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

Antonio Frangioni

Dipartimento di Informatica, Università di Pisa

AIRO PhD School 2021
&
5th AIRO Young Workshop

“Napoli” — February 9, 2021

Meta–Outline

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

Outline – Parts III & IV

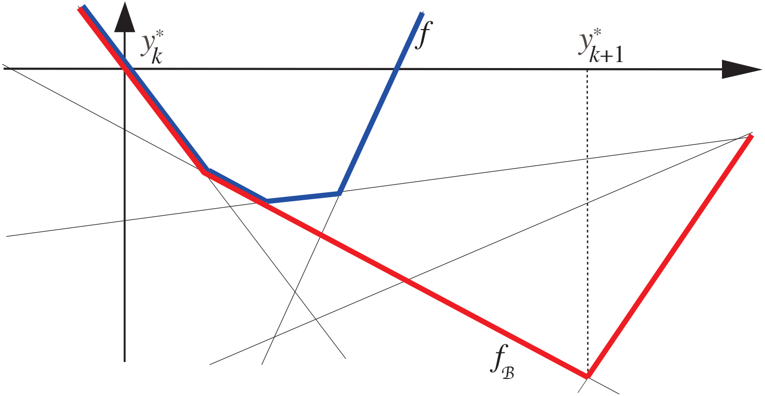
1. Stabilization
2. Dual-Optimal Cuts
3. Cuts Selection
4. Disaggregated Model
5. Easy Components
6. Structured Decomposition
7. Incremental, Inexact, Asynchronous
8. A Useful Companion on the Road
9. Conclusions (for good)

Part III:
Many Twists and Turns

Stabilization

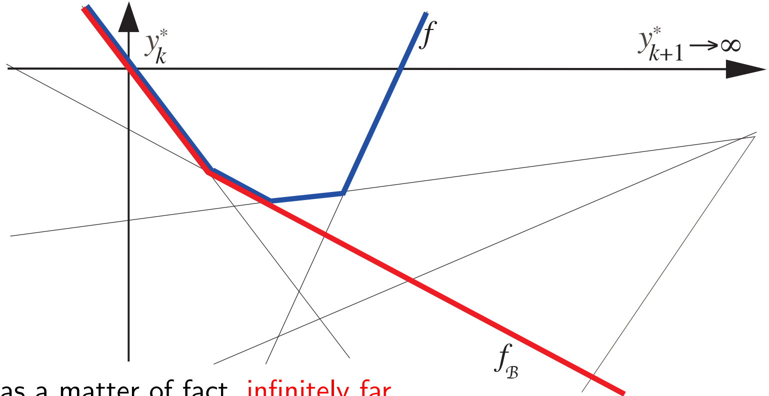
Issue with the Cutting-Plane approach: instability

- y^*_{k+1} can be very far from y^*_k, where $f_{\mathcal{B}}$ is a "bad model" of f



Issue with the Cutting-Plane approach: instability

- y^*_{k+1} can be very far from y^*_k, where $f_{\mathcal{B}}$ is a "bad model" of f



. . . as a matter of fact, infinitely far

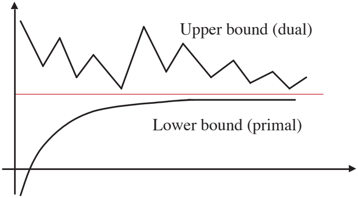
- $(\Pi_{\mathcal{B}})$ empty $\equiv$ $(\Delta_{\mathcal{B}})$ unbounded $\Rightarrow$ Phase 0 / Phase 1 approach

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

The effects of instability

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

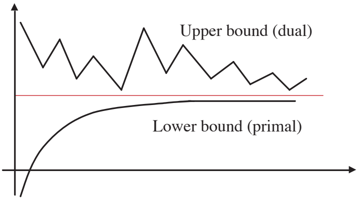
$\Longrightarrow$ tailing off, slow convergence



The effects of instability

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

  $\Longrightarrow$ tailing off, slow convergence



- The solution is pretty obvious: stabilize it
- Gedankenexperiment: starting from known dual optimum, constrain duals in a box of given width

width	time		iter.		columns	
∞	4178.4	%	509	%	37579	%
200.0	835.5	20.0	119	23.4	9368	24.9
20.0	117.9	2.8	35	6.9	2789	7.4
2.0	52.0	1.2	20	3.9	1430	3.8
0.2	47.5	1.1	19	3.7	1333	3.5

Works wonders! . . .

Stabilizing DW/Lagrange/CG

…if only we knew the dual optimum! (which we don't)

- Current point $\bar{y}$, box of size $t > 0$ (how chosen??) around it
- Stabilized dual master problem[34]

$$(\Delta_{\mathcal{B},\bar{y},t}) \qquad \min \left\{ f_{\mathcal{B}}(\,\bar{y} + d\,) \;:\; \|\, d\,\|_\infty \leq t \,\right\} \tag{1}$$

- Corresponding stabilized primal master problem

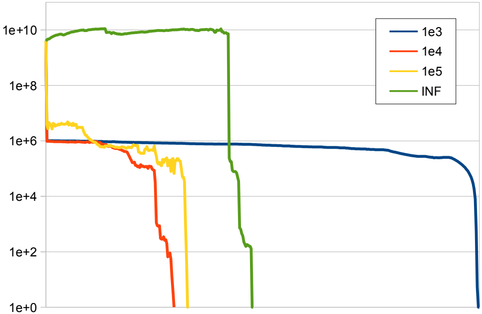
$$(\Pi_{\mathcal{B},\bar{y},t}) \qquad \max \left\{ cx + \bar{y}s - t\|\, s\,\|_1 \;:\; s = b - Ax \;,\; x \in conv(\mathcal{B}) \,\right\}$$

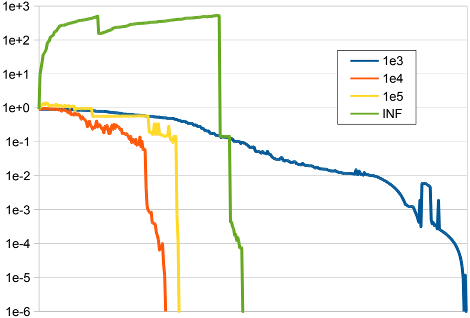
  i.e., just Dantzig-Wolfe with slacks (s)

- When $f(\bar{y} + d^*) \ll f(\bar{y})$, move $\bar{y} = \bar{y} + d^*$ ("serious step")
- Uses just LP tools, relatively minor modifications to $(\Delta_{\mathcal{B}})$
- Does this really work?

[34] Marsten, Hogan, Blankenship "The Boxstep Method for Large-scale Optimization" *Op. Res.*, 1975

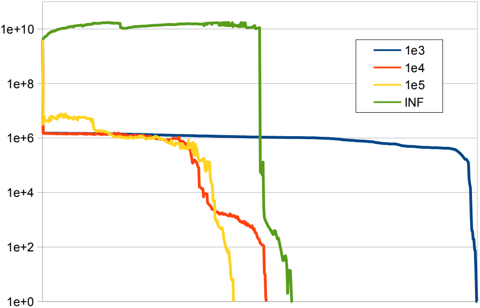
Computational results of the boxstep method (pds7)

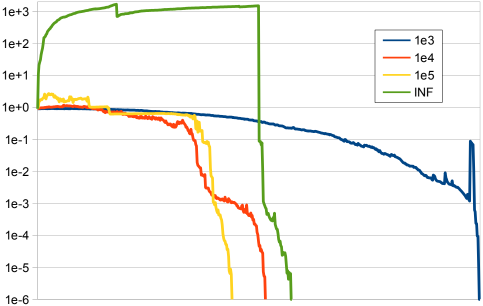




- Pure multicommodity flow instance (no design)
- Left = distance from final dual optimum
  right = relative gap with optimal value
- Stabilized with (fixed) different t, un-stabilized ($t = \infty$)
- One can clearly over-stabilize

Computational results of the boxstep method (pds18)

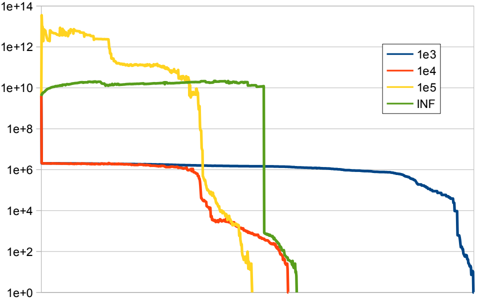




- All cases show a "combinatorial tail" where convergence is very quick
- $t = 1\text{e}{+3}$: "smooth but slow" until the combinatorial tail kicks in, a short-step approach not unlike subgradient methods[35]
- $t = \infty$: apparently trashing along until some magic threshold is hit
- "intermediate"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

[35] Camerini, Fratta, Maffioli "On Improving Relaxation Methods by Modified Gradient Techniques" *Math. Prog. Study*, 1975

Computational results of the boxstep method (pds30)



- $t = $ 1e+5: initially even worse than $t = \infty$ but ends up faster
- Clearly, some on-line tuning of t would be appropriate
- Perhaps a different stabilizing term would help? Why not[36]

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

- "Because it's not LP" $\Longrightarrow$ a different duality need be used

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

Generalized proximal/trust region stabilization

- General stabilizing term $\mathcal{D}$, stabilized dual problem

$$(\Delta_{\bar{y},\mathcal{D}}) \qquad \phi_{\mathcal{D}}(\bar{y}) = \min \{ f(\bar{y} + d) + \mathcal{D}(d) \}$$

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

- Stabilized primal problem = Fenchel's dual of $(\Delta_{\bar{y},\mathcal{D}})$

$$(\Pi_{\bar{y},\mathcal{D}}) \qquad \min \{ f^*(s) - s\bar{y} + \mathcal{D}^*(-s) \}$$

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

- For our dual f, a generalized augmented Lagrangian

$$\max \{ cx + \bar{y}(b - Ax) - \mathcal{D}^*(Ax - b) \; : \; x \in conv(X) \}$$

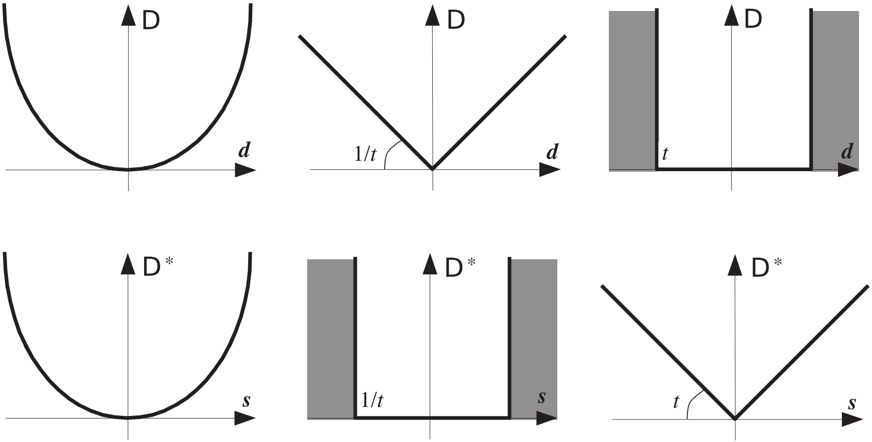
- A "primal" exists even for a non-dual f: $v(\Pi) = -f^*(0) = v(\Delta)$ for

$$(\Pi) \qquad \max\{ -f^*(s) \; : \; s = 0 \}$$

- General theory exist[37], but never mind

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

Classical stabilizing terms



$$\mathcal{D} = \frac{1}{2t}\|\cdot\|_2^2 \qquad \mathcal{D} = \frac{1}{t}\|\cdot\|_1 \qquad \mathcal{D} = I_{B_\infty(t)}$$

$$\mathcal{D}^* = \frac{1}{2}t\|\cdot\|_2^2 \qquad \mathcal{D}^* = I_{B_\infty(1/t)} \qquad \mathcal{D}^* = t\|\cdot\|_1$$

Fancier stabilizing terms (very nonlinear)

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

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

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

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

- Bregman functions[39]

$$\mathcal{D}_{\bar{y}}(d) = (\, \psi(\bar{y} + d) - \psi(\bar{y}) - \nabla\psi(\bar{y})d \,)$$

with ψ fixed, strictly convex, differentiable, with compact level sets

- Others (φ-divergences, ...), all "very nonlinear"

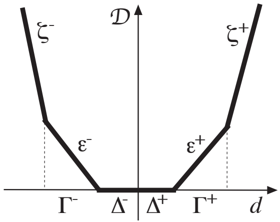
---

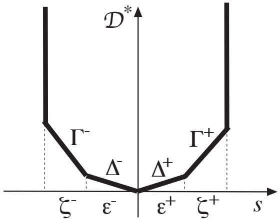
[38] Pinar, Zenios "Parallel Decomposition of Multicommodity [...] Using a Linear-Quadratic [...]" *ORSA J. Comp.*, 1992

[39] Chen, Teboulle "Convergence Analysis of a Proximal-like Minimization Algorithm Using Bregman Functions" *SIOPT*, 1993

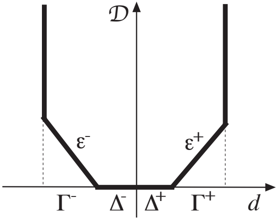
A 5-piecewise-linear function

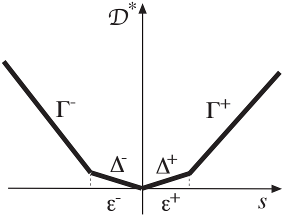
Trust region on $\bar{y}$ + small penalty close + much larger penalty farther[40]





Slightly simplified version: only 3 pieces.





[40] Ben Amor, Desrosiers, F. "On the Choice of Explicit Stabilizing Terms in Column Generation" *DAM*, 2009

A 5-piecewise-linear master problem

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

A 5-piecewise-linear master problem

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

- Same constraints as $(\Pi_{\mathcal{B}})$, 4 slack variables for each constraint
- Many parameters: *widths* $\Gamma^\pm$ and $\Delta^\pm$, *penalties* $\zeta^\pm$ and $\varepsilon^\pm$, different roles for small and large penalties
- Large penalties $\zeta^\pm$ easily make $(\Delta_{\mathcal{B},\bar{y},\mathcal{D}})$ bounded $\Longrightarrow$ no Phase 0
- 3-pieces: either large penalty $\Longrightarrow$ small moves, or
  small penalty $\Longrightarrow$ instability

On unboundedness and early termination

- A ray χ of X: $x \in X \Longrightarrow x + \lambda\chi \in X$ for $\lambda \to \infty \Longrightarrow$ $(c - yA)\chi > 0 \Longrightarrow f(y) = \infty \Longrightarrow$ constraint $cr \leq y(A\chi)$ in the dual
- One might even hide the convexity constraint:
  - $A\bar{x} \to \left[\ A\bar{x}\ ,\ 1\ \right] \quad , \quad b \to \left[\ b\ ,\ 1\ \right]$;
  - Ignoring the special role of v (just another y)
  - Advantage: everything is a constraint

On unboundedness and early termination

- A ray χ of X: $x \in X \Longrightarrow x + \lambda\chi \in X$ for $\lambda \to \infty \Longrightarrow$ $(c - yA)\chi > 0 \Longrightarrow f(y) = \infty \Longrightarrow$ constraint $cr \leq y(A\chi)$ in the dual
- One might even hide the convexity constraint:
  - $A\bar{x} \to \left[\, A\bar{x} \,,\, 1 \,\right] \quad , \quad b \to \left[\, b \,,\, 1 \,\right]$;
  - Ignoring the special role of v (just another y)
  - Advantage: everything is a constraint

$\boxed{\text{This is a bad idea!}}$

On unboundedness and early termination

- A ray χ of X: $x \in X \Longrightarrow x + \lambda\chi \in X$ for $\lambda \to \infty \Longrightarrow$
  $(c - yA)\chi > 0 \Longrightarrow f(y) = \infty \Longrightarrow$ constraint $cr \leq y(A\chi)$ in the dual
- One might even hide the convexity constraint:
  - $A\bar{x} \to [\ A\bar{x}\ ,\ 1\ ]\quad , \quad b \to [\ b\ ,\ 1\ ]$;
  - Ignoring the special role of v (just another y)
  - Advantage: everything is a constraint

$\boxed{\text{This is a bad idea!}}$

- Moving $\bar{y}$ requires testing for decrease in f-value, but when a ray is generated, $f(\,\bar{y} + d^*\,) = \infty$
- Ignoring convexity constraint $\Longrightarrow$ Proximal Point: solve the problem exactly for $\bar{y}$ before moving it
- Convexity constraints are good: invent them if they are not there

A Glimpse to Computational Results

- State-of-the-art `GenCol` code, large-scale, difficult MDVS instances (only root relaxation times)
- 5-pieces better than 3-pieces, 5-then-3 even better
- Quadratic more "stable", but optimized 5-pieces always faster (quadratic has far less parameters, easier but less flexible)
- Comparing 5-piecewise with (BP) or without (PP) early termination

		p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
time	CG	139	177	235	159	3138	3966	3704	1742	3685	3065
	PP	33	36	38	28	482	335	946	572	1065	2037
	BP	26	28	35	21	295	257	639	352	545	1505
iter	CG	117	149	200	165	408	524	296	186	246	247
	PP	47	47	49	45	93	64	98	83	86	150
	BP	37	43	44	36	57	53	59	49	51	101
mpt	CG	88	125	165	105	1679	2004	1955	925	1984	1743
	PP	13	16	17	10	189	128	428	257	542	1326
	BP	10	14	15	10	100	70	329	206	334	983

- Stabilization works well, approximate stabilization works better

Other Forms of Stabilization

- Proximal level[41]: closest point promising given amount of decrease

$$(\Delta_{\mathcal{B},\bar{y},l}) \qquad \min \left\{ \, \tfrac{1}{2t} \| \, d \, \|_2^2 \; : \; f_{\mathcal{B}}(\, \bar{y} + d \,) \leq f(\, \bar{y} \,) - l \, \right\} \qquad (2)$$

- l somehow easier to manage than t, easy rules available that allow to keep $\bar{y}$ fixed (but possible in proximal, too)
- Trade blows in practice, but doubly-stabilised possible[42]
- Different approach: aim for center (analytic[43] or Chebychev[44]) of localization set $\mathcal{L} = \{ \, (\, y \, , \, v \,) \; : \; f_{\mathcal{B}}(\, y \,) \leq v \leq f(\, \bar{y} \,) \, \} \subset \mathbb{R}^{n+1}$
- "Good" theoretical performances, but in practice a penalty term is still required[45]

---

[41] Lemaréchal, Nemirovskii, Nesterov "New Variants of Bundle Methods" *Math. Prog.*, 1995

[42] de Oliveira, Solodov "A Doubly-Stabilized Bundle Method for Nonsmooth Convex Optimization" *Math. Prog.*, 2016

[43] Gondzio, González-Brevis, Munari "New Developments in the Primal-Dual Column Generation Technique" *EJOR*, 2013

[44] Ouorou "A proximal cutting plane method using Chebychev center for nonsmooth convex optimization" *Math. Prog.*, 2009

[45] Babonneau, Beltran, Haurie, Tadonki, Vial "Proximal-ACCPM: a Versatile Oracle-Based [...]" *Adv. Comp. Man. Sci.* 2007

From Minimally to Maximally Intrusive Stabilization

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

[46] Ben-Ameur, Neto "Acceleration of Cutting-Plane and Column Generation Algorithms [. . .]" *Networks*, 2007

[47] Pessoa, Sadykov, Uchoa, Vanderbeck "Automation [. . .] of [. . .] Stabilization [. . .] in Column Generation" *IJoC*, 2018

[48] Mifflin, Sagastizábal "A $\mathcal{VU}$-algorithm for Convex Minimization" *Math. Prog.*, 2005.

[49] Helmberg, Pichler "Dynamic Scaling and Submodel Selection in Bundle Methods for Convex Optimization" *OO*, 2017

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

Stabilized Benders' Decomposition

- Stabilized master problem easy to do: with trust region

$$(B_{\mathcal{B},\bar{x},t}) \qquad \min \{ \, v_{\mathcal{B}}(\, x \,) \; : \; \| \, x - \bar{x} \, \|_{\infty} \leq t \; , \; x \in X \, \}$$

  pretty identical to (1) (no dual, though)
- For $X \subseteq \{ \, 0 \, , \, 1 \, \}^n$, local branching constraint

$$\textstyle\sum_{i \,:\, \bar{x}_i = 1} (1 - x_i) + \sum_{i \,:\, \bar{x}_i = 0} x_i \leq t$$

- However, $x^* = \bar{x}$ only $\Longrightarrow \bar{x}$ local optimum (nonconvex)
  $\Longrightarrow$ have to increase t until $t = n$ (∞)
- Silver lining: reverse box $\| \, x - \bar{x} \, \|_{\infty} \geq t$ (nonconvex) now easy
- Level stabilization a-la (2) also possible[51], pros and cons:
  $(B_{\mathcal{B},\bar{x},l})$ can be solved inexactly (but larger and more difficult),
  l easier to manage than t and need not go ∞ (but no reverse box)
- All in all it does work[52] (but nontrivial)

[51] van Ackooij, F., de Oliveira "Inexact Stabilized Benders' Decomposition Approaches [...]" *COAP*, 2016
[52] Baena, Castro, F. "Stabilized Benders Methods for Large-scale Combinatorial Optimization [...]" *Man. Sci.*, 2020

Dual-Optimal Cuts

Dual-Optimal Cuts

- Stabilizing = restricting the dual space
- The above approaches need stability center $\bar{y}$, to be updated: it'd be nice if we could do without
- Simple observation: dual constraints = primal variables
  $\Longrightarrow$ need to add even more variables to the primal

  ... in such a way that not all dual optimal solution are cut

Dual-Optimal Cuts

- Stabilizing = restricting the dual space
- The above approaches need stability center $\bar{y}$, to be updated: it'd be nice if we could do without
- Simple observation: dual constraints = primal variables
  $\Longrightarrow$ need to add even more variables to the primal

  ... in such a way that not all dual optimal solution are cut
- Actually quite simple:
  the new variables must not add new primal solutions[53]

[53] Ben Amor, Desrosiers, Valério de Carvalho "Dual-optimal Inequalities for Stabilized Column Generation" *Op. Res.*, 2006

Dual-Optimal Cuts for Multicommodity flows

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

- Constraints become

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

  where "$-$" if (i,j) is reversed in c; hence, one also needs

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

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

Dual-Optimal Cuts for Multicommodity flows

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

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

where "$-$" if (i,j) is reversed in c; hence, one also needs

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

- Any feasible solution to the extended model can be converted into a feasible solution to the original model
- $|\mathcal{C}| \in O(n^2)$ if G is planar, all-pairs SPT pricing otherwise
- Some good results, other applications (Cutting Stock, different cuts)

Cuts Selection

(Feasibility) Cuts Selection

- $v(x) = -\infty \Longrightarrow$ any $\bar{\omega} \in W_\infty$ gives a cut: which one is "best"?
- If LP solver choses, can't expect it to pick a "good one"
- $(\, x^*\,,\, v^*\,)$ solution of $(B_{\mathcal{B}})$: a cut does not $\exists \iff$

$$v(x^*) = \max\{\, ez \;:\; Ez \leq d - Dx \,\} \geq v^*$$

$$\equiv \max\{\, 0z \;:\; ez^* \geq v^* \,,\; Ez^* \leq d - Dx^* \,\} = 0$$

- Hence a cut does $\exists \iff$

$$\min\{\, w(d - Dx^*) - w_0 v^* \;:\; wE = w_0 e \,,\; (\, w \,,\, w_0 \,) \geq 0 \,\} = -\infty$$

$\equiv$ (homogeneity)

$$0 > \min\; w(d - Dx^*) - w_0 v^*$$

$$wE = w_0 e \,,\; w\beta + w_0\beta_0 = 1 \,,\; (\, w \,,\, w_0 \,) \geq 0$$

however chosen $(\, \beta \,,\, \beta_0 \,)$: a proper choice improves performances[54]

[54] Fischetti, Salvagnin, Zanette "A Note on the Selection of Benders' Cuts" *Math. Prog.*, 2010

Disaggregated Model

Disaggregated Model for the Block-diagonal Program

- The real decomposition case:

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

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

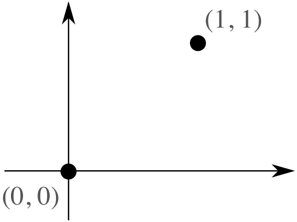
- Disaggregated DW reformulation:

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

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

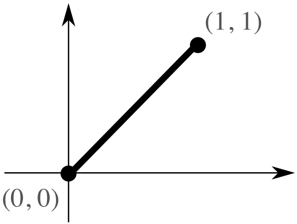
- A different multiplier $\theta^k_{\bar{x}}$ for each $k \in K$: aggregated is $\theta^k_{\bar{x}} = \theta^h_{\bar{x}}$ for $h \neq k \Longrightarrow$ a restriction (less solutions $\equiv$ bad)

Geometry of Disaggregated Models



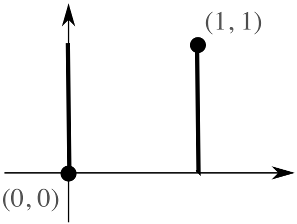
- Given X,

Geometry of Disaggregated Models



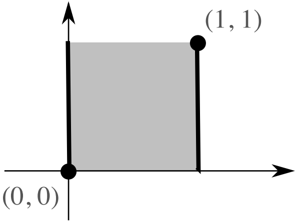
- Given X, taking the convex hull of Cartesian products

Geometry of Disaggregated Models



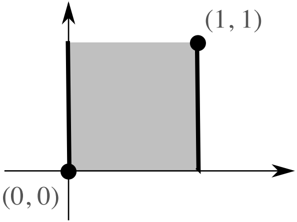
- Given X, taking the convex hull of Cartesian products is smaller (bad) than first making convex hulls

Geometry of Disaggregated Models



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

Geometry of Disaggregated Models



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

- From the dual viewpoint

$$f_{\mathcal{B}}(\,y\,) = \sum_{k \in K} f^k_{\mathcal{B}}(\,y\,)$$

the sum of individual models is better than the model of the sum

Disaggregated Dantzig-Wolfe and Multicommodity flows

- Aggregated DW: $\mathcal{S} = \{$ (extreme) flows $s = [\bar{x}^{1,s}, \ldots, \bar{x}^{k,s}]$ $\}$

$$
\begin{array}{lll}
\min & \sum_{s \in \mathcal{S}} \big(\sum_{k \in K} \sum_{(i,j) \in A} c^k_{ij} \bar{x}^{k,s}_{ij} \big) \theta_s & \\
& \sum_{s \in \mathcal{S}} \big(\sum_{k \in K} \bar{x}^{k,s}_{ij} - u_{ij} \big) \theta_s \leq 0 & (i,j) \in A \\
& \sum_{s \in \mathcal{S}} \theta_s = 1 \quad , \quad \theta_s \geq 0 & s \in \mathcal{S}
\end{array}
$$

- Disaggregated + scaling $\equiv$ arc-path formulation:

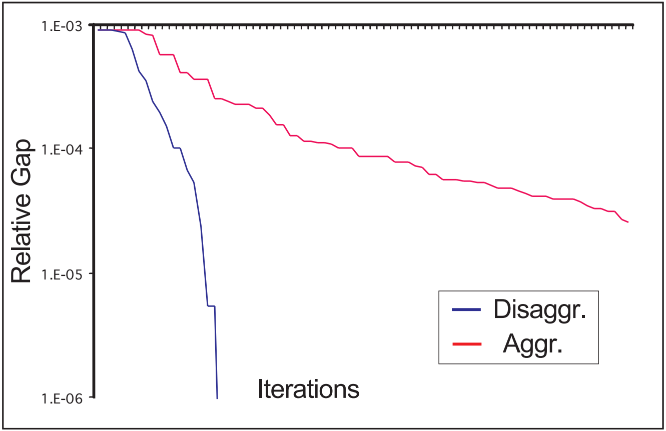
  $p \in \mathcal{P}^k = \{$ s^k–t^k paths $\}$, c_p cost, $f_p (= d^k \theta^k_s)$ flow, $\mathcal{P} = \cup_{k \in K} \mathcal{P}^k$

$$
\begin{array}{lll}
\min & \sum_{p \in \mathcal{P}} c_p f_p & \\
& \sum_{p \in \mathcal{P} \, : \, (i,j) \in p} f_p \leq u_{ij} & (i,j) \in A \\
& \sum_{p \in \mathcal{P}^k} f_p = d^k & k \in K \\
& f_p \geq 0 & p \in \mathcal{P}
\end{array}
$$

- More columns but sparser, (a few) more rows, much more efficient[55]
- Master problem size $\approx$ time increases, but convergence speed more so

[55] Jones, Lustig, Farwolden, Powell "Multicommodity Flows: the Impact of Formulation on Decomposition" *Math. Prog.* 1993

Disaggregated decomposition



- Easily extended to any decomposable X[15]
- Stabilized versions immediate

More or Less Disaggregated?

- That was ≈ 30 years ago with $|K| \approx 10$, still true if $|K| \approx 10000$?
- Aggregation is arbitrary, then why "all or nothing"?
- Partition $C = (C_1, C_2, \ldots, C_h)$ of K, partially aggregated model $f_{\mathcal{B}}^{C}$ with h components $f_{\mathcal{B}}^{i}$, each the sum over one C_i
- Basically, $\theta_s^k = \theta_s^h$ only for each $(h, k) \in C_i \times C_i$
- Exploring the trade-off between master problem size $\Longrightarrow$ time and iterations, subproblems remain the same
- How to choose the C_i? In general open problem
- Aggregation can be dynamic[56], even more open problem, but it can work[49]

---

[56] van Ackooij, F. "Incremental Bundle Methods Using Upper Models" *SIOPT*, 2018

Easy Components

Decomposition of Multicommodity Network Design

- Multicommodity flow + arc design costs f_{ij} ($z_{ij} \in \{0, 1\}$)
- $\mathcal{S}$ = extreme points of z ($2^{|A|}$ vertices of the unitary hypercube):

$$
\begin{array}{lll}
\min & \sum_{p \in \mathcal{P}} c_p f_p + \sum_{s \in \mathcal{S}} \left(\sum_{(i,j) \in A} f_{ij} \bar{z}^s_{ij} \right) \theta_s & \\
 & \sum_{p \in \mathcal{P} \,:\, (i,j) \in p} f_p \leq u_{ij} \sum_{s \in \mathcal{S}} \bar{z}^s_{ij} \theta_s & (i,j) \in A \\
 & \sum_{p \in \mathcal{P}^k} f_p = d^k & k \in K \\
 & f_p \geq 0 & p \in \mathcal{P} \\
 & \sum_{s \in \mathcal{S}} \theta_s = 1 \quad , \quad \theta_s \geq 0 & s \in \mathcal{S}
\end{array}
$$

- Are you sure you're sane? Arguably not:

  replacing a $2n$ formulation with a 2^n one!

- . . . and with very long, dense rows

Multicommodity Network Design, the Right Way

- The unitary hypercube is a cartesian product: why not $\mathcal{S}^{ij} = \{0, 1\}$?
- $z_{ij} \;\longrightarrow\; 0 \cdot \theta^{ij,0} + 1 \cdot \theta^{ij,1} \;,\; \theta^{ij,0} + \theta^{ij,1} = 1 \;,\; \theta^{ij,0} \geq 0 \;,\; \theta^{ij,1} \geq 0.$

$$\boxed{z_{ij} \in [\, 0 \,,\, 1 \,]}$$

Multicommodity Network Design, the Right Way

- The unitary hypercube is a cartesian product: why not $\mathcal{S}^{ij} = \{0, 1\}$?
- $z_{ij} \;\longrightarrow\; 0 \cdot \theta^{ij,0} + 1 \cdot \theta^{ij,1} \;,\; \theta^{ij,0} + \theta^{ij,1} = 1 \;,\; \theta^{ij,0} \geq 0 \;,\; \theta^{ij,1} \geq 0.$

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

- Arc-path formulation with original arc design variables

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

- Only generate the right variables

Is it always this easy?

- No: what if one had, say,

$$\sum_{(i,j)\in A} z_{ij} \leq r \qquad ?$$

- Design subproblem can no longer be disaggregated
- But, one could write the arc-path formulation in that case, too
- And could add that constraint to the master problem
- Can this be stabilized? Of course it can[57]

[57] F., Gorgone "Bundle Methods for Sum-Functions With "Easy" Components: [...] Network Design" *Math. Prog.*, 2014

Stabilized decomposition with "easy components"

- f Lagrangian function of structured optimization problem

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

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

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

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

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

  "easy because a compact convex formulation is known"

Stabilized decomposition with "easy components"

- f Lagrangian function of structured optimization problem

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

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

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

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

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

  "easy because a compact convex formulation is known"

- Usual approach: disregard differences

  Better idea: treat "easy" components specially

- In practice: insert "full" description of f^2 in the master problem

- Master problem size may increase (at the beginning), but "perfect" information is known

- Dual master problem: abstract form

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

- Primal master problem: abstract and implementable form

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

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

- Barring some details (do not translate $f^1_{\mathcal{B}}$), everything works
- Performances can improve dramatically (not hard to see why)

A Glimpse to Computational Results

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

[58] F. "Solving Semidefinite Quadratic Problems Within Nonsmooth Optimization Algorithms" *Comput. & O.R.*, 1996

The Easy Component Need Not Be Linear

- Nonlinear multicommodity routing:

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

  with classical (convex) Kleinrock delay function

- Decomposes into $|K|$ flows $+ |A|$ simple convex subproblems

- Specialized models of $|A|$ convex functions using the conjugate

- Specialized treatment of these "easy" C^2 functions with Newton model instead of the cutting-plane model[59]

- Substantially improved performances

[59] Lemaréchal, Ororou, Petrou "A Bundle-type Algorithm for Routing in Telecommunication Data Networks" *COAP*, 2009

Structured Decomposition

The Structured Dantzig-Wolfe Idea

- Assumption 1: Alternative Formulation of "easy" set

$$conv(X) = \{\ x = C\theta \ :\ \Gamma\theta \leq \gamma\ \}$$

The Structured Dantzig-Wolfe Idea

- Assumption 1: Alternative Formulation of "easy" set

$$conv(X) = \{\ x = C\theta \ :\ \Gamma\theta \leq \gamma\ \}$$

- Assumption 2: padding with zeroes

$$\Gamma_{\mathcal{B}}\bar{\theta}_{\mathcal{B}} \leq \gamma_{\mathcal{B}} \quad \Rightarrow \Gamma[\ \bar{\theta}_{\mathcal{B}}\ ,\ 0\ ] \leq \gamma$$

$$\Rightarrow X_{\mathcal{B}} = \left\{\ x = C_{\mathcal{B}}\theta_{\mathcal{B}} \ :\ \Gamma_{\mathcal{B}}\theta_{\mathcal{B}} \leq \gamma_{\mathcal{B}}\ \right\} \subseteq conv(X)$$

The Structured Dantzig-Wolfe Idea

- Assumption 1: Alternative Formulation of "easy" set

$$conv(X) = \{ \; x = C\theta \; : \; \Gamma\theta \leq \gamma \; \}$$

- Assumption 2: padding with zeroes

$$\Gamma_{\mathcal{B}}\bar{\theta}_{\mathcal{B}} \leq \gamma_{\mathcal{B}} \quad \Rightarrow \Gamma \left[\; \bar{\theta}_{\mathcal{B}} \; , \; 0 \; \right] \leq \gamma$$

$$\Rightarrow X_{\mathcal{B}} = \left\{ \; x = C_{\mathcal{B}}\theta_{\mathcal{B}} \; : \; \Gamma_{\mathcal{B}}\theta_{\mathcal{B}} \leq \gamma_{\mathcal{B}} \; \right\} \subseteq conv(X)$$

- Assumption 3: easy update of rows and columns

Given $\mathcal{B}$, $\bar{x} \in conv(X)$, $\bar{x} \notin X_{\mathcal{B}}$, it is "easy" to find $\mathcal{B}' \supset \mathcal{B}$

($\Rightarrow \Gamma_{\mathcal{B}'}$, $\gamma_{\mathcal{B}'}$) such that $\exists \; \mathcal{B}'' \supseteq \mathcal{B}'$ such that $\bar{x} \in X_{\mathcal{B}''}$.

The Structured Dantzig-Wolfe Idea

- Assumption 1: Alternative Formulation of "easy" set

$$conv(X) = \{\ x = C\theta \ :\ \Gamma\theta \leq \gamma\ \}$$

- Assumption 2: padding with zeroes

$$\Gamma_{\mathcal{B}}\bar{\theta}_{\mathcal{B}} \leq \gamma_{\mathcal{B}} \ \ \Rightarrow \Gamma\big[\ \bar{\theta}_{\mathcal{B}}\ ,\ 0\ \big] \leq \gamma$$

$$\Rightarrow X_{\mathcal{B}} = \left\{\ x = C_{\mathcal{B}}\theta_{\mathcal{B}} \ :\ \Gamma_{\mathcal{B}}\theta_{\mathcal{B}} \leq \gamma_{\mathcal{B}}\ \right\} \subseteq conv(X)$$

- Assumption 3: easy update of rows and columns

Given $\mathcal{B}$, $\bar{x} \in conv(X)$, $\bar{x} \notin X_{\mathcal{B}}$, it is "easy" to find $\mathcal{B}' \supset \mathcal{B}$

$(\Rightarrow \Gamma_{\mathcal{B}'},\ \gamma_{\mathcal{B}'})$ such that $\exists\ \mathcal{B}'' \supseteq \mathcal{B}'$ such that $\bar{x} \in X_{\mathcal{B}''}$.

- Structured master problem

$$(\Pi_{\mathcal{B}}) \qquad \max\left\{\ cx \ :\ Ax = b\ ,\ x = C_{\mathcal{B}}\theta_{\mathcal{B}}\ ,\ \ \Gamma_{\mathcal{B}}\theta_{\mathcal{B}} \leq \gamma_{\mathcal{B}}\ \right\} \qquad (3)$$

$\equiv$ structured model

$$f_{\mathcal{B}}(y) \ = \ \max\{\ (c - yA)x + xb \ :\ x = C_{\mathcal{B}}\theta_{\mathcal{B}}\ ,\ \Gamma_{\mathcal{B}}\theta_{\mathcal{B}} \leq \gamma_{\mathcal{B}}\ \} \qquad (4)$$

The Structured Dantzig-Wolfe Algorithm

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

The Structured Dantzig-Wolfe Algorithm

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

- Easy[60] to prove that:
  - finitely terminates with an optimal solution of (Π)
  - ... even if (proper) removal from $\mathcal{B}$ is allowed (when cx^* increases)
  - ... even if X is non compact and $\mathcal{B} = \emptyset$ at start (Phase 0)

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

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

[60] F., Gendron "0-1 reformulations of the multicommodity capacitated network design problem" *DAM*, 2009

Stabilizing the Structured Dantzig-Wolfe Algorithm

- Exactly the same as stabilizing DW: stabilized master problem

$$(\Delta_{\mathcal{B},\bar{y},\mathcal{D}}) \qquad \min\left\{ f_{\mathcal{B}}(\,\bar{y}+d\,) + \mathcal{D}(\,d\,) \right\}$$

  except $f_{\mathcal{B}}$ is a different model of f (not the cutting plane one)

- Even simpler from the primal viewpoint:

$$\max\left\{ cx + \bar{y}s - \mathcal{D}^*(-s) \,:\, s = b - Ax\,,\ x = C_{\mathcal{B}}\theta_{\mathcal{B}}\,,\ \Gamma_{\mathcal{B}}\theta_{\mathcal{B}} \leq \gamma_{\mathcal{B}} \right\}$$

- With proper choice of $\mathcal{D}$, still a Linear Program; e.g.

$$\begin{aligned} \max \quad & \ldots - (\Delta^- + \Gamma^-)s''_- - \Delta^- s'_- - \Delta^+ s'_+ - (\Delta^+ + \Gamma^+)s''_+ \\ & s''_- + s'_- - s'_+ - s''_+ = b - Ax \ ,\ \ldots \\ & s''_+ \geq 0 \ ,\ \varepsilon^+ \geq s'_+ \geq 0 \ ,\ \varepsilon^- \geq s'_- \geq 0 \ ,\ s''_- \geq 0 \end{aligned}$$

  dual optimal variables of "$s = b - Ax$" still give d^*, ...

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

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

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

Incremental, Inexact, Asynchronous

Incremental Computation of Subproblems

- (Partial) aggregation can contribute to reducing master problem cost but subproblem cost remains the same
- Subproblem cost high if $|K|$ large and/or subproblems hard, trade-off very application-dependent (you get to meet all sorts)
- Clearly interesting to avoid "unnecessary" subproblems computations
- In fact quite easy to understand early on if $f(\bar{y} + d^*) \not\ll f(\bar{y})$ "null steps" can be declared without computing all subproblems[62]
- Early declaring "serious steps" harder, but possible[56] provided you can estimate the Lipchitz constant (nontrivial)
- Trade-off still all to explore.

[62] Gaudioso, Giallombardo, Miglionico "On Solving the Lagrangian Dual [...] via an Incremental Approach" *COAP*, 2007

Inexact Computation of Subproblems

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

---

[63] de Oliveira, Sagastizábal, C. Lemaréchal "Convex Proximal Bundle Methods [...] for Inexact Oracles" *Math. Prog.*, 2014
[64] de Oliveira, Sagastizábal "Level Bundle Methods for Oracles With On-Demand Accuracy" *OM&S*, 2014
[65] Schramm, Zowe. "A Version of the Bundle Idea for Minimizing a Nonsmooth Function [...] *SIOPT*, 1992

Asynchronous Computation of Subproblems

- Clear avenue to reduce wall-clock time: parallelize subproblems
- Master-slave version "obvious"[38], popular for stochastic programs[66]
- Runs afoul of Amdahl's Law: speedup limited by master problem cost and large master problems is what works best (most often)
- May use specialised algorithms[58] and hardware[6], but issue remains
- Completely asynchronous versions possible[67]
- Still to be completed (proximal? multiple masters?), general efficient implementations highly nontrivial
- Interesting variants for "loosely coupled subproblems"[68]

[66] Lubin, Martin, Petra, Sandıkçı "On Parallelizing Dual Decomposition in Stochastic Integer Programming" *O.R. Lett.*, 2013
[67] Iutzeler, Malick, de Oliveira "Asynchronous Level Bundle Methods" *Math. Prog.*, 2020
[68] Fischer, Helmberg "A Parallel Bundle Framework for Asynchronous Subspace Optimisation [. . .]" *SIOPT*, 2014

Part IV:
A Useful Companion on the Road

Decomposition in Practice

- Decomposition is complex, but so is any Branch-and-X
- Need general-purpose efficient decomposition software:
  - `Cplex` does Benders', structure automatic or user hints
  - `SCIP`[30] does B&C&P (one-level D-W), pricing & reformulation up to the user (plugins)
  - `GCG`[30] extends `SCIP` with automatic and user-defined (one-level) D-W and recently also a generic (one-level) Benders' approach[69]
  - D-W approaches for two-stage stochastic programs are implemented in `DDSIP`[70] and `PIPS`[71], the latter interfaced with `StructJuMP`[72]
  - The `BaPCoD` B&C&P code has been used to develop `Coluna.jl`[73], doing one-level D-W and (alpha) Benders', multi-level planned
- 4 years ago there was no multi-level, nor `C++`, so we started one

[69] Maher "Implementing the Branch-and-Cut approach for a general purpose Benders' decomposition framework" *EJOR*, 2021

[70] https://github.com/RalfGollmer/ddsip

[71] https://github.com/Argonne-National-Laboratory/PIPS

[72] https://github.com/StructJuMP/StructJuMP.jl

[73] https://github.com/atoptima/Coluna.jl

https://gitlab.com/smspp/smspp-project

Open source (LGPL3), public as of yesterday!

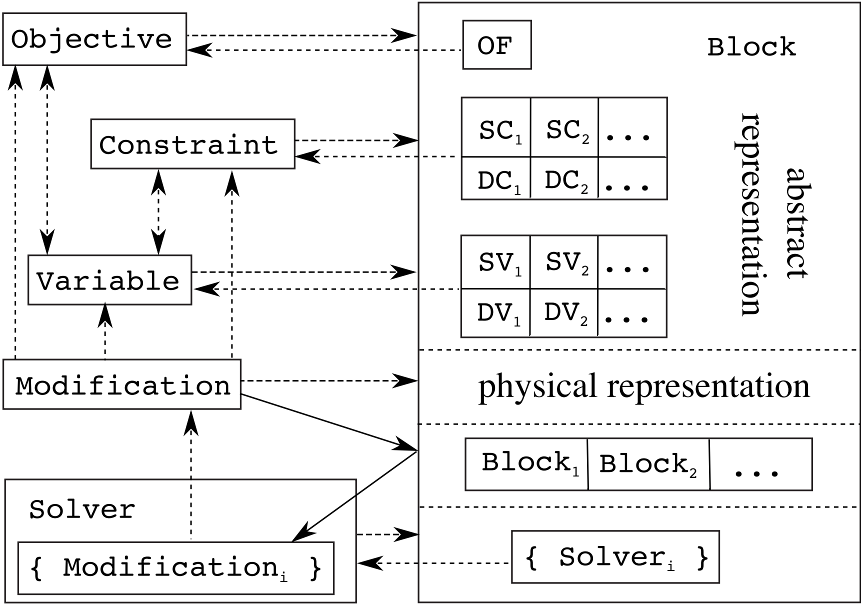
What SMS++ is

- A core set of C++-17 classes implementing a modelling system that:
  - explicitly supports the notion of Block $\equiv$ nested structure
  - separately provides "semantic" information from "syntactic" details (list of constraints/variables $\equiv$ one specific formulation among many)
  - allows exploiting specialised Solver on Block with specific structure
  - manages any dynamic change in the Block beyond "just" generation of constraints/variables
  - supports reformulation/restriction/relaxation of Block
  - has built-in parallel processing capabilities
  - should be able to deal with almost anything (bilevel, PDE, . . .)
- An hopefully growing set of specialized Block and Solver
- In perspective an ecosystem fostering collaboration and code sharing

What SMS++ is not

- An algebraic modelling language: `Block` / `Solver` are C++ code (although it provides some modelling-language-like functionalities)
- For the faint of heart: primarily written for algorithmic experts (although users may benefit from having many pre-defined `Block`)
- Stable: only version 0.4, lots of further development ahead, significant changes in interfaces not ruled out, actually expected (although current `Block` / `Solver` very thoroughly tested)
- Interfaced with many solvers: only `Cplex`, `SCIP`, `MCFClass`, `StOpt` (although the list should hopefully grow)

A Crude Schematic



Block

- Block = abstract class representing the general concept of "a (part of a) mathematical model with a well-understood identity"
- Each :Block a model with specific structure (e.g., MCFBlock:Block = a Min-Cost Flow problem)
- Physical representation of a Block: whatever data structure is required to describe the instance (e.g., G, b, c, u)
- Possibly alternative abstract representation(s) of a Block:
  - one Objective (but possibly vector-valued)
  - any # of groups of (static) Variable
  - any # of groups of std::list of (dynamic) Variable
  - any # of groups of (static) Constraint
  - any # of groups of std::list of (dynamic) Constraint

  groups of Variable/Constraint can be single (std::list) or std::vector (. . .) or boost::multi_array
- Any # of sub-Blocks (recursively), possibly of specific type (e.g., Block::MMCFBlock has k Block::MCFBlock inside)

Variable

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

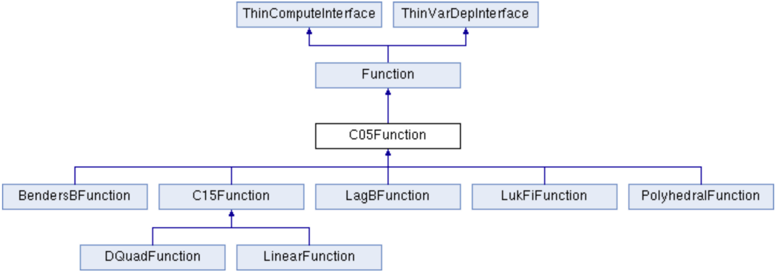
Constraint

- Abstract concept, thought to be extended (any algebraic constraint, a matrix constraint, a PDE constraint, bilevel program, . . .)
- Depends from a set of `Variable` (`:ThinVarDepInterface`)
- Either satisfied or not by the current value of the `Variable`, checking it possibly costly (`:ThinComputeInterface`)
- Knows which `Block` it belongs to
- Can be relaxed and enforced
- Fundamental design decision: "name" of a `Constraint` = its memory address $\Longrightarrow$ copying a `Constraint` makes a different `Constraint` $\Longrightarrow$ dynamic `Constraint`s always live in `std::lists`
- `ConstraintModification:Modification` (relax/enforce)

Objective

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

Function



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

C05Function and C15Function

- 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 $\Longrightarrow$ reoptimization (linearizations shift, some linearizations entries changing in simple ways)
- C15Function supports (partial) Hessians
- Arbitrary hierarchy of :Function possible/envisioned, any one that makes sense for application and/or solution method

Closer to the ground

- `ColVariable:Variable`: "value $=$ one single real" (possibly $\in \mathbb{Z}$)
- `RowConstraint:Constraint`: "$l \leq$ a real $\leq u$" $\Longrightarrow$ has dual variable (single real) attached to it
- `OneVarConstraint:RowConstraint`: "a real" $=$ a single `ColVariable` $\equiv$ bound constraints
- `FRowConstraint:RowConstraint`: "a real" given by a `Function`
- `FRealObjective:RealObjective`: "value" given by a `Function`
- `LinearFunction:Function`: a linear form in `ColVariable`
- `DQuadFunction:Function`: a separable quadratic form
- Many things missing (`AlgebraicFunction`, `DenseLinearFunction`, `Matrix/VectorVariable`, ...)

Block and 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
- However, Variable are always present to interface with Solver (this may change thanks to methods factory)
- A general-purpose Solver uses the abstract representation
- Dynamic Variable/Constraint can be generated on demand (user cuts/lazy constraints/column generation)
- For a Solver attached to a Block:
  - Variable not belonging to the Block are constants
  - Constraint not belonging to the Block are ignored

  (belonging = declared there or in any sub-Block recursively)
- Objective of sub-Blocks summed to that of father Block if has same verse, otherwise min/max

Solver

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

Block and Modification

- Most Block components can change, but not all:
  - set of sub-Block
  - # and shape of groups of Variable/Constraint
- Any change is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification object
- anyone_there() $\equiv \exists$ interested Solver (Modification needed)
- However, two different kinds of Modification (what changes):
  - physical Modification, only specialized Solver concerned
  - abstract Modification, only Solver using it concerned
- Abstract Modification used to keep both representations in sync
  $\Longrightarrow$ a single change may trigger more than one Modification
  $\Longrightarrow$ concerns_Block() mechanism to avoid this to repeat
  $\Longrightarrow$ parameter in changing methods to avoid useless Modification
- Specialized Solver disregard abstract Modification and vice-versa
- A Block may refuse to support some changes (explicitly declaring it)

Modification

- Almost empty base class, then everything has its own derived ones
- Heavy stuff can be attached to a `Modification` (e.g., added/deleted dynamic `Variable`/`Constraint`)
- Each `Solver` has the responsibility of cleaning up its list of `Modification` (smart pointers → memory eventually released)
- `Solver` supposedly reoptimize to improve efficiency, which is easier if you can see all list of changes at once (lazy update)
- `GroupModification` to (recursively) pack many `Modification` together ⟹ different "channels" in `Block`
- `Modification` processed in the arrival order to ensure consistency
- A `Solver` may optimize the changes (`Modifications` may cancel each outer out . . .), but its responsibility

Support to (coarse-grained) Parallel Computation

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

Solution

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

Configuration

- `Block` a tree-structured complex object $\Longrightarrow$ `Configuration` for them a (possibly) tree-structured complex object
- But also `SimpleConfiguration<T>:Configuration` (`T` an `int`, a `double`, a `std::pair<>`, ...)
- `[C/O/R]BlockConfiguration:Configuration` set [recursively]:
  - which dynamic `Variable`/`Constraint` are generated, how (`Solver`, time limit, parameters ...)
  - which `Solution` is produced (what is saved)
  - the `ComputeConfiguration:Configuration` of any `Constraint`/`Objective` that needs one
  - a bunch of other `Block` parameters
- `[R]BlockSolverConfiguration:Configuration` set [recursively] which `Solver` are attached to the `Block` and their `ComputeConfiguration:Configuration`
- Can be `clear()`-ed for cleanup

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`/`Constraint`), useful for algorithmic purposes (branch, fix, solve, ...)
- Solution of R^3Block useful to `Solver` for original `Block`: `map_back_solution()` (best effort in case of dynamic `Variable`)
- Sometimes keeping R^3Block in sync with original necessary: `map_forward_Modification()`, task of original `Block`
- `map_forward_solution()` and `map_back_Modification()` useful, e.g., dynamic generation of `Variable`/`Constraint` in the R^3Block
- `:Block` is in charge of all this, thus decides what it supports

A lot of other support stuff

- All tree-structured complex objects (`Block`, `Configuration`, ...) and `Solver` have an (almost) automatic factory
- All tree-structured complex objects (...) have methods to serialize/deserialize themselves to `netCDF` files
- A methods factory for changing the physical representation without knowing of which `:Block` it exactly is (standardised interface)
- `AbstractBlock` for constructing a model a-la algebraic language, can be derived for "general `Block` + specific part"
- `PolyhedralFunction[Block]`, very useful for decomposition
- `AbstractPath` for indexing any `Constranit`/`Variable` in a `Block`
- `FakeSolver:Solver` stashes away all `Modification`, `UpdateSolver:Solver` immediately forwards/R^{3}Bs them
- ...

Main Existing `:Block`

- `MCFBlock`/`MMCFBlock`: single/multicommodity flow (p.o.c.)
- `UCBlock` for UC, abstract `UnitBlock` with several concrete (`ThermalUnitBlock`, `HydroUnitBlock`, ...), abstract `NetworkBlock` with a few concrete (`DCNetworkBlock`)
- `LagBFunction:{C05Function,Block}` transforms any `Block` (with appropriate `Objective`) into its dual function
- `BendersBFunction:{C05Function,Block}` transforms any `Block` (with appropriate `Constraint`) into its value function
- `StochasticBlock` implements realizations of scenarios into any `Block` (using methods factory)
- `SDDPBlock` represents multi-stage stochastic programs suitable for Stochastic Dual Dynamic Programming

Main “Basic” :Solver

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

[74] https://github.com/frangio68/Min-Cost-Flow-Class

[75] https://gitlab.com/frangio68/ndosolver_fioracle_project

[76] https://gitlab.com/stochastic-control/StOpt

Our Masterpiece: LagrangianDualSolver

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

The many things that we do not have (yet)

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

The many things that we do not have (yet)

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

[77] F., Perez Sanchez "Transforming Mathematical Models Using Declarative Reformulation Rules" *LNCS*, 2011

Conclusions
(for good, this time)

Conclusions and (a lot of) future work

- Decomposition methods (D-W, Benders') old ideas, well-understood, but by-the-book decomposition often not effective enough
- Many nontrivial ideas to improve on the standard approaches

Conclusions and (a lot of) future work

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

Conclusions and (a lot of) future work

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

Conclusions and (a lot of) future work

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

Conclusions and (a lot of) future work

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

[78] Kiwiel "An alternating linearization bundle method for ... and nonlinear multicommodity flow problems" *Math. Prog.*, 2013

Acknowledgements

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