Fully Incremental Bundle Methods: (Un)cooperative (Un)faithful Oracles and Upper Models

Antonio Frangioni[1] Wim van Ackooij[2]

1. Dipartimento di Informatica, Università di Pisa
2. EDF R&D OSIRIS

23^{rd} International Symposium on Mathematical Programming

ISMP 2018 – Bordeaux, July 3, 2018

Outline

Motivation: Lagrangian relaxation

- Hard block-structured problem

$$\sup \left\{ \sum_{k\in\mathcal{K}} c^k u^k \;:\; \sum_{k\in\mathcal{K}} A^k u^k = b \;,\;\; u^k \in U^k \quad k \in \mathcal{K} \right\} \qquad (1)$$

- Lagrangian dual w.r.t. linking constraints

$$\min \left\{ f(x) = xb + \sum_{k\in\mathcal{K}} f^k(x) = \sup \left\{ (c^k - xA^k) u^k : u^k \in U^k \right\} \right\} \qquad (2)$$

- $\nu(2) \geq \nu(1)$, bound tight, useful for heuristic and exact approaches
- Countless many applications, e.g. Uncertain Unit Commitment[1,2]
- Many small subproblems rather than a large one, but:
  - to be solved many times (iterative approach to (2))
  - possibly each one still rather hard
  - possibly rather different from each other (thermal vs. hydro units ...)
  - did I say they can be many already?

[1] van Ackooij, Danti Lopez, F., Lacalandra, Tahanan "Large-scale Unit Commitment Under Uncertainty [...]" *AOR*, 2018

[2] Scuzziato, Finardi, F. "Comparing Spatial and Scenario Decomposition for Stochastic [...]" *IEEE Trans. Sust. En.*, 2018

Solving the Lagrangian Dual

- Sequence $\{ x_i \}$ of iterates $\Longrightarrow$ solutions $u_i = [\, u_i^k \,]_{k \in \mathcal{K}}$
  $\Longrightarrow f^k(x_i) = (c^k - x_i A^k) u_i^k$, $-x_i A^k = z_i^k \in \partial f^k(x_i)$
- Bundles $\mathcal{B}^k = \{ \, (\, z_i^k \, , \, \alpha_i^k = z_i^k x_i - f_i^k \,) \, \}$, Cutting Plane models
  $\check{f}_{\mathcal{B}}^k(x) = \max \{ \, z_i^k x - \alpha_i^k \; : \; (z_i^k, \alpha_i^k) \in \mathcal{B}^k \, \} \leq f^k(x)$
- Master Problem $x_+ \in \operatorname{argmin} \{ \check{f}_{\mathcal{B}}(x) = xb + \sum_{k \in \mathcal{K}} \check{f}_{\mathcal{B}}^k(x) \, \}$ (a LP)
  $\Longrightarrow$ Cutting-Plane Method
- Several issues (MP unbounded below), especially instability:
  $\{ \, x_i \, \}$ "swings wildly" even if x_i close to the optimum
- Gedankenexperiment: start from x_*, constrain $\| x - x_* \|_\infty \leq \delta$

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

- Would work wonders . . .

Solving the Lagrangian Dual

- Sequence $\{ x_i \}$ of iterates $\Longrightarrow$ solutions $u_i = [\, u_i^k \,]_{k \in \mathcal{K}}$
  $\Longrightarrow f^k(x_i) = (c^k - x_i A^k) u_i^k$, $-x_i A^k = z_i^k \in \partial f^k(x_i)$
- Bundles $\mathcal{B}^k = \{ \, (\, z_i^k \, , \, \alpha_i^k = z_i^k x_i - f_i^k \,) \, \}$, Cutting Plane models
  $\check{f}_{\mathcal{B}}^k(x) = \max \{ \, z_i^k x - \alpha_i^k \; : \; (z_i^k, \alpha_i^k) \in \mathcal{B}^k \, \} \leq f^k(x)$
- Master Problem $x_+ \in \operatorname{argmin}\{ \check{f}_{\mathcal{B}}(x) = xb + \sum_{k \in \mathcal{K}} \check{f}_{\mathcal{B}}^k(x) \, \}$ (a LP)
  $\Longrightarrow$ Cutting-Plane Method
- Several issues (MP unbounded below), especially instability:
  $\{ \, x_i \, \}$ "swings wildly" even if x_i close to the optimum
- Gedankenexperiment: start from x_*, constrain $\| x - x_* \|_\infty \leq \delta$

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

- Would work wonders . . . if only we knew x_*

Stabilizing the CPM

- Stability center $\bar{x}$, stabilization parameter $t > 0$
- Stabilized MP (proximal version) $x_+ = \text{argmin}\{\ \check{f}_{\mathcal{B}}(x) + \frac{1}{2t}\|x - \bar{x}\|^2\ \}$
- Translated function $f^k_{\bar{x}}(d) = f^k(\bar{x} + d) - f^k(\bar{x}) \Longrightarrow$
  translated model $\check{f}^k_{\mathcal{B},\bar{x}}(d) = \check{f}^k_{\mathcal{B}}(\bar{x} + d) - f^k(\bar{x}) \Longrightarrow$
  linearization errors $\alpha^k_i(\bar{x}) = f^k(\bar{x}) - [\ f^k(x_i) + z^k_i(\bar{x} - x_i)\ ] \geq 0 \Longrightarrow$
  $\check{f}^k_{\mathcal{B},\bar{x}}(d) = \max\{\ z^k_i d - \alpha^k_i(\bar{x})\ :\ i \in \mathcal{B}^k\ \} \leq f^k_{\bar{x}}(d) \Longrightarrow$
  $z^k_i \in \partial_{\alpha^k_i} f^k(\bar{x})$ (for simplicity, $\alpha^k_i(\bar{x}) \to \alpha^k_i$)
- Primal and dual MP (Θ = unitary simplex):

$$\min\{\ \textstyle\sum_{k\in\mathcal{K}} v^k + \frac{1}{2t}\|d\|^2\ :\ v^k \geq z^k_i d - \alpha^k_i \quad i \in \mathcal{B}^k\ ,\ \ k \in \mathcal{K}\} \quad (3)$$

$$\min\left\{\ \frac{1}{2}t\left\|\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{B}^k} z^k_i\theta^k_i\right\|^2 + \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{B}^k}\alpha^k_i\theta^k_i\ :\ \theta^k \in \Theta^k \quad k \in \mathcal{K}\right\} \quad (4)$$

(Standard) Proximal Bundle Method

- $\nu(3) = -\nu(4)$, primal-dual relationships

$$z_* = \sum_{k \in \mathcal{K}} \left(z_*^k = \sum_{i \in \mathcal{B}^k} z_i^k \theta_{i*}^k \right) \ , \ \alpha_* = \sum_{k \in \mathcal{K}} \left(\alpha_*^k = \sum_{i \in \mathcal{B}^k} \alpha_i^k \theta_{i*}^k \right) \geq 0$$

$$d_* = -tz_* \ \ , \ \ v_* = -t \|z_*\|^2 - \alpha_* = \sum_{k \in \mathcal{K}} \left(v_*^k = d_* z_*^k - \alpha_*^k \right) \leq 0$$

- $x_+ = \bar{x} + d_* = \bar{x} - tz_*$ with $z_* \in \partial_{\alpha_*} f(\bar{x})$ (ε-subgradient method)
- Serious Step condition: $f(x_+) \leq f(\bar{x}) + mv_*$, $m \in (0,1)$ (Armijo-type) $\Longrightarrow \bar{x} \leftarrow x_+$ (SS), otherwise $\bar{x}$ unchanged (Null Step)
- With just fixed t, $\{ \bar{x}_i \} \to x_*$[3], then dynamic t-strategies[4]
- Disaggregate MP (3)/(4) $\Longrightarrow$ good convergence[5] (usually)
- However, solve all subproblems exactly at every iteration
- Sometimes too costly, need to do better

---

[3] Correa, Lemaréchal "Convergence of Some Algorithms for Convex Minimization" *Math. Prog.*, 1993

[4] Lemaréchal, Sagastizábal "Variable metric bundle methods: from conceptual to implementable forms" *Math. Prog.*, 1997

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

Outline

The Incremental Idea

- Would be nice to only compute a subset $\mathcal{Z} \subset \mathcal{K}$ of components
- This is clearly possible (and easy) at NS
- Effect of a NS: new information enters $\mathcal{B} \Longrightarrow \|z_*\|^2 \searrow 0$ and $\alpha_* \searrow 0$
- $\sim$ SS condition can be declared knowing only f^k for $k \in \mathcal{Z}$:

$$\Delta f^{\mathcal{Z}} = \sum_{k \in \mathcal{Z}} \left(\Delta f^k = f^k(x_+) - \check{f}^k(x_+) \right) \geq m(-v_*) \geq m\nu(4) \quad (5)$$

(use $\check{f}^k(x_+) \leq f^k(x_+)$ for $k \notin \mathcal{Z}$)

- Technical lemma: (5) $\Longrightarrow$

$$\nu(4) - \nu(4_+) \geq \frac{\Delta f^{\mathcal{Z}}}{2} \min \left\{ 1 \, , \, \frac{\Delta f^{\mathcal{Z}}}{t_+ \|z_*^{\mathcal{Z}} - z^{\mathcal{Z}}\|^2} \right\} \geq 0$$

- Assuming $\|z^k\|$ bounded and t_i bounded above (easy)

$v_* \geq \varepsilon > 0 \Longrightarrow \nu(4_i) \searrow -\infty$ ↯ $\Longrightarrow \|z_*\|^2 \searrow 0$ and $\alpha_* \searrow 0$

Incremental vs. Inexact

- Can be seen as special case of inexact Bundle method
- $f(x_i)$ approximately computed with unknown (bounded) error ε: just use $\check{f}^k(x_i)/z_*^k$ in place of $f^k(x_i)/z_i^k$ (lower oracle)
- A whole convergence theory exists, even for non-lower oracles[6]
- Issue: by under-estimating $f(x_i)$, you can do a "bad SS"
- Technically: $\alpha_i^k \geq 0$ no longer true $\Longrightarrow$ $v_* > 0$ can happen $\Longrightarrow$ SS condition no longer characterizes a descent step ↯
- Solution I: Noise Reduction $\equiv$ change t (increase it) requires proper handling of NR steps
- Solution II: exact oracle at SS $\equiv \varepsilon = 0 \equiv \mathcal{Z} = \mathcal{K}$ requires lots of work at every SS (in theory, only the last one)
- Can we do better?

[6] de Oliveira, Sagastizábal, Lemaréchal "Convex proximal bundle methods in depth: [. . .] inexact oracles" *Math. Prog.*, 2014

Outline

Ingredient I: A More Detailed Oracle

- Informative cooperative oracle: inputs point x, lower and upper targets $-\infty \leq \underline{\mathtt{tar}}^k \leq \overline{\mathtt{tar}}^k \leq \infty$, accuracy $0 \leq \varepsilon^k \leq \infty$, outputs:

$$\left\{ \begin{array}{l} \left[\begin{array}{l} \text{i) function value information: two values } \underline{f}^k \text{ and } \bar{f}^k \text{ s.t.} \\ \qquad -\infty \leq \underline{f}^k \leq f^k(x) \leq \bar{f}^k \leq \infty \quad \text{and} \quad \bar{f}^k - \underline{f}^k \leq \varepsilon^k \\ \text{ii) first-order information: if } \underline{f}^k > -\infty \text{, a } z^k \in \mathbb{R}^n \text{ s.t.} \\ \qquad f^k(\cdot) \geq \underline{f}^k + z^k(\cdot - x) \end{array} \right. \\ \text{iii) s.t. at least one between } \bar{f}^k \leq \overline{\mathtt{tar}}^k \text{ and } \underline{f}^k \geq \underline{\mathtt{tar}}^k \text{ holds} \end{array} \right.$$

- Typical application: exact approach for hard (Lagrangian) problem
  - heuristic $\longrightarrow$ "good" $\bar{u}^k \in U^k \Longrightarrow \underline{f}^k = c^k \bar{u}^k \leq f^k(x)$, $z^k = -A^k \bar{u}^k$
  - relaxation $\longrightarrow$ "good" upper bound $\bar{f}^k \geq f^k(x)$
  - any amount of branching and/or cutting to make $\underline{f}^k$ and $\bar{f}^k$ "close"
- Explicit upper bound (almost[7]) never considered before
- Parameters allow to stop early; e.g., if $\bar{f}^k \leq \overline{\mathtt{tar}}^k$ heuristic not ran at all, no z^k even produced (vice-versa if $\underline{f}^k \geq \underline{\mathtt{tar}}^k$)

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

Ingredient II: Upper Model

- Upper bundle $\mathcal{P}^k = \{ (x_i , \bar{f}_i^k) \} \Longrightarrow$ trivial upper model:

  $$\dot{f}_{\mathcal{P}}^k(x) = \inf \{ \textstyle\sum_{i \in \mathcal{P}^k} \bar{f}_i^k \theta_i^k \; : \; \sum_{i \in \mathcal{P}^k} x_i \theta_i^k = x \, , \; \theta^k \in \Theta^k \} \geq f(x)$$

- Obvious issue: $\dot{f}_{\mathcal{P}}^k(x) = \infty$ for $x \notin \bar{X}_{\mathcal{P}}^k = \mathrm{conv}(\{ x_i \; : \; i \in \mathcal{P}^k \})$

- Assumption: f^k globally Lipschitz $\equiv \| z^k \| \leq L^k$, L^k known $\Longrightarrow$

$$\hat{f}_{\mathcal{P}}^k(x) = \inf \{ \textstyle\sum_{i \in \mathcal{P}^k} \bar{f}_i^k \theta_i^k + L^k \|s^k\|_2 \; : \; \sum_{i \in \mathcal{P}^k} x_i \theta_i^k + s^k = x \, , \; \theta^k \in \Theta^k \}$$

$$= \inf\{ \dot{f}_{\mathcal{P}}^k(w) + L^k \|x - w\|_2 \} < \infty$$

- $\hat{f}_{\mathcal{P}}^k(x) \geq f(x)$, requires solving a SOCP to be computed

- $\mathcal{P}^k$ can be handled independently from $\mathcal{B}^k$: poorman's upper bundles

  $$\mathcal{P}_*^k = \Big\{ (x_*^k , \bar{f}_*^k) = (\textstyle\sum_{i \in \mathcal{P}^k} x_i \theta_{i*}^k + s_*^k , \sum_{i \in \mathcal{P}^k} \bar{f}_i^k \theta_{i*}^k + L^k \|s_*^k\|_2) \Big\}$$

  with s_*^k, θ_*^k corresponding to $\bar{x}_i$ ($\Longrightarrow$ cheap)

- $\{ \hat{f}_{\mathcal{P}}^k(\bar{x}_i) \}$ non increasing, finite even if $\mathcal{Z}_i \subset \mathcal{K}_i$

Ingredient III: Worst-Case Linearization Errors

- Linearization errors defined using the upper model:

$$\alpha_i^k(\bar{x}, \mathcal{P}) = \hat{f}_{\mathcal{P}}^k(\bar{x}) - [\, \underline{f}_i^k + z_i^k(\bar{x} - x_i) \,] \qquad (6)$$

(still α_i^k for simplicity, still $z_i^k \in \partial_{\alpha_i^k} f^k(\bar{x})$)

- Easily updated as $\bar{x}$ changes (information transport property)

$$\alpha_i^k(\tilde{x}, \mathcal{P}) = z_i^k(\, \bar{x} - \tilde{x} \,) + \alpha_i^k(\bar{x}, \mathcal{P}) + (\, \hat{f}_{\mathcal{P}}^k(\tilde{x}) - \hat{f}_{\mathcal{P}}^k(\bar{x}) \,) \;\; (\, \geq 0 \,) \qquad (7)$$

- Take into account the gap between upper and lower bound:

$$\hat{f}_{\mathcal{P}}^k(\bar{x}) - \check{f}_{\mathcal{B}}^k(\bar{x}) = \min \{\, \alpha_j^k \;:\; j \in \mathcal{B}^k \,\} \leq \alpha_i^k \quad \forall i \in \mathcal{B}^k \qquad (8)$$

- Reliable upper approximation of the true α_i^k, even if $\mathcal{Z}_i \subset \mathcal{K}_i$
- Different from inexact approach, which uses $\hat{f}_{\mathcal{P}}^k(\bar{x})$ instead of $\hat{f}_{\mathcal{P}}^k(\tilde{x})$ $\Longrightarrow$ no more "unreliable" SS $\Longrightarrow$ no more NR

The Main Loop

0 $\forall\, k \in \mathcal{K}$, $(\underline{f}_1^k, \bar{f}_1^k, z_1^k) \leftarrow \mathcal{O}^k(-\infty, \infty, \varepsilon^k, \bar{x}_1)$; $\mathcal{P}_1^k \leftarrow \{\, (\bar{x}_1, \bar{f}_1^k)\,\}$,
$\mathcal{B}_1^k \leftarrow \{\, (z_1^k, \alpha^k(\bar{x}_1, \mathcal{P}_1^k))\,\}$; $\ell \leftarrow 1$

1 solve (3)/(4) for $d_{*,\ell}$, $v_{*,\ell}^k$, $\theta_{*,\ell}^k$, $z_{*,\ell}^k$ and $\alpha_{*,\ell}^k$;

2 `if` $\|z_{*,\ell}\| \leq \delta_1$ `&&` $\alpha_{*,\ell} \leq \delta_2$ `then stop`

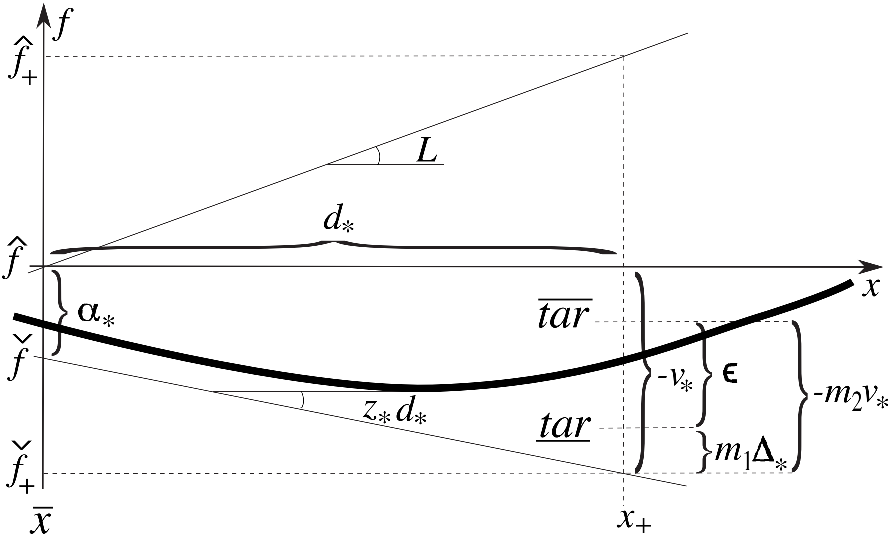
3 $\Delta_{*,\ell} \leftarrow t_\ell \left\| z_{*,\ell} \right\|^2 /2 + \alpha_{*,\ell}$; $x_{\ell+1} \leftarrow \bar{x}_\ell + d_{*,\ell}$; $\check{f}_\ell(x_{\ell+1}) = \hat{f}_\ell(\bar{x}_\ell) + v_{*,\ell}$;
$\overline{\mathtt{tar}}_\ell \leftarrow \check{f}_\ell(x_{\ell+1}) - m_2 v_{*,\ell}$; $\underline{\mathtt{tar}}_\ell \leftarrow \check{f}_\ell(x_{\ell+1}) + m_1 \Delta_{*,\ell}$;
$\varepsilon_\ell \leftarrow \overline{\mathtt{tar}}_\ell - \underline{\mathtt{tar}}_\ell$;
$(\mathcal{B}_{\ell+1}, \mathcal{P}_{\ell+1}) \leftarrow$ `Inner_Loop`$(\, \mathcal{B}_\ell,\, \mathcal{P}_\ell,\, \bar{x}_\ell,\, x_{\ell+1},\, \varepsilon_\ell,\, v_{*,\ell}^k,\, \Delta_{*,\ell},\, m_1,\, m_2)$;

4 `if` $\hat{f}_{\ell+1}(x_{\ell+1}) \leq \overline{\mathtt{tar}}_\ell$ `then` perform a SS; `if` $\check{f}_{\ell+1}(x_{\ell+1}) \geq \underline{\mathtt{tar}}_\ell$ `then` perform a NS; (if both hold, choose arbitrarily)

5 Appropriately update $\mathcal{P}_{\ell+1}$, $\mathcal{B}_{\ell+1}$, $t_{\ell+1}$; $\ell \leftarrow \ell + 1$; `go to 1`;

- Quite a few algorithmic parameters: $\delta_1 \geq 0$, $\delta_2 \geq 0$, $0 < m_1 < m_2 < 1$, $\bar{x}_1$, $t_1 > 0$, $0 \leq \varepsilon^k < \infty$

A Picture is Worth 1000 Words



- Example of SS and NS both possible

Nontrivial Details

- Convergence quite easy with standard ideas (t-management, $\mathcal{B}$-management, ...) and results[3] except for a few subtle points
- Adding $(x_{\ell+1}, \bar{f}^k)$ to $\mathcal{P}^k_\ell$ may decrease $\hat{f}_\ell(\bar{x}_\ell) \Longrightarrow \alpha^k_i \Longrightarrow \alpha^k_* \Longrightarrow -v_*$
- The SS condition may not hold any more with the recomputed v_* which is why we don't recompute it ($\overline{\texttt{tar}}$, $\underline{\texttt{tar}}$ fixed in Inner Loop)
- "Almost fake" SS
- Anyway, SS $\Longrightarrow \hat{f}_{\ell+1}(\bar{x}_{\ell+1}) \ll \hat{f}_\ell(\bar{x}_\ell)$, even if $\hat{f}_\ell(\bar{x}_\ell) \ll \hat{f}_{\ell+1}(\bar{x}_\ell)$
- Similarly, "almost fake" NS, but $\alpha^k_i \Longrightarrow \alpha^k_* \Longrightarrow \nu(4)$ decreases
- All in all, convergence holds if the Inner Loop works
- Whenever the oracle is called, a sanity check is done:

$$\bar{f}^k_+ := \min\{\, \bar{f}^k_+ \,,\, \hat{f}^k_{\mathcal{P}}(x_+) \,\} \quad , \quad \underline{f}^k_+ := \max\{\, \underline{f}^k_+ \,,\, \check{f}^k_{\mathcal{B}}(x_+) \,\} \qquad (9)$$

The Inner Loop

0 Input $\mathcal{B}$, $\mathcal{P}$, $\bar{x}$, x_+, $\underline{\mathtt{tar}}$, $\overline{\mathtt{tar}}$, ε, v_*^k, Δ_*, m_1, m_2; $\mathcal{Z} \leftarrow \emptyset$;
for each $k \in \mathcal{K}$ do $\mathcal{P}^k \leftarrow \mathcal{P}^k \cup \{\, (\, x_+ \,,\, \hat{f}_+^k = \hat{f}_{\mathcal{P}}^k(x_+)\,)\,\}$;
Arbitrarily set $\beta^k \geq 0$ s.t. $\sum_{k \in \mathcal{K}} \beta^k = 1$; $\check{f}_+^k = \hat{f}^k + v_*^k$;
$\underline{\mathtt{tar}}^k \leftarrow \check{f}_+^k + m_1 \beta^k \Delta_*$; $\overline{\mathtt{tar}}^k \leftarrow \check{f}_+^k - m_2 \beta^k v_*$;

1 Arbitrarily select $k \in \mathcal{K}$ and $\varepsilon^k \geq \beta^k \varepsilon$; $(\underline{f}_+^k, \bar{f}_+^k, z^k) \leftarrow$
$\mathcal{O}^k(\, \min\{\, \underline{\mathtt{tar}}^k \,,\, \underline{\mathtt{tar}} - \underline{f}_+^{-k} \,\}, \max\{\, \overline{\mathtt{tar}}^k \,,\, \overline{\mathtt{tar}} - \bar{f}_+^{-k} \,\} \,,\, \varepsilon^k \,,\, x_+ \,)$;
update $\underline{f}_+^k$ and $\bar{f}_+^k$ according to (9);

2 $\mathcal{P}^k \leftarrow \mathcal{P}^k \cup \{\, (\, x_+ \,,\, \hat{f}_+^k = \hat{f}_{\mathcal{P}}^k(x_+)\,)\,\}$ replacing the previous pair;
if z^k has been produced then
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{k\}$; $\mathcal{B}^k \leftarrow \mathcal{B}^k \cup \{\, (z^k \,,\, \alpha_+^k(\bar{x}, \mathcal{P})\,)\,\}$;

3 if neither $\hat{f}_{\ell+1}(x_{\ell+1}) \leq \overline{\mathtt{tar}}_\ell$ nor $\check{f}_{\ell+1}(x_{\ell+1}) \geq \underline{\mathtt{tar}}_\ell$ then go to 1;

- "$-k$" $= \mathcal{K} \setminus \{\, k \,\}$; at first call, $\varepsilon^k = \infty$
- Assumption: eventually, $\varepsilon^k = \beta^k \varepsilon \implies$ terminates
- Deciding β^k nontrivial, but interesting (cost, AI/ML prediction, ...)

Aside: Partial Aggregation

- Approach would seem to necessarily require disaggregation
- Usually disaggregation is good (fast convergence), but
  if $|\mathcal{K}|$ large, MP can be very costly
- Partial aggregation may be in principle useful
- Static partial aggregation easy to do, but inflexible
- Alluring idea: one single cut for each iteration, even if $\mathcal{Z}_\ell \neq \mathcal{Z}_{\ell+1}$
- Actually possible: partly aggregated cuts $\sum_{k \in \mathcal{Z}} v^k \geq z_i^{\mathcal{Z}} d - \alpha_i^{\mathcal{Z}}$
- Need to keep the disaggregated representation z_i^k, α_i^k
- Need to keep the disaggregated upper bundles $\mathcal{P}^k$
- For the rest it works without problems, possibly well[8]

[8] Helmberg, Pichler "Dynamic scaling and submodel selection in bundle methods for convex optimization" *OO* 6180, 2017

Accuracy

- The algorithm converges to an (approximatively) δ_2-optimal solution (exactly if $\delta_1 = 0$, otherwise somewhat hard to establish)
- Each oracle never asked more than $\delta_2 \beta^k$ absolute accuracy: the relative size of $f^k(x_*)$ matters, as well as the choice of β^k
- If δ_2 small, high accuracy is required (albeit only towards the end of the algorithm)
- This may be impossible or too costly
- Thus far, "gentlemen agreement" between algorithm and oracle: algorithm only asks as little as possible, but oracle must cooperate
- What if the oracle cannot/does not want to cooperate?

Outline

Uncooperative Oracles

- Any reasonable uncooperative oracle boundedly so:
  finite maximum error $\bar{\varepsilon} < \infty \Longrightarrow$ at best $\bar{\varepsilon}$-optimal solution[9]
- There are actually three different forms of uncooperative oracles
- Each form corresponds to a different NR step
- Only one of them was known before
- All forms give similar results (approximately $\bar{\varepsilon}$-optimal solution)
  but with differences (a-posteriori optimality estimate)
- Cheating oracles particularly tricky ($2\bar{\varepsilon}$-optimal if not uniform)
- Different types of uncooperative oracles can be mixed

[9] d'Antonio, F. "Convergence analysis of deflected conditional approximate subgradient methods" *SIOPT*, 2009

Informative boundedly uncooperative oracles

- Declare a-priori the smallest achievable accuracy $0 < \bar{\varepsilon}^k < \infty$
  $\equiv$ only works if $\varepsilon^k = \overline{\mathtt{tar}}^k - \underline{\mathtt{tar}}^k \geq \bar{\varepsilon}^k$
- Approximation algorithm with worst-case a-priori guarantee,
  B&C with no limit on resources, ...
- A-priori NR:

2.1 if $\varepsilon_\ell < \bar{\varepsilon}$ then if $\|z_{*,\ell}\|^2 \leq \delta_1$ then stop else $t_{\ell+1} \leftarrow\gg t_\ell$; $\bar{x}_{\ell+1} \leftarrow \bar{x}_\ell$; $\mathcal{B}_{\ell+1} \leftarrow \mathcal{B}_\ell$; $\mathcal{P}_{\ell+1} \leftarrow \mathcal{P}_\ell$; $\ell \leftarrow \ell + 1$; go to 1;

- Provided $t_\ell \nearrow \infty$ during sequences of NR + NS, converges to (approximately) $[\varepsilon' = (\bar{\varepsilon} = \sum_{k \in \mathcal{K}} \bar{\varepsilon}^k)/(m_2 - m_1)]$-optimal solution ($\approx \bar{\varepsilon}$ as $m_2 \approx 1$ and $m_1 \approx 0$)
- Actually, $(\alpha_{*,\infty} = \liminf_{\ell \to \infty} \alpha_{*,\ell} \leq \varepsilon')$-optimal:
  a-posteriori optimality measure (still approximate if $\delta_1 > 0$)

Uninformative faithful boundedly uncooperative oracles

- Only works if $\varepsilon^k \geq \bar{\varepsilon}^k$, but $\bar{\varepsilon}^k < \infty$ unknown
- Faithful $\equiv$ all answers are correct, possibly just not enough accuracy
- A-posteriori guarantee (PTAS, B&C, ...) but resource limit
- Inner Loop may not satisfy SS or NS condition, "emergency stop"
- A-posteriori NR:

> **4.1** `if` neither SS condition nor NS condition hold `then`
>
> `if` $\|z_{*,\ell}\|^2 \leq \delta_1$ `then stop else` $t_{\ell+1} \leftarrow\gg t_\ell$; $\bar{x}_{\ell+1} \leftarrow \bar{x}_\ell$;

- Provided $t_\ell \nearrow \infty$ during sequences of NR + NS, converges to (approximately) $\max\{\varepsilon', \delta_2\}$-optimal solution
- Again, $\alpha_{*,\infty}$ a-posteriori estimate of solution quality
- More optimistic version: the oracle does not bound itself to obtain accurate solutions but may still attain them $\Longrightarrow$ could get δ_2-optimal solution even if $\delta_2 < \varepsilon'$

Uninformative cheating boundedly uncooperative oracles

- Uncooperative, unknown $\bar{\varepsilon}^k < \infty$ and no $\bar{f}^k \Longrightarrow$
  has to cheat and report "fake" $\bar{f}^k$
- Consequence: $z_i^k \in \partial_{(\alpha_i^k + \bar{\varepsilon}^k)} f^k(\bar{x})$ with unknown $\bar{\varepsilon}^k$
- Pure heuristic $\equiv$ standard assumption in the literature[6] $\equiv$
  uniformly cheating: $\bar{f}^k = \underline{f}^k$ (apparently good for any ε^k)
- Delayed a-posteriori NS:

  1.1 `if` $\alpha_{*,\ell} < -m_3 t_\ell \|z_{*,\ell}\|^2$ (< 0) `then if` $\|z_{*,\ell}\|^2 \leq \delta_1$ `then stop`
  `else` $t_{\ell+1} \leftarrow\gg t_\ell$; $\bar{x}_{\ell+1} \leftarrow \bar{x}_\ell$; $\mathcal{B}_{\ell+1} \leftarrow \mathcal{B}_\ell$; $\mathcal{P}_{\ell+1} \leftarrow \mathcal{P}_\ell$; $\ell \leftarrow \ell + 1$;
  `go to` 1;

  looks "ex ante" like 2.1, but it is "more ex-post" than 4.1
- Require specific arguments[6] because $\alpha_*^k \not\geq 0$ (although $\alpha_*^k + \bar{\varepsilon}^k \geq 0$)
- As usual, $t_\ell \nearrow \infty$ during sequences of NR + NS

If you Cheat, at Least Do So Uniformly

- Uniformly cheating $\Longrightarrow$ no 2.1 and 4.1
- Technical point: slight modification of lower targets

$$\begin{aligned} \underline{\mathtt{tar}}_\ell &:= \check{f}_\ell(x_{\ell+1}) - m_1 v_{*,\ell} \\ \varepsilon_\ell &:= \overline{\mathtt{tar}}_\ell - \underline{\mathtt{tar}}_\ell = (m_2 - m_1)(-v_{*,\ell}) \\ \underline{\mathtt{tar}}^k &:= \check{f}^k_+ - m_1 \beta^k v_* \end{aligned}$$

(a bit worse since $-v_{*,\ell} > \Delta_{*,\ell}$)

- Allows any $m_3 < 1$; usually $m_3 < 1/2^6 \Longrightarrow$ can use original
- Converges to (approximately) $\max\{\bar{\varepsilon}, \delta_2\}$-optimal solution
- Non-uniformly cheating $\Longrightarrow$ $2\bar{\varepsilon}$-optimal (adversary oracle)
- No informative cheating oracle ($\bar{f}^k = \underline{f}^k + \bar{\varepsilon}^k$)
- All three kinds of oracles can be mixed (bit technical, not difficult[10])

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

Outline

1. Motivation and classic results
2. Partly Incremental/Inexact Approaches
3. Upper Models & Fully Incremental Approaches
4. Uncooperative Oracles
5. Computational results
6. The software issue
7. Conclusions and (a Lot of) Future Work

Computational results

Computational results

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

Computational results

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

Computational results

- Er . . . I said it'd be quick . . .
- No, seriously, we still don't have them
- We believe they will be good, because a lot can be done (choosing k, choosing β^k, choosing ε^k, . . .)
- We haven't had the time to do significant tests yet
  $\equiv$ on many significantly different relevant applications
- Part of the issue: developing significant application is "hard"
- Many things have to be recoded each time
- No tools for embedding Lagrangian relaxation into B&C

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 lot can be done (choosing k, choosing β^k, choosing ε^k, . . .)
- We haven't had the time to do significant tests yet
  $\equiv$ on many significantly different relevant applications
- Part of the issue: developing significant application is "hard"
- Many things have to be recoded each time
- No tools for embedding Lagrangian relaxation into B&C
  . . . yet

Outline

Putting all this in practice

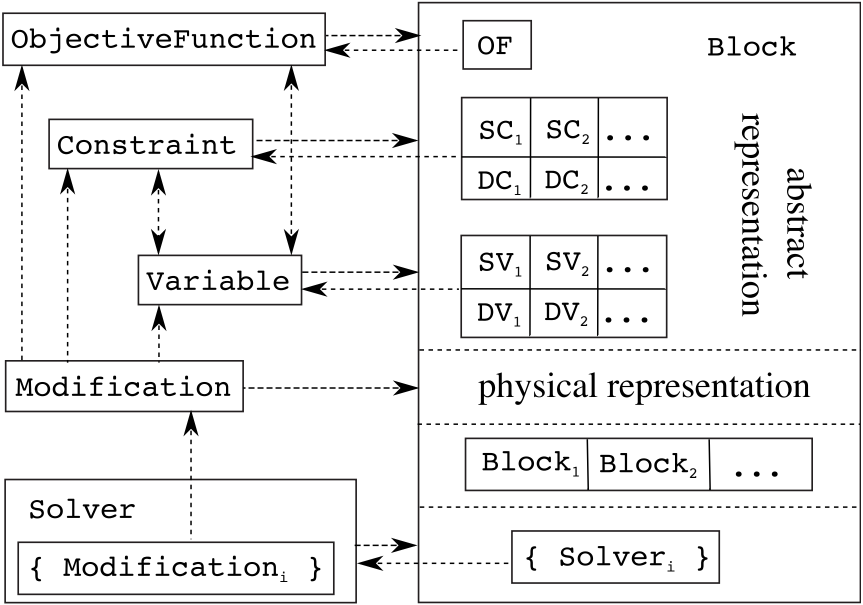
- . . . easier said than done
- Specialized implementations for one application "relatively easy"
- General implementations for all problems with same structure harder: it took $\approx$ 10 years from idea to [5] on top of existing, nicely structured C++ bundle code
- Issue: extracting structure from problems
- Issue: really using this in a B&C approach
- 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
OF
Block
Constraint
SC1 SC2 ...
DC1 DC2 ...
abstract
representation
Variable
SV1 SV2 ...
DV1 DV2 ...
Modification
physical representation
Block1 Block2 ...
Solver
{ Modificationi }
{ Solveri }

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 `Variable`s 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 ...

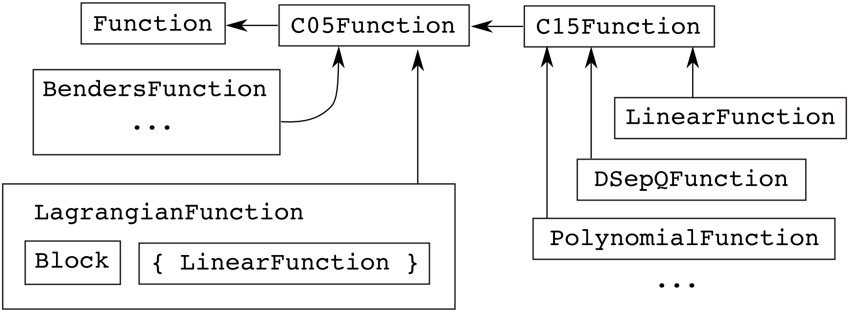
R^3Block

- 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[11]
- Asynchronous evaluation still not defined
- Handles set of Variables upon which it depends
- FunctionModification[Variables] for "easy" changes $\Longrightarrow$ reoptimization (shift, adding/removing "quasi separable" Variables)

[11] 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)

Outline

A Lot of Work, Then Maybe Conclusions

- Fully incremental Proximal Bundle Methods possible
- Should be easy to extend to Level/Doubly Stabilized[12]
- Still a lot to learn computationally (choosing β^k, ...)
- Fully asynchronous Bundle now looks doable (Frank's talk)
- 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?

[12] de Oliveira and M. Solodov "A doubly stabilized bundle method for nonsmooth convex optimization" *Math. Prog.*, 2016

Advertisement

SMS++ WANTS YOU!

- If you can do one or more of:
  - advanced C++ programming;
  - HPC/parallel programming;
  - large-scale optimization, decomposition;
- If you want to earn 30000€ per year (before income taxes)
- If you are willing to move to Pisa for the next two years
- Then please do apply here:

  `https://www.unipi.it/ateneo/bandi/assegni/asse2018/inf/30lug2018/`

  before 30/07/2018 (selection 07/09/2018, starts 01/10/2018)
- Please forward to all possible interested parties and/or contact me

Thanks!

Acknowledgements

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