	6	871	8e-3
132	28	56	294	5300	1e-6	12	831	9e-3
301	21	26	5033	27231	1e-6	26	794	3e-3
1930	133	133	3122	14547	1e-6	51	760	4e-2
131	2	3	344	7169	1e-6	12	827	3e-3
708	246	337	2256	17034	2e-5	29	869	1e-2
2167	284	508	5475	15061	3e-6	58	817	2e-2
8908	242	253	11863	13953	1e-6	109	765	2e-2

- Much better accuracy/time than Cplex and competing decompositions
- Can be extended to dynamic easy components[21]
- You need all the tricks of the trade ≡ master problem reformulations

[21] F., Gendron "A Stabilized Structured Dantzig-Wolfe Decomposition Method" *Math. Prog.*, 2013

Outline

Motivation: knapsack decomposition

- Relax the flow conservation constraints (2)

$$
\begin{array}{lll}
\min & \sum_{(i,j)\in A} \big(\sum_{k \in K} (c^k_{ij} - \pi^k_i + \pi^k_j) x^k_{ij} + f_{ij} y_{ij} \big) & \\
 & \sum_{k \in K} d^k x^k_{ij} \leq u_{ij} y_{ij} & (i,j) \in A \\
 & 0 \leq x^k_{ij} \leq u^k_{ij} y_{ij} & (i,j) \in A \, , \; k \in K \\
 & y \in Y &
\end{array}
$$

- If $Y = \{\, 0 \, , \, 1 \,\}^{|A|}$, then it decomposes by arc
- If x^k_{ij} continuous, continuous knapsack + discrete decision $\Longrightarrow$ no integrality property $\Longrightarrow$ better bound
- Still reasonable if x^k_{ij} discrete (knapsack, costly but even better bound)
- Used to be one of the best choices for Lagrangian approaches[22,23]

[22] Crainic, F., Gendron "Bundle-based relaxation methods for multicommodity [...] network design" *Disc. Appl. Math.*, 2001

[23] Holmberg, Hellstrand "Solving the uncapacitated network design problem by a Lagrangian heuristic [...]" *OR*, 1998

Knapsack decomposition for non-separable Y

- Still solvable with (appropriate) $Y \subset \{ 0 , 1 \}^{|A|}$: first

$$
\begin{array}{rll}
f^*_{ij}(\pi) = \min & \sum_{k \in K} (c^k_{ij} - \pi^k_i + \pi^k_j) x^k_{ij} & \\
& \sum_{k \in K} d^k x^k_{ij} \leq u_{ij} & \\
& 0 \leq x^k_{ij} \leq u^k_{ij} & k \in K
\end{array}
$$

  and then $\min \{ \sum_{(i,j) \in A} (f^*_{ij}(\pi) + f_{ij}) y_{ij} \; : \; y \in Y \}$

- Computational cost $\approx$ same (if Y not too nasty), but Lagrangian function no longer separable
- Wave goodbye to disaggregate master problem $\Longrightarrow$ easy components $\Longrightarrow$ knapsack decomposition fallen out of favour
- Still, the Lagrangian problem is 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

$$\max \; dy + \textstyle\sum_{i \in N} c_i x_i \tag{10}$$
$$Dy + \textstyle\sum_{i \in N} C_i x_i = b \tag{11}$$
$$A_i x_i \leq b_i y_i \qquad i \in N \tag{12}$$
$$x_i \in X_i \qquad i \in N \tag{13}$$
$$y \in Y \tag{14}$$

- m linking constraints (11): Lagrangian relaxation

$$\phi(\lambda) = \lambda b + \max \; \{ \, (d - \lambda D) y + \textstyle\sum_{i \in N} (c_i - \lambda C_i) x_i \; : \; (12) \, , \; (13) \, , \; (14) \, \}$$

- Two-stage solution procedure

$$\phi_i(\lambda) = \max \; \{ \, (c_i - \lambda C_i) x_i \; : \; x_i \in X_i \, \} \qquad i \in N \tag{15}$$
$$\phi(\lambda) = \lambda b + \max \; \{ \, \textstyle\sum_{i \in N} (d_i - \lambda D^i + \phi_i(\lambda)) y_i \; : \; y \in Y \, \} \tag{16}$$

- D-W reformulation is not disaggregate

$$\max \sum_{(\bar{y},\bar{x}) \in YX} \big(d\bar{y} + \sum_{i \in N} c_i \bar{x}_i \big) \theta_{(\bar{y},\bar{x})} \quad (17)$$

$$\sum_{(\bar{y},\bar{x}) \in YX} \big(D\bar{y} + \sum_{i \in N} C_i \bar{x}_i \big) \theta_{(\bar{y},\bar{x})} = b \quad (18)$$

$$\sum_{(\bar{y},\bar{x}) \in YX} \theta_{(\bar{y},\bar{x})} = 1 \quad , \quad \theta_{(\bar{y},\bar{x})} \geq 0 \qquad (\bar{y},\bar{x}) \in YX \quad (19)$$

- Can be made so the hard way: also relax (12) ($\mu = [\mu_i]_{i \in N} \geq 0$)

$$\phi(\lambda, \mu) = \lambda b + \psi(\lambda, \mu) + \sum_{i \in N} \psi_i(\lambda, \mu_i) \qquad \text{with} \quad (20)$$

$$\psi_i(\lambda, \mu_i) = \max \big\{ (c_i - \lambda C_i - \mu_i A_i) x_i \ : \ x_i \in X_i \big\} \quad (21)$$

$$\psi(\lambda, \mu) = \max \big\{ \sum_{i \in N} (d_i - \lambda D^i - \mu_i b_i) y_i \ : \ y \in Y \big\} \quad (22)$$

- Many more multiplayers ($|K||A|$ in FC-MMCF)
- Can easily destroy any advantage due to separability

- "Easy component" Y version:

$$\max\ dy + \sum_{i \in N} \sum_{\bar{x}_i \in X_i} (c_i \bar{x}_i) \theta_{\bar{x}_i} \quad (23)$$

$$Dy + \sum_{i \in N} \sum_{\bar{x}_i \in X_i} (C_i \bar{x}_i) \theta_{\bar{x}_i} = b \quad (24)$$

$$\sum_{\bar{x}_i \in X_i} (A_i \bar{x}_i) \theta_{\bar{x}_i} \leq y_i \qquad i \in N \quad (25)$$

$$\sum_{\bar{x}_i \in X_i} \theta_{\bar{x}_i} = 1 \qquad i \in N \quad (26)$$

$$y \in Y \ , \ \theta_{\bar{x}_i} \geq 0 \qquad \bar{x}_i \in X_i \ , \ i \in N$$

- Nifty idea: replace (25)–(26) with

$$\sum_{\bar{x}_i \in \bar{X}_i} \theta_{\bar{x}_i} = y_i \qquad i \in N \quad (27)$$

  then relax (27) with multipliers $\gamma = [\gamma_i]_{i \in N} \geq 0$
- Multipliers are from master problem constraints (which they are ...)
- Non-easy component version obvious
- Much fewer multipliers (1 instead of m), much more elegant

Outline

Computational results

[24] Klose, Görtz "A branch-and-price algorithm for the capacitated facility location problem" *EJOR*, 2007

Computational results

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

[24] Klose, Görtz "A branch-and-price algorithm for the capacitated facility location problem" *EJOR*, 2007

Computational results

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

[24] Klose, Görtz "A branch-and-price algorithm for the capacitated facility location problem" *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[24]
- We haven't had the time to test this yet
- It may be interesting to discuss a bit why

[24] Klose, Görtz "A branch-and-price algorithm for the capacitated facility location problem" *EJOR*, 2007

Outline

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 ≈ 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
  ≈ 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

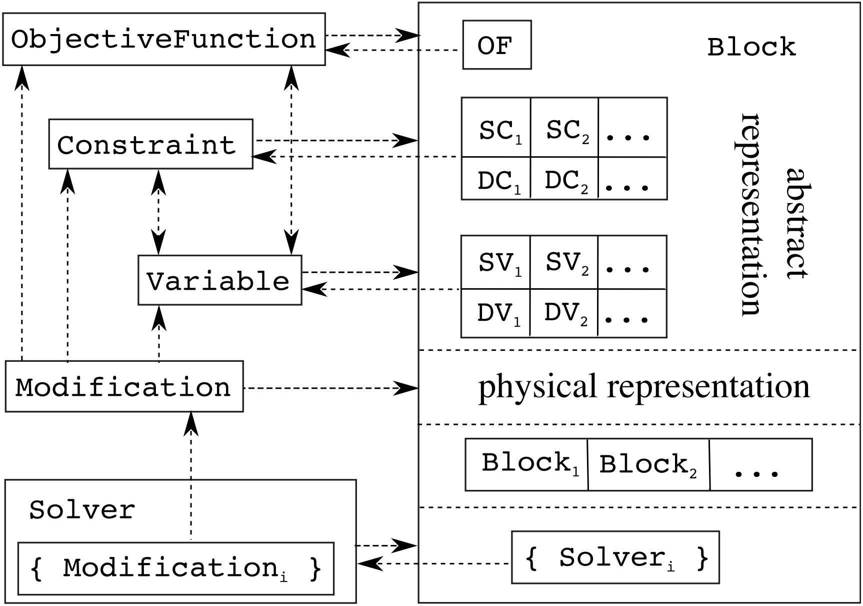
  https://www.plan4res.eu/

Design goals

- A modelling system which:
  - 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 solvers on blocks with specific structure
  - caters all needs of complex solution methods: dynamic generation of constraints/variables, modifications in the data, reoptimization, ...
- Open source (LGPL3) `C++17` library

  `https://gitlab.com/smspp/smspp-project`
- Easily extendable "core" classes + [interface with] efficient general solvers
- Built-in asynchronous and parallel capabilities (thanks Cray!)
- Set of (more or less) specialized blocks/solvers for plan4res

The Core SMS++



ObjectiveFunction
Constraint
Variable
Modification
Solver
{ Modification$_i$ }
OF
Block
SC$_1$ SC$_2$ ...
DC$_1$ DC$_2$ ...
SV$_1$ SV$_2$ ...
DV$_1$ DV$_2$ ...
abstract representation
physical representation
Block$_1$ Block$_2$ ...
{ Solver$_i$ }

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
- Physical representation: whatever data describes the instance
- Abstract representation of a Block:
  - one Objective (but possibly vector-valued)
  - any # of groups of (pointers) to static/dynamic Variable
  - any # of groups of (pointers) to static/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
- Many support mechanisms:
  - general netCDF serialize/deserialize
  - factory + "methods factory"
  - Configuration, BlockConfiguration, BlockSolverConfiguration
  - R^3Block concept ...

Solver

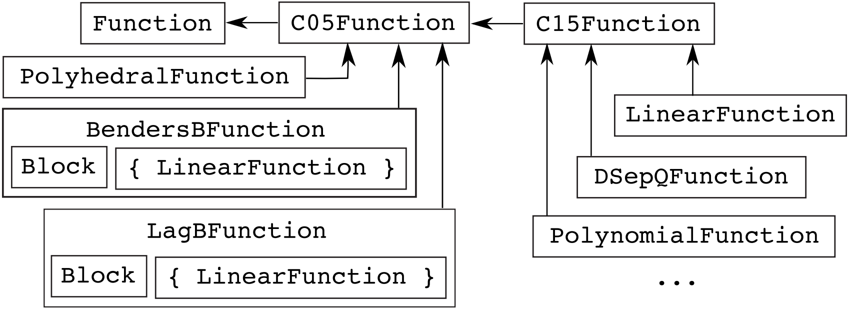
- Any # of Solver attached to a Block to solve it
- :Solver for a specific :Block can use the physical representation
  $\Longrightarrow$ no need for explicit Constraint
  $\Longrightarrow$ abstract representation of Block only constructed on demand
- A general-purpose Solver uses the abstract representation
- Dynamic Variable/Constraint can be generated on demand
- 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
- Somehow slanted towards RealObjective (optimality guarantees = upper and lower bounds)
- CDASolver:Solver is "Convex Duality Aware": bounds are associated to dual solutions (possibly, multiple)

Modification

- Most Block components can change (but not all)
- Any change is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification object
- Two different kinds of Modification (what changes):
  - physical Modification, only specialized Solver concerned
  - abstract Modification, only Solver using it concerned
- 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 → memory eventually released)
- Solver supposedly reoptimize to improve efficiency, which is easier if you can see all Modification at once (may cancel each outer out)
- GroupModification (recursively) packs many Modification together

R^3Block

- Often reformulation crucial, but also relaxation or restriction: `get_R3_Block()` produces one, possibly using sub-`Block`s'
- Obvious special case: copy (clone), should always work
- Available R^3Blocks `Block::`-specific
- R^3Block completely independent (new `Variable`/`Constraints`), useful for algorithmic purposes (branch, fix, solve, ...)
- Solution of R^3Block useful to `Solvers` for original `Block`: `map_back_solution()` (best effort in case of dynamic `Variables`)
- Sometimes keeping R^3Block 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 R^3Block
- `Block::` is in charge of all this, thus decides what it supports



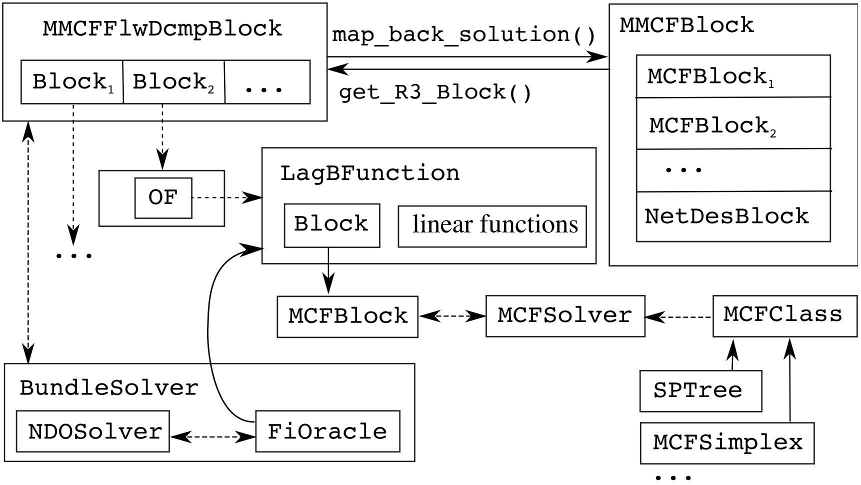
- Function only deals with (real) values
- Handles set of Variables upon which it depends
- Approximate computation supported in a quite general way[25]
- Asynchronous Function computation possible
- FunctionModification[Variables] for "easy" changes $\Longrightarrow$ reoptimization (shift, adding/removing "quasi separable" Variables)

[25] van Ackooij, F. "Incremental bundle methods using upper models" *SIOPT*, 2018

C05Function

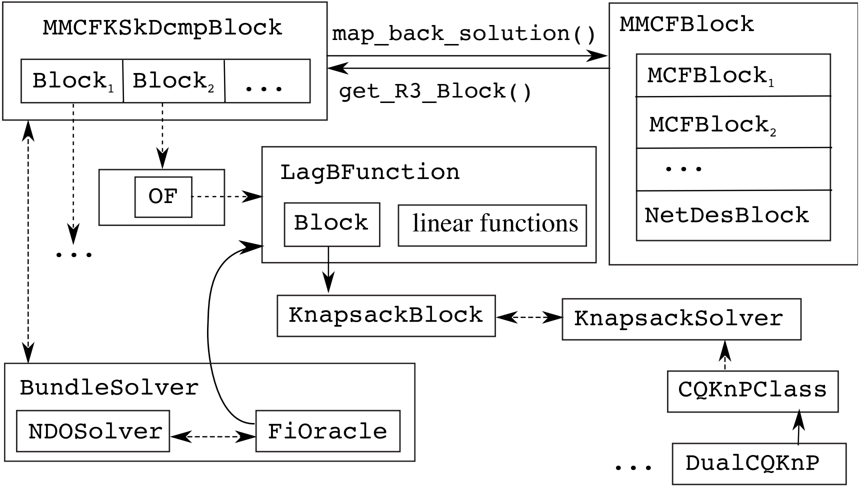
- C05Function/C15Function deal with 1st/2nd order information
- General concept of "linearization" (gradient, Clarke subgradient, ...)
- Multiple linearizations produced at each evaluation (local pool)
- Global pool of linearizations for reoptimization:
  - convex combination of linearizations
  - "important linearization" (at optimality)
- C05Function::LagBFunction has one isolated Block
  + set of (so far) LinearFunction to define Lagrangian term
- asynchronous Solver $\Longrightarrow$ asynchronous Function
- Solutions from Block $\equiv$ linearizations: Solver provides local pool
- LagBFunction handles global pool
- All changes lead to reoptimization-friendly Modification
- BendersBFunction similar (linearization $\equiv$ dual solution)

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

A Lot of Work, Then Maybe Conclusions

- Decomposition for Multicommodity flows a very old idea, yet a lot of work required to make it efficient
- Crucial aspect: proper reformulations of master problems
- Our proposal: yet another proper reformulation of master problem
- Huge challenge: make these techniques mainstream (at least, less desperately bleeding-edge)
- A new hope: structured modelling system
- Beta 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 in its infancy (but seems nice)
- Not for the faint of heart, but we are trying. Someone cares to join?

Acknowledgements

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