Separable Lagrangian decomposition for the Knapsack Relaxation of 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

7^{th} International Workshop on Freight Optimization and Logistics
Odysseus 2018

Cagliari, June 6, 2018

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 (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]

$$x_{ij}^k \leq u_{ij}^k \qquad , \qquad \sum_{k \in K} u_{ij}^k \leq u_{ij}$$

- 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

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

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

Outline

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 ($\bar{\Pi}$):

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

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

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

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

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

- Dantzig-Wolfe decomposition $\equiv$ Cutting Plane approach to (Δ)[7]

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

Issue with the approach: instability

- λ^*_{k+1} can be very far from λ^*_k, where $f_{\mathcal{B}}$ is a "bad model" of f
- $(\Pi_{\mathcal{B}})$ empty $\equiv$ $(\Delta_{\mathcal{B}})$ 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 δ

δ	1e+4	1e+2	1e+0	1e−2	1e−4	1e−5	1e−6
r.it.	1.07	1.12	0.86	0.77	0.56	0.19	0.04

  (relative iterations to $\delta = \infty$)
- Would work wonders ...

Issue with the approach: instability

- λ^*_{k+1} can be very far from λ^*_k, where $f_{\mathcal{B}}$ is a "bad model" of f
- $(\Pi_{\mathcal{B}})$ empty $\equiv$ $(\Delta_{\mathcal{B}})$ 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 δ

δ	1e+4	1e+2	1e+0	1e−2	1e−4	1e−5	1e−6
r.it.	1.07	1.12	0.86	0.77	0.56	0.19	0.04

(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[8]

$$(\Delta_{\mathcal{B},\bar{\lambda},t}) \qquad \min \left\{ \, f_{\mathcal{B}}(\, \bar{\lambda} + d \,) \; : \; \| \, d \, \|_{\infty} \leq t \, \right\} \tag{8}$$

- Corresponding stabilized primal master problem

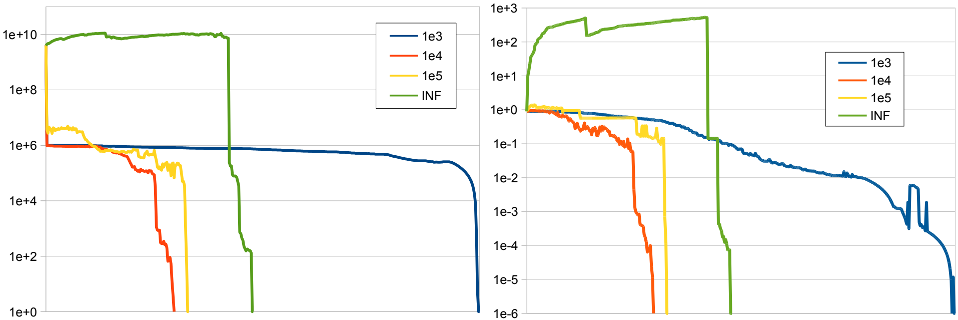
$$(\Pi_{\mathcal{B},\bar{\lambda},t}) \quad \max \left\{ \, cx + \bar{\lambda} z - t \| \, z \, \|_1 \; : \; z = b - Ax \; , \; x \in conv(\mathcal{B}) \, \right\} \tag{9}$$

  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[9]

$$(\Delta_{\mathcal{B},\bar{\lambda},t}) \qquad \min \left\{ f_{\mathcal{B}}(\bar{\lambda} + d) + \tfrac{1}{2t} \| d \|_2^2 \right\}$$

- More general: stabilizing term $\mathcal{D}$, stabilized master problems

$$\begin{array}{ll} (\Delta_{\mathcal{B},\bar{\lambda},\mathcal{D}}) & \min \left\{ f_{\mathcal{B}}(\bar{\lambda} + d) + \mathcal{D}(d) \right\} \\ (\Pi_{\mathcal{B},\bar{\lambda},\mathcal{D}}) & \max \left\{ cx + \bar{\lambda}(b - Ax) - \mathcal{D}^*(Ax - b) \, : \, x \in conv(\mathcal{B}) \right\} \end{array} \qquad (10)$$

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

- Change $\bar{\lambda}$ when $f(\bar{\lambda} + d^*) \ll f(\bar{\lambda})$, appropriate $\mathcal{D} \Longrightarrow$ converges[10]
- Nifty aggregation trick: still converges with "poorman bundle" $\mathcal{B} = \{ x^* \}$ (although rather slowly[11] $\approx$ volume[12] $\equiv$ subgradient)

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

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

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

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

In practice?

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

$$
(\Pi_{\mathcal{B},\bar{\lambda},\mathcal{D}}) \quad \left\{ \begin{array}{ll} \max & c\left(\sum_{\bar{x}\in\mathcal{B}} \bar{x}\theta_{\bar{x}}\right) - \bar{\lambda}\left(s^- + w^- - w^+ - s^+\right) \\ & \qquad + \gamma^- s^- + \delta^- w^- + \delta^+ w^+ + \gamma^+ s^+ \\ & A\left(\sum_{\bar{x}\in\mathcal{B}} \bar{x}\theta_{\bar{x}}\right) + s^- + w^- - w^+ - s^+ = b \\ & \sum_{\bar{x}\in\mathcal{B}} \theta_{\bar{x}} = 1 \;,\; \theta_{\bar{x}} \geq 0 \quad \bar{x} \in \mathcal{B} \\ & 0 \leq s^- \leq \zeta^- \;,\; 0 \leq s^+ \leq \zeta^+ \\ & 0 \leq w^- \leq \varepsilon^- \;,\; 0 \leq w^+ \leq \varepsilon^+ \end{array} \right.
$$

same constraints as $(\Pi_{\mathcal{B}})$ + 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

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

Disaggregate master problem

- Exploit separability: $X = X^1 \times X^2 \times \ldots \times X^{|K|} \Longrightarrow$
  $conv(X) = conv(X^1) \times conv(X^2) \times \ldots \times conv(X^{|K|}) \Longrightarrow$

$$\begin{aligned} \max \quad & \textstyle\sum_{k \in \mathcal{K}} c^k \left(\sum_{\bar{x}^k \in X^k} \ \bar{x}^k \theta^k_{\bar{x}} \right) \\ & \textstyle\sum_{k \in \mathcal{K}} A^k \left(\sum_{\bar{x}^k \in X^k} \ \bar{x}^k \theta^k_{\bar{x}} \right) = b \\ & \textstyle\sum_{\bar{x}^k \in X^k} \ \theta^k_{\bar{x}} = 1 \quad , \quad \theta^k \geq 0 \qquad k \in \mathcal{K} \end{aligned}$$

- 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)

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

Stabilized decomposition with “easy components”

- Structured problem with “easy variables”:

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

  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

Stabilized decomposition with “easy components”

- Structured problem with “easy variables”:

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

  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

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

- 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

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

- Primal master problem: abstract form

$$(\Pi_{\mathcal{B},\bar{\lambda},\mathcal{D}}) \quad \max \left\{ \begin{array}{l} c_1x_1 + c_2(x_2) + \bar{\lambda}z - \mathcal{D}^*(-z) \\ z = b - A_1x_1 - A_2x_2 \\ x_1 \in conv(\mathcal{B}) \;\; , \;\; x_2 \in X^2 \end{array} \right.$$

and implementable form

$$(\Pi_{\mathcal{B},\bar{y},\mathcal{D}}) \quad \max \left\{ \begin{array}{l} c_1 \left(\sum_{\bar{x}_1 \in \mathcal{B}} \bar{x}_1 \theta_{\bar{x}_1} \right) + c_2(x_2) + \bar{\lambda}z - \mathcal{D}^*(-z) \\ z = b - A_1 \left(\sum_{\bar{x}_1 \in \mathcal{B}} \bar{x}_1 \theta_{\bar{x}_1} \right) - A_2x_2 \\ \sum_{\bar{x}_1 \in \mathcal{B}} \theta_{\bar{x}_1} = 1 \;\; , \;\; G(x_2) \leq g \end{array} \right. \qquad (11)$$

- Barring some details (do not translate $f^1_{\mathcal{B}}$), 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 ($Y = \{0, 1\}^{|A|}$)
- 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) $\Longrightarrow$ dynamic generation needed[17,18]

Cplex		DE	
static	dynamic	static	dynamic
54	10	44	32
315	54	233	48
1539	112	1234	29
2789	458	2227	65

[17] F., Lodi, Rinaldi "New approaches for optimizing over the semimetric polytope" *Math. Prog.*, 2005

[18] Belloni, Sagastizábal "Dynamic Bundle Methods" *Math. Prog.*, 2009

Computational results: you have to do it all right

DE			PDE			DD			FA-1			FA-2		
time	iter	gap	time	iter	gap	time	iter	gap	time	iter	gap	time	iter	gap
32	77	1e-7	1000	2980	2e-2	1000	2714	2e-1	1000	1990	2e-1	410	14880	9e-7
48	30	3e-7	3000	2896	6e-2	3000	3720	7e-2	3000	7351	2e-1	1855	11141	3e-6
29	24	2e-7	9000	8370	2e-2	9000	5061	5e-2	9000	10918	1e-1	1254	9035	2e-6
65	20	3e-8	27000	5618	3e-2	27000	2148	4e-2	27000	5293	8e-2	1732	12940	1e-6

- Most (stabilized) decompositions simply too slow to converge
- To be efficient, you have to let information accumulate!
- Optimal setting: maximum $|\mathcal{B}| = 50 \cdot |K|$, 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	time	it	gap
31.69	77	1e-7	289.41	841	7e-7	104.60	218	2e-7	72.96	194	1e-6
47.53	30	3e-7	3000.76	1585	3e-4	1564.82	803	4e-5	363.67	159	3e-7
28.98	24	2e-7	1125.93	726	4e-7	2585.05	796	1e-6	141.61	65	1e-6
65.31	20	3e-8	81.33	20	3e-8	17415.68	2121	8e-5	669.34	78	5e-7

trying to save on master problem cost a bad idea

Once you do it all right

Cplex				DE		FA–2					FA–V				
primal	dual	net.	barr.	1e−6	1e−12	time	f	add	it	gap	time	f	add	it	gap
12	10	11	15	32	64	410	12	7	14880	9e-7	3	0.6	0.5	875	9e-3
64	53	61	71	48	51	1855	19	16	11141	3e-6	6	1.2	1.2	842	2e-2
139	114	132	157	29	29	1254	32	20	9035	1e-6	12	2.3	2.2	796	3e-2
559	456	531	587	65	66	1732	100	67	12940	1e-6	26	5.1	5.0	760	4e-2
46	39	43	60	26	32	322	12	10	10320	1e-6	6	0.9	1.1	871	8e-3
147	132	144	209	28	56	294	15	9	5300	1e-6	12	2.1	2.4	831	9e-3
509	301	478	648	21	26	5033	169	155	27231	1e-6	26	4.5	5.4	794	3e-3
2329	1930	2302	2590	133	133	3122	192	169	14547	1e-6	51	8.6	10.6	760	4e-2
196	131	156	304	2	3	344	20	12	7169	1e-6	12	2.0	2.3	827	3e-3
926	708	862	1174	246	337	2256	111	118	17034	2e-5	29	5.0	6.1	869	1e-2
2706	2167	2542	3272	284	508	5475	192	249	15061	3e-6	58	9.2	13.0	817	2e-2
11156	8908	11675	11683	242	253	11863	349	413	13953	1e-6	109	16.7	24.1	765	2e-2

- 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]

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

Outline

Motivation: knapsack decomposition

- Relax the flow conservation constraints (2)

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

- Decomposes by arc if $Y = \{\,0\,,\,1\,\}^{|A|}$, easy ($\approx$ (continuous) knapsack) but no integrality property $\Longrightarrow$ better bound
- Still solvable with (appropriate) $Y \subset \{\,0\,,\,1\,\}^{|A|}$: optimal $x^*_{ij}(\pi)$ gives cost $f^*_{ij}(\pi)$, then $\min\{\, \sum_{(i,j)\in A} f^*_{ij}(\pi)y_{ij} \;:\; y\in Y\,\}$
- However, Lagrangian function no longer separable: goodbye disaggregate master problem, easy components, and all the rest
- 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

$$\max \; dy + \textstyle\sum_{i \in N} c_i x_i \tag{12}$$
$$Dy + \textstyle\sum_{i \in N} C_i x_i = b \tag{13}$$
$$A_i x_i \leq b_i y_i \qquad i \in N \tag{14}$$
$$x_i \in X_i \qquad i \in N \tag{15}$$
$$y \in Y \tag{16}$$

- m linking constraints (13): Lagrangian relaxation

$$\phi(\lambda) = \lambda b + \max \; \{\, (d - \lambda D)y + \textstyle\sum_{i \in N}(c_i - \lambda C_i)x_i \;:\; (14)\,,\,(15)\,,\,(16) \,\}$$

solved with above two-stage procedure:

$$\phi_i(\lambda) = \max \; \{\, (c_i - \lambda C_i)x_i \;:\; x_i \in X_i \,\} \qquad i \in N \tag{17}$$
$$\phi(\lambda) = \lambda b + \max \; \{\, \textstyle\sum_{i \in N}(d_i - \lambda D^i + \phi_i(\lambda))y_i \;:\; y \in Y \,\} \tag{18}$$

Making it separable: the dumb way

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

$$\max \ \sum_{(\bar{y},\bar{x}) \in YX} \big(\, d\bar{y} + \sum_{i \in N} c_i \bar{x}_i \, \big) \theta_{(\bar{y},\bar{x})} \tag{19}$$

$$\sum_{(\bar{y},\bar{x}) \in YX} \big(\, Dy + \sum_{i \in N} C_i x_i \, \big) \theta_{(\bar{y},\bar{x})} = b \tag{20}$$

$$\sum_{(\bar{y},\bar{x}) \in YX} \theta_{(\bar{y},\bar{x})} = 1 \quad , \quad \theta_{(\bar{y},\bar{x})} \geq 0 \qquad (\bar{y},\bar{x}) \in YX \tag{21}$$

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

$$\phi(\lambda,\mu) = \lambda b + \psi(\lambda,\mu) + \sum_{i \in N} \psi_i(\lambda,\mu_i) \qquad \text{with} \tag{22}$$

$$\psi_i(\lambda,\mu_i) = \max \ \{ \, (c_i - \lambda C_i - \mu_i A_i) x_i \ : \ x_i \in X_i \, \} \tag{23}$$

$$\psi(\lambda,\mu) = \max \ \{ \ \sum_{i \in N} (d_i - \lambda D^i - \mu_i b_i) y_i \ : \ y \in Y \, \} \tag{24}$$

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

Making it separable: the better way

- "Easy component" version: X_i = convex combination, original Y

$$\max\ dy + \sum_{i \in N} \sum_{\bar{x}_i \in \bar{X}_i} (c_i \bar{x}_i) \theta_{\bar{x}_i} \tag{25}$$

$$Dy + \sum_{i \in N} \sum_{\bar{x}_i \in \bar{X}_i} (C_i \bar{x}_i) \theta_{\bar{x}_i} = b \tag{26}$$

$$\sum_{\bar{x}_i \in \bar{X}_i} (A_i \bar{x}_i) \theta_{\bar{x}_i} \leq y_i \qquad i \in N \tag{27}$$

$$\sum_{\bar{x}_i \in \bar{X}_i} \theta_{\bar{x}_i} \leq 1 \tag{28}$$

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

  (assuming $0 \in \bar{X}_i$, but generalizes)

- Nifty idea: replace (27)–(28) with

$$\sum_{\bar{x}_i \in \bar{X}_i} \theta_{\bar{x}_i} \leq y_i \qquad i \in N \tag{29}$$

  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

Computational results

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

Computational results

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

[20] 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

[20] 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[20]
- We haven't had the time to test this yet
- It may be interesting to discuss a bit why

[20] 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[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

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

Outline

Putting all this in practice

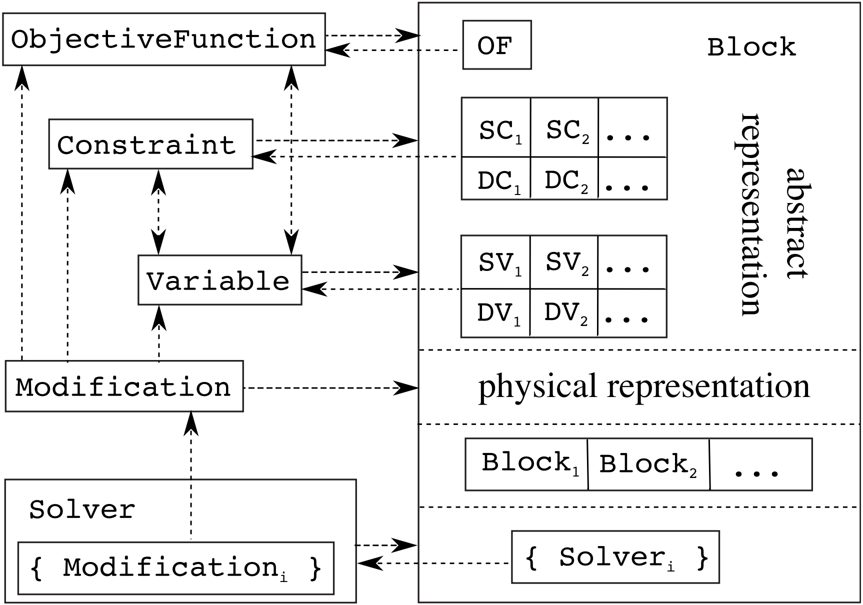
- . . . easier said than done
- Specialized implementations for one application "relatively easy"
- General implementations for all problems with same structure harder: it took ≈ 10 years from idea to paper for easy components on top of existing, nicely structured C++ bundle code
- Issue: extracting 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/

What We Want

- A modelling language/system which:
  - explicitly supports the notion of block $\equiv$ 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++



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 (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::MCFBlocks 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 $\Longrightarrow$ copying a `Variable` makes a different `Variable` $\Longrightarrow$ 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 $\Longrightarrow$ copying a `Constraint` makes a different `Constraint` $\Longrightarrow$ 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)

Block and Solver

- Any # of Solvers attached to a Block to solve it
- Solver:: for a specific Block:: can use the physical representation
  $\Longrightarrow$ no need for explicit Constraints
  $\Longrightarrow$ 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 ...

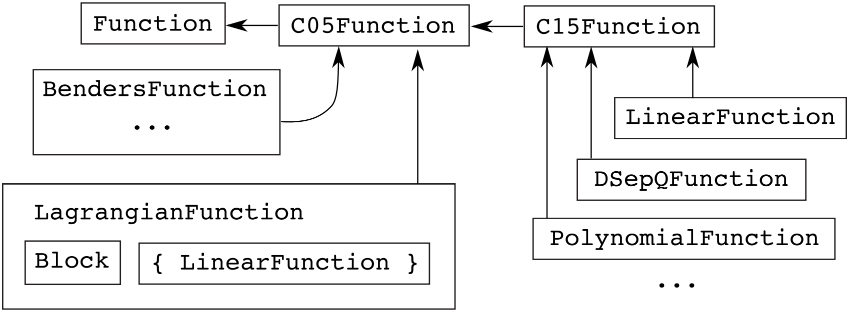
R^3Block

- 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, a `Configuration` needed
- 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

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 "$l \leq$ 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 ⟹ reoptimization (shift, adding/removing "quasi separable" Variables)

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

C05Function

- C05Function/C15Function deal with $1^{st}/2^{nd}$ 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 $\Longrightarrow$ reoptimization (linearizations shift, some linearizations entries changing in simple ways)
- C15Function supports Hessians, unclear how much reoptimization possible/useful

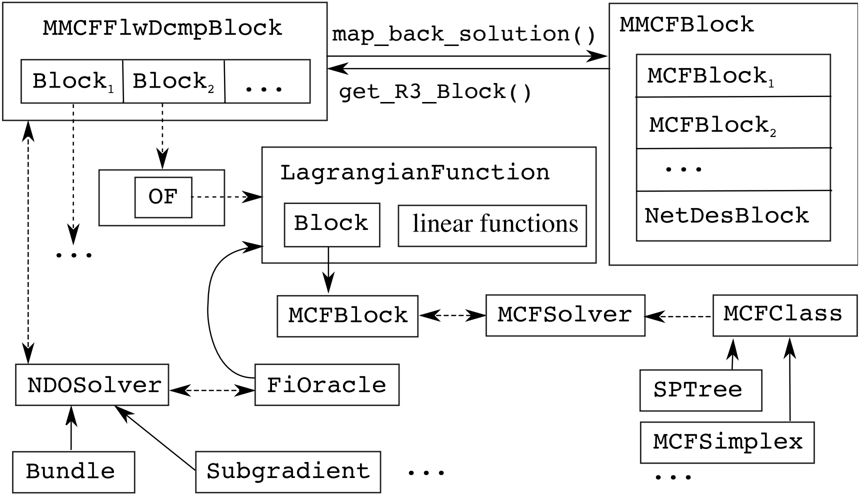
LagrangianFunction

- C05Function::LagrangianFunction has one isolated Block + set of (so far) LinearFunction to define Lagrangian term
- evaluate() = Block.get_registered_solvers()[i].solve(): asynchronous Solver $\Longrightarrow$ 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

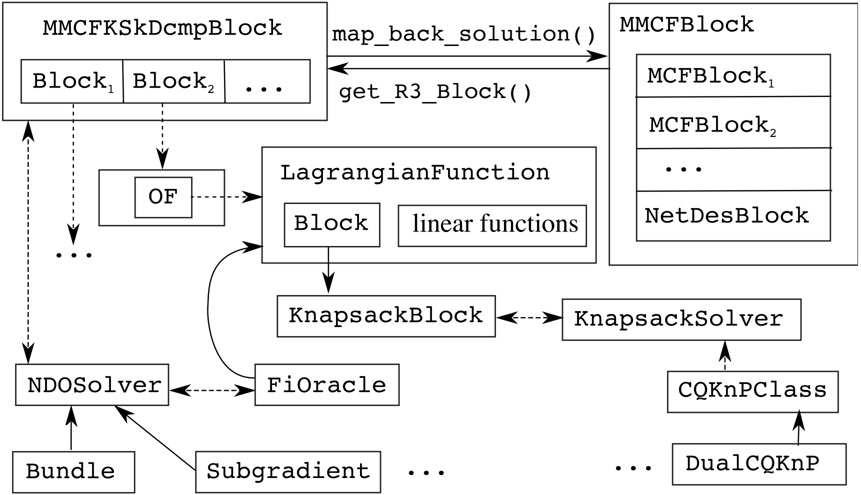
- `un_any_thing()` template functions/macros to extract (`std::vector` or `boost::multi_array` of) (`std::list` of) `Variable`/`Constraints` out of a `boost_any` and work on that
- `Solution::ColVariableSolution` uses the abstract representation of any `Block` that only have (`std::vector` or `boost::multi_array` 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

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: 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

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