An introduction to energy optimization in SMS++
Part I: SMS++ basics & energy-related components

Antonio Frangioni
Dipartimento di Informatica, Università di Pisa
EdF Labs — May 25-26, 2023
Please sign up here

https://docs.google.com/spreadsheets/d/1Facxjf6-W80oiIJBvp0LvanWXBmeeOCdVL9aO2WVGtY/edit#gid=0
Meta–Outline

- Part I: SMS++ basics & energy-related components
- Part II: hands-on with SMS++ for energy optimization
- Part III: a quick recap of decomposition techniques
- Part IV: decomposition & energy optimization in SMS++
1 SMS++: design goals

2 Important motivation: energy optimization
   - Drilling down: Thermal Units
   - Scaling up: longer-term problems

3 SMS++: basic components

4 SMS++: existing Block and Solver

5 A glimpse to some results

6 Conclusions (Part I)
https://gitlab.com/smspp/smspp-project

“For algorithm developers, from algorithm developers”

Open source (LGPL3)

Version 0.5.1 (still a long way to go)
What SMS++ is

- A core set of C++-17 classes implementing a *modelling system* that:
  - explicitly supports the notion of Block ≡ nested structure
  - separately provides “semantic” information from “syntactic” details, different formulations (set of constraints/variables) of the same problem
  - allows exploiting specialised Solver on Block with specific structure
  - (potentially) manages any (sensible) 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 0.5.1, lots of further development ahead, significant changes in interfaces not ruled out, actually expected (although current Block / Solver very thoroughly tested)

- **Interfaced with many existing solvers**: Cplex, SCIP, MCFClass, StOpt (although the is growing, and there are built-in ones)
Outline

1. SMS++: design goals

2. **Important motivation: energy optimization**
   - Drilling down: Thermal Units
   - Scaling up: longer-term problems

3. SMS++: basic components

4. SMS++: existing Block and Solver

5. A glimpse to some results

6. Conclusions (Part I)
Schedule a set of generating units over a time horizon $T$ (hours / 15m in day / week) to satisfy the (forecasted) demand $d_t$ at each $t \in T$.

Gazillions €€€ / $$$, enormous amount of research\(^1\)

Different types of production units, different constraints:
- Thermal (comprised nuclear): min / max production, min up / down time, ramp rates on production increase / decrease, start-up cost depending on previous downtime, others (modulation, . . .)
- Hydro (valleys): min / max production, min / max reservoir volume, time delay to get to the downstream reservoir, others (pumping, . . .)
- Non programmable (ROR hydro) intermittent units (solar / wind, . . .)
- Fancy things (small-scale storage, demand response, smart grids, . . .)

Plus the interconnection network (AC / DC, transmission / distribution) and reliability (primary / secondary reserve, $n – 1$ units, . . .)

\(^1\)van Ackooij, Danti Lopez, F., Lacalandra, Tahanan “Large-scale Unit Commitment Under Uncertainty [ . . . ]” AOR 2018
Algorithmic approaches

- Several types of almost independent blocks + linking constraints: perfect for decomposition methods\(^2\), especially in the uncertain case\(^3\).
- Many different structures, today’s one: thermal units (but \(\exists\) many others, e.g. hydro units\(^4\), Energy Communities\(^5\), stochastic\(^3\), . . .)

\(^5\)Fioriti, F., Poli “Optimal Sizing of Energy Communities with Fair Revenue Sharing […]” *Applied Energy* 2021
1. SMS++: design goals

2. Important motivation: energy optimization
   - Drilling down: Thermal Units
   - Scaling up: longer-term problems

3. SMS++: basic components

4. SMS++: existing Block and Solver

5. A glimpse to some results

6. Conclusions (Part I)
Basic aspects of thermal units

- Natural variables $p_t \in \mathbb{R}_+$: power level at time $t \in T$

- Standard constraints:
  - maximum ($\bar{p}_{\text{max}}$) and minimum ($\bar{p}_{\text{min}}$) power output at $t \in T$
    (but start-up / shut-down limits $\bar{l}/\bar{u}$ potentially $\neq$)
  - Ramp-up / down constraints ($\Delta_+ / \Delta_- = \text{ramp-up} / \text{down limit}$)
  - Min up / down-time constraints ($\tau_+ / \tau_- = \text{min up} / \text{down-time}$)

- Power cost: convex quadratic with fixed term
  $$f_t(p_t) = a_t p_t^2 + b_t p_t + c_t \quad \text{(most often } a, b, c \text{ independent from } t)$$

- Possibly rather complex time-dependent start-up costs
  (dependent on intervening down time, maybe also on $t \in T$)
Thermal Units are “easy” via Dynamic Programming

A(n improved) DP algorithm\(^6\) based on the state-space graph \(G\):

- nodes \((t, \uparrow)/(t, \downarrow)\): unit starts up / shuts down at time \(t\)
- arc \beit((h, \uparrow), (k, \downarrow))\beit with \(k - h + 1 \geq \tau^+\): unit on from \(h\) to \(k\) (included)
- arc \beit((h, \downarrow), (k, \uparrow))\beit with \(k - k - 2 \geq \tau^-\): unit off from \(h + 1\) to \(k - 1\)

An \(s-d\) path from represents a schedule for the unit

\[
\begin{align*}
\text{"on" arcs } & \beit((h, \uparrow), (k, \downarrow))\beit: \text{ optimal dispatching cost } z_{hk}^* + \sum_{t=h}^{k} c_t \\
\text{"off" arcs } & \beit((h, \downarrow), (k, \uparrow))\beit: \text{ start-up cost for } k - h - 2 \text{ off time periods}
\end{align*}
\]

---

Optimal dispatch cost $z^*_{hk}$: solving the Economic Dispatch problem $(ED_{hk})$ on $p_h, p_{h+1}, \ldots, p_k$

$$z^*_{hk} = \min \sum_{t=h}^{k} f^t(p_t)$$

$$p_{\min} \leq p_h \leq \bar{l}$$

$$p_{\min} \leq p_t \leq p_{\max} \quad h + 1 \leq t \leq k - 1$$

$$p_{\min} \leq p_k \leq \bar{u}$$

$$p_{t+1} - p_t \leq \Delta_+ \quad t = h, \ldots, k - 1$$

$$p_t - p_{t+1} \leq \Delta_- \quad t = h, \ldots, k - 1$$

Complexity:

- acyclic graph $O(n)$ nodes, $O(n^2)$ arcs $\Rightarrow O(n^2)$ for optimal path
- $O(n^3)$ for computing all costs via specialized inner DP for $(ED_{hk})$

$\Rightarrow O(n^3)$ overall

Basic building block for efficient Lagrangian approaches

---

7 F., Gentile, Lacalandra “Solving Unit Commitment Problems with General Ramp Contraints” IJEPES, 2008
Basic MIP formulation

- Natural variables $u_t \in \{0, 1\}$: on / off state at $t \in T$

- Standard formulation with time-dependent start-up costs

$$\min s(u) + \sum_{t \in T} (a_t p_t)^2 + b_t p_t + c_t u_t \quad (7)$$

$$\bar{p}_{\min} u_t \leq p_t \leq \bar{p}_{\max} u_t \quad t \in T \quad (8)$$

$$p_t \leq p_{t-1} + u_{t-1} \Delta_+ + (1 - u_{t-1}) \bar{l} \quad t \in T \quad (9)$$

$$p_{t-1} \leq p_t + u_t \Delta_- + (1 - u_t) \bar{u} \quad t \in T \quad (10)$$

$$u_t \leq 1 - u_{r-1} + u_r \quad t \in T, r \in [t - \tau_+, t - 1] \quad (11)$$

$$u_t \geq 1 - u_{r-1} - u_r \quad t \in T, r \in [t - \tau_-, t - 1] \quad (12)$$

requires extra constraints + continuous variables\(^8\) for $s(u)$

- Rather weak formulation $\implies$ weak lower bounds $\implies$ rather nasty MIQP, unsolvable as-is for $> 20$ units (real-world versions worse)

- Especially since UC needs be solved “unreasonably fast”

---

\(^8\) Nowak, Römisch “Stochastic Lagrangian Relaxation Applied to Power Scheduling [...]” Annals O.R. 2000
Improved MIP formulations (1)

- Convex hull of the min-up / down constraints (11) / (12) known\(^9\): exponential number of constraints, but separable in poly time

- Indeed, extended formulation\(^{10}\): start-up / shut-down \(v_t / w_t\) variables

\[
u_t - u_{t-1} = v_t - w_t \quad t \in T
\]  

- Can be extended to start-up / shut-down limits\(^{11}\) \((\tau_+ \geq 2 \neq \tau_+ = 1)\)

\[
\begin{align*}
p_1 & \leq \bar{p}_{\text{max}} u_t - (\bar{p}_{\text{max}} - \bar{u}) w_{t+1} \\
p_t & \leq \bar{p}_{\text{max}} u_t - (\bar{p}_{\text{max}} - \bar{l}) v_t - (\bar{p}_{\text{max}} - \bar{u}) w_{t+1} \quad t \in [2, |T| - 1] \\
p_T & \leq \bar{p}_{\text{max}} u_t - (\bar{p}_{\text{max}} - \bar{l}) v_t
\end{align*}
\]

---


\(^{10}\) Rajan, Takriti, “Minimum Up/Down polytopes of the unit commitment problem with start-up costs”, IBM RC23628, 2005

Improved MIP formulations (2)

- Convex quadratic objective function with semi-continuous variables:
  Perspective Reformulations\textsuperscript{12,13} \( \sum_{t \in T} \left( a_t \left( p_t \right)^2 / u_t + b_t p_t + c_t u_t \right) \)

- Several ways to deal with the “more nonlinearity”\textsuperscript{14,15}

- Start-up cost is a concave in previous shut-down period length \( \tau \):
  \( cs(\tau) = V(1 - e^{-\lambda \tau}) + F \) (only required for integer \( \tau \))

- Convex hull description of the start-up cost fragment: extended formulation with temperature variables\textsuperscript{16}

\textsuperscript{12} F., Gentile “Perspective cuts for a class of convex 0-1 mixed integer programs” \textit{Math. Prog.} 2006
\textsuperscript{13} F., Gentile, Lacalandra “Tighter approximated MILP formulations for Unit Commitment Problems” \textit{IEEE TPWRS} 2009
\textsuperscript{14} F., Gentile “A Computational Comparison of [. . .]: SOCP vs. Cutting Planes” \textit{ORL} 2009
\textsuperscript{15} F., Furini, Gentile “Approximated Perspective Relaxations: a Project&Lift Approach” \textit{COAP} 2016
\textsuperscript{16} Silbernagl, Huber, Brandenberg “[. . .] MIP Unit Commitment by Modeling [. . .] Temperatures” \textit{IEEE TPWRS} 2016
Improved MIP formulations (2)

- **Convex quadratic** objective function with semi-continuous variables:
  \[
  \sum_{t \in T} \left( \frac{a_t (p_t)^2}{u_t} + b_t p_t + c_t u_t \right)
  \]

- Several ways to deal with the "more nonlinearity"

- Start-up cost is a concave in previous shut-down period length \( \tau \):
  \[
  cs(\tau) = V(1 - e^{-\lambda \tau}) + F \quad (\text{only required for integer } \tau)
  \]

- Convex hull description of the start-up cost fragment: extended formulation with temperature variables

- All these only deal with **partial fragments** of the (thermal) (single-)Unit Commitment problem

---

12 F., Gentile “Perspective cuts for a class of convex 0-1 mixed integer programs” *Math. Prog.* 2006
13 F., Gentile, Lacalandra “Tighter approximated MILP formulations for Unit Commitment Problems” *IEEE TPWRS* 2009
14 F., Gentile “A Computational Comparison of [...] : SOCP vs. Cutting Planes” *ORL* 2009
16 Silbernagl, Huber, Brandenberg “[...] MIP Unit Commitment by Modeling [...] Temperatures” *IEEE TPWRS* 2016
DP algorithm \(\rightarrow\) another MIP formulation

- arc variables \(y_{on}^{hk}\) on arc \(((h, \uparrow), (k, \downarrow))\), \(y_{off}^{hk}\) on arc \(((h, \downarrow), (k, \uparrow))\)

- node-arc matrix \(E\) for \(G\), \(s-d\) path RHS \(b\): flow conservation constraints
  \[
  Ey = b
  \] (14)

- power variables \(p_{t}^{hk}\) for \(t = h, \ldots, k\) for each “on” arc \(((h, \uparrow), (k, \downarrow))\)

\[
\begin{align*}
\overline{p}_{\text{min}}y_{on}^{hk} & \leq p_{h}^{hk} \leq \overline{y}_{on}^{hk} \\
\overline{p}_{\text{min}}y_{on}^{hk} & \leq p_{t}^{hk} \leq \overline{p}_{\text{max}}y_{on}^{hk} & t = h + 1, \ldots, k - 1 \\
\overline{p}_{\text{min}}y_{on}^{hk} & \leq p_{k}^{hk} \leq \overline{y}_{on}^{hk} \quad \forall (h, k) \tag{15}
\end{align*}
\]

\[
\begin{align*}
p_{t+1}^{hk} - p_{t}^{hk} & \leq y_{on}^{hk}\Delta_{+} & t = h, \ldots, k - 1 \\
p_{t}^{hk} - p_{t+1}^{hk} & \leq y_{on}^{hk}\Delta_{-} & t = h, \ldots, k - 1
\end{align*}
\]

(14)–(15) describes the convex hull if objective linear\(^{17}\)

- Slightly \(\neq\) version (independently obtained) use DP to separate cuts\(^{18}\)

---

\(^{17}\) F., Gentile “New MIP Formulations for the Single-Unit Commitment Problems with Ramping Constraints” IASI RR 2015

\(^{18}\) Knueven, Ostrowski, Wang “Generating Cuts from the Ramping Polytope for the Unit Commitment […]” IJOC 2018
About the new formulation

- \( O(n^2) \) binary + \( O(n^3) \) continuous variables, \( O(n^3) \) constraints

- Computational usefulness dubious (but perfect for Structured DW\(^{19}\))

- Convex hull proof uses polyhedral result, but no real reason for linearity

- In fact, “easy” convex generalisation\(^{20}\): add Perspective Reformulation
  \( \Rightarrow \) “perfect” formulation (convex hull)

- This is (or at least it should have been\(^{21}\)) clearly necessary

- Nonlinear generalization of known polyhedral result:
  appropriate star-composition of convex hulls gives the convex hull

---

\(^{19}\) F., Gendron “A stabilized structured Dantzig-Wolfe decomposition method” *Math. Prog.* 2013

\(^{20}\) Bacci, F., Gentile, Tavlaridis-Gyparakis “New MINLP Formulations for the Unit Commitment Problem […]” *OR* 2023

\(^{21}\) Bacci, F., Gentile “A Counterexample to an Exact […] Formulation for the Single-Unit Commitment […]” *IASI RR* 2019
More practical formulations I: $p_t$

- Idea 1: kill the many $p_{hk}^t$ entirely

- Obvious map between 3-bin variables and flow ones

$$x_t = \sum_{(h,k): t \in T(h,k)} y_{hk}^t, \quad \nu_t = \sum_{k \geq t} y_{tk}^t, \quad w_{t+1} = \sum_{h \leq t} y_{ht}^t$$

- Strengthen 3-bin formulation using the flow variables:

$$p_t - p_{t-1} \leq -l \sum_{h:h \leq t-1} y_{ht-1}^t + \Delta^+ \sum_{(h,k): t-1 \in T(h,k-1)} y_{hk}^t + \bar{t} \sum_{k:k \geq t} y_{tk}^t$$

$$p_{t-1} - p_t \leq -l \sum_{k:k \geq t} y_{tk}^t + \Delta^- \sum_{(h,k): t-1 \in T(h,k-1)} y_{hk}^t + \bar{u} \sum_{h:h \leq t-1} y_{ht-1}^t$$

$$l \sum_{(h,k): t \in T(h,k)} y_{hk}^t \leq p_t \leq u \sum_{(h,k): t \in T(h,k)} y_{hk}^t$$

$$p_t \leq \bar{l} \sum_{k:k \geq t} y_{tk}^t + \bar{u} \sum_{h:h \leq t} y_{ht}^t + \sum_{(h,k): h < t < k} \psi_{hk}^t y_{hk}^t$$

(some changes needed when $\tau^+ = 1$ and at the beginning of time)
More practical formulations II: Start-Up

- Idea 2: aggregate the many $p_{t}^{hk}$ somehow

- $p_{t}^{h} =$ started-up at $h$ (don’t care when shut-down), $p_{t} = \sum_{h:h \leq t} p_{t}^{h}$

- Modified formulation

\[
\begin{align*}
    p_{t}^{h} - p_{t-1}^{h} & \leq -ly_{ht-1}^{h} + \Delta^{+} \sum_{k:k \geq t} y_{hk}^{h} \\
    p_{t-1}^{h} - p_{t}^{h} & \leq \bar{u} y_{ht-1}^{h} + \Delta^{-} \sum_{k:k \geq t} y_{hk}^{h} \\
    p_{1}^{0} & \leq (\Delta^{+} + p_{0}) \sum_{k:1 \leq k} y_{0k}^{0} \\
    - p_{1}^{0} & \leq (\Delta^{-} - p_{0}) \sum_{k:1 \leq k} y_{0k}^{0} \\
    \lfloor \sum_{k:k \geq t} y_{hk}^{h} \rfloor & \leq p_{t}^{h} \leq u \sum_{k:k \geq t} y_{hk}^{h} \\
    p_{i}^{h} & \leq \bar{I} \sum_{k:k > h} y_{hk}^{h} + \min \{ \bar{I}, \bar{u} \} y_{hh}^{h} \\
    p_{t}^{h} & \leq \bar{u} y_{ht}^{h} + \sum_{k:k > t} \psi_{t}^{hk} y_{hk}^{h}
\end{align*}
\]

- Unfortunately did not work too well like “twin” Shut-Down\textsuperscript{22}

\textsuperscript{22} Bacci, F., Gentile “Start-up/Shut-down MINLP Formulations for the Unit Commitment […]” CTW2020, 2021
1 SMS++: design goals

2 Important motivation: energy optimization
   - Drilling down: Thermal Units
   - Scaling up: longer-term problems

3 SMS++: basic components

4 SMS++: existing Block and Solver

5 A glimpse to some results

6 Conclusions (Part I)
The tactical level: Seasonal Storage Valuation

- Mid-term (1y) cost-optimal management of water levels in reservoirs considering uncertainties (inflows, temperatures, demands, ...)

- Very large size, nested structure

- Perfect structure for Stochastic Dual Dynamic Programming\(^{23,24}\) with multiple UC inside

- Issue: SDDP needs dual variables, so a (tight) relaxation is needed

---

\(^{23}\) Pereira, Pinto “Multi-stage stochastic optimization applied to energy planning” *Math. Prog.*, 1991

The strategic level: Investment Layer

- Long-term (30y) optimal (cost, pollution, CO\textsubscript{2} emissions, . . . ) planning of production/transmission investments considering multi-level uncertainties scenarios (technology, economy, politics, . . .)

- Many scenarios, huge size, multiple nested structure $\implies$ multiple nested Benders’ or Lagrangian decomposition and/or SDDP

- TtBoMK no literature about it (no-one ever tried something this crazy)

- Rather nasty with SMS++ , at all doable without?
1 SMS++: design goals

2 Important motivation: energy optimization
   • Drilling down: Thermal Units
   • Scaling up: longer-term problems

3 SMS++: basic components

4 SMS++: existing Block and Solver

5 A glimpse to some results

6 Conclusions (Part I)
The basic object: Block

Block

physical representation

Block_1 | Block_2 | ...
Block – physical representation

- **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., $G$, $b$, $c$, $u$) and methods to change it

- Any # of sub-**Blocks** (recursively), possibly of specific type (e.g., :MMCFBlock:Block may have $k$ :MCFBlock:Block inside)

- Has a (possibly, nullptr) father *Block, possibly of specific type (but this is better avoided)
UCBlock, UnitBlock, NetworkBlock

UCBlock

UnitBlock_1 | UnitBlock_2 | ... | NetworkBlock_1 | ...

UnitBlock | ThermalUnitBlock | HydroUnitBlock | BatteryUnitBlock | IntermittentUnitBlock

NetworkBlock | DCNetworkBlock | ECNetworkBlock

T
UCBlock, UnitBlock, NetworkBlock recursively

UCBlock

UnitBlock₁ UnitBlock₂ ... NetworkBlock₁ ...

UnitBlock

NetworkBlock

ThermalUnitBlock

HydroUnitBlock BatteryUnitBlock IntermittentUnitBlock

HydroSystemBlock

HydroUnitBlock₁ ... PolyhedralFunctionBlock

DCNetworkBlock

ECNetworkBlock
SDDPBlock, StochasticBlock

SDDPBlock

ScenarioSet

scenario_1
scenario_2
...
scenario_k

StochasticBlock_1 | StochasticBlock_2 | ... | StochasticBlock_n

DataMapping

:Block
In many cases the Block semantic is irrelevant/ not useful; what is needed is an algebraic representation.

**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.

Abstract representation of sub-Block “belongs” to father Block but not vice-versa: where stuff is defined matters.
Block & abstract representation

Objective
Block
Constraint
Variable

Block

abstract representation

SC₁ SC₂ ...
DC₁ DC₂ ...

SV₁ SV₂ ...
DV₁ DV₂ ...

physical representation

Block₁ Block₂ ...

OF

A. Frangioni (DI – UniPi)
"Dynamic" abstract representation

- Abstract representation **not always (all) necessary** $\implies$ (partly) constructed on demand: `generate_abstract_variables()`, `generate_abstract_constraints()`, `generate_objective()`

- A Configuration (will see) for **partial construction** (:Block-specific)

- Configuration of `generate_abstract_variables()` can be used to **choose a specific formulation** (think $\neq$ ones in ThermalUnitBlock)

- Can change number and type of groups of Variable / Constraint and even number and type of sub-Block (recent addition, perhaps to be revised) $\implies$ no **Solver must be present**

- Generates **once the static part** (:Block must check for multiple calls)

- **Dynamic part** can be generated repeatedly, typically looking at primal / dual solution written in Variable / Constraint
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 $\implies$ copying a Variable makes a different Variable $\implies$ dynamic Variables always live in std::lists
- Wondering about using boost::stable_vectors instead / in addition
ThinVarDepInterface

- Generic concept of “something depending on a set of Variable”
- Specific implementation demanded to derived classes for efficiency
- Variable can be added / removed, accessed by index or pointer
- General mechanism for removing Variable, adding Variable depends on derived class
- “Abstract” STL-like iterator and const-iterator for access
- Other specific methods to describe / search / map the set
- Specific twist: a :ThinVarDepInterface is constructed after and destructed before “its” Variable, clear() method to avoid un-necessary data structure updating during destruction
ThinComputeInterface

- Generic concept of “something that can take time to compute()”
- Specific provisions for the fact that the compute() can:
  - end in several ways (OK, error, stopped, ...) and be resumed
  - be influenced by (std::vector of) int / double / std::string parameters, gathered in a `ComputeConfig::Configuration` object
  - terminate with approximate solutions, defined in quite general way
  - produce user-defined events triggered by conditions (time, iterations, ...)
- Defaults so that “simple” objects with no parameter do nothing
- Clear rules about effect of changes in the underlying object during and after compute() to allow for “reoptimization”
- Specific support for changes in `Variable value (which do not trigger a Modification, will see)` that change the result (e.g., `Function`, will see)
- General `compute_async()` returning a `std::future`

---

25 van Ackooij, F. “Incremental Bundle Methods Using Upper Models” SIOPT, 2018
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 requires (costly) compute() (:ThinComputeInterface)
- Knows which Block it belongs to
- Can be relaxed and enforced
- Fundamental design decision: “name” of a Constraint = its memory address \(\Rightarrow\) copying a Constraint makes a different Constraint \(\Rightarrow\) dynamic Constraints always live in std::lists
- Wondering about using boost::stable_vectors instead / in addition
Objective

- Abstract concept, does not specify its return value (vector, set, ...)
- Either minimized or maximized
- Depends from a set of Variable (:ThinVarDepInterface)
- Must be compute()-d 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)
- Objective of sub-Blocks summed to that of father Block if has same verse, otherwise min/max (clearly geared towards real-valued ones)
Abstract representation in UCBlock

linking constraints (demand, reserve, pollution, ...)

UCBlock

UnitBlock₁ | UnitBlock₂ | ... | NetworkBlock₁ | ...

UnitBlock  u[T] p[T]  NetworkBlock  s[n]

ThermalUnitBlock

HydroUnitBlock  BatteryUnitBlock  IntermittentUnitBlock

HydroSystemBlock

HydroUnitBlock₁ ... PolyhedralFunctionBlock

A. Frangioni (DI – UniPi)

SMS++ @ EdF

May 25-26, 2023 35 / 70
“Dynamic” abstract representation in ThermalUnitBlock

- As of release 5.2.0 (≈ yesterday) ThermalUnitBlock provides 6 different formulations: 3bin, T\textsuperscript{26}, DP, p\textsubscript{t}, SU, SD

- For all of them, Perspective Reformulation option is available via Perspective Cuts\textsuperscript{12} separated by generate\_dynamic\_variables()

- Selected via Configuration of generate\_abstract\_variables() or BlockConfig (will see)

- Most only useful for testing and scientific research, but different bound / time trade-offs may be useful in production (will see)

- DP would be perfect for Stabilised Structured DW\textsuperscript{1}, infrastructure all there but we need BundleSolver 2.0

\textsuperscript{26}Knueven, Ostrowski, Watson “On mixed-integer [...] formulations for the unit commitment problem” INFORMS JoC 2020
- **Real-valued Function**
- **Depends from** a set of Variable (:ThinVarDepInterface)
- **Must be** `compute()`-d w.r.t. the current value of the Variable, possibly a costly operation (:ThinComputeInterface)
- Approximate computation supported (:ThinComputeInterface)
C05Function and C15Function

- C05Function / C15Function deal with 1st / 2nd order information (not necessarily continuous)

- General concept of “diagonal linearization” (gradient, convex / concave / Clarke subgradient, ...) or “vertical linearization” (feasibility cut)

- Multiple linearizations produced at each evaluation (local pool)

- Global pool of linearizations for reoptimization:
  - convex combination of linearizations
  - “important linearization” (at optimality)

- PolyhedralFunction: C05Function simple & useful example

- C15Function supports (partial) Hessians

- Arbitrary hierarchy of Function possible / envisioned, any one that makes sense for application and / or solution method
Very basic stuff

- **ColVariable**: Variable: “value = one single real” (possibly $\in \mathbb{Z}$, possibly with $\leq / \geq 0 / \pm 1$ inherent bound constraints)

- **RowConstraint**: Constraint: “$l \leq a \text{ real} \leq u$” $\implies$ has dual variable (single real) attached to it

- **OneVarConstraint**: RowConstraint: “a real” = a single ColVariable $\equiv$ general bound constraints

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

- **FRealObjective**: RealObjective: “value” given by a **Function**

- **LinearFunction**: C15Function: a linear form in ColVariable

- **DQuadFunction**: C15Function: a separable quadratic form ...

- Many possible future developments: AlgebraicFunction, DenseLinearFunction, Matrix / VectorVariable, ...
Hybrids: LagBFunction and BendersBFunction

- Derive from both Block and Function

**LagBFunction** = Lagrangian function:
  - Objective of inner Block must be FRealObjective with LinearFunction or DQuadFunction
  - set of LinearFunction on ColVariable of inner Block \((Ax)\)
  - set of ColVariable not of inner Block \((\lambda)\)

= optimal value of inner Block with added term \(\lambda Ax\) in the Objective

**BendersBFunction** = value function:
  - set of (RHS and / or LHS) of RowConstraint in inner Block
  - set of LinearFunction \((Ax)\) on ColVariable not of inner Block

= optimal value of inner Block with changed RHS / LHS +Ax

Both require a : [CDA]Solver (will see), handle global / local pool with Solution (will see), handle translation of Modification (will see)
C05 Function in SDDPBlock

SDDPBlock

ScenarioSet

scenario₁
scenario₂
...
scenarioₖ

AbstractPath₁
AbstractPath₂
...
AbstractPathₙ

StochasticBlock₁ StochasticBlock₂ ... StochasticBlockₙ

DataMapping

BendersBlock

FRealObjective

BendersBBFunction

:Block

PolyhedralFunction
Block and Solver

- Any # of Solver attached to a Block to solve it, but each Solver attached to (at most) precisely one Block
- Solver for a specific Block can use the physical representation ➞ no need for explicit Constraint ➞ abstract representation of Block only constructed on demand
- However, Variable are always present to interface with Solver (this may change with physical Solution concept, under development)
- A general-purpose Solver uses the abstract representation
- Solver typically trigger dynamic Constraint / Variable generation (user cuts / lazy constraints / column generation)
- For a Solver attached to a (sub-)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)
Solver

- **Solver** = interface between users and algorithms solving one given Block (but the user of a Solver may well be another Solver)

- **Solver:** ThinComputeInterface, inherits and extends interface

- Solutions are written directly into the Variable of the Block (but could possibly directly produce Solution, under discussion)

- 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** (possibly, multiple) **dual solutions**
UCBlock and :Solver

linking constraints (demand, reserve, pollution, ...)

UnitBlock₁ | UnitBlock₂ | ... | NetworkBlock₁ | ...

UnitBlock | u[T] | p[T] | NetworkBlock | s[n]

ThermalUnitBlock

DPThermalUnitSolver

HydroUnitBlock | BatteryUnitBlock | IntermittentUnitBlock

HydroSystemBlock

HydroUnitBlock₁ | ... | PolyhedralFunctionBlock
SDDPBlock and :Solver

```
SDDPBlock

ScenarioSet

StochasticBlock1
StochasticBlock2
... 
StochasticBlockn

AbstractPath1
AbstractPath2
... 
AbstractPathn

SDDPSolver

DataMapping
BendersBlock
FRealObjective

BendersBFunction

PolyhedralFunction

ScenarioSet

scenario1
scenario2
... scenario_k

SDDPBlock

:Block

:CDASolver
```

A. Frangioni (DI – UniPi)
Any change is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification object.

\[ \text{anyone\_there()} \equiv \exists \text{interested Solver (Modification needed)} \]

Too coarse mechanism, must be improved ("types" of Modification)

Each Solver has the responsibility of cleaning up its list of Modification (smart pointers $\rightarrow$ memory eventually released)

Solver supposedly reoptimize to improve efficiency, which is easier if you can see all list of changes at once (lazy update)

Consistency easily ensured if Modification processed in the arrival order

A Solver may optimize the changes (Modifications may cancel each outer out ...), but its responsibility
### Modification

- Almost empty base class, then everything has its own derived ones
- Changes in each object “automatically” produce a Modification, e.g.
  - `BlockModAD::Modification` add / remove dynamic Variable / Constraint in Block
  - `VariableModification::Modification` (fix / unfix)
  - `ConstraintModification::Modification` (relax / enforce)
  - `ObjectiveModification::Modification` (change verse)
  - `[C05]FunctionMod[Var/Lin][Rngd/Sbst]` for “easy” changes (↑ / ↓ shift, range / subset of entries / variables changing ...)
  - “physical Modification” in :Block (e.g., `MCFBlockMod[Rngd/Sbst]` for changes in costs / capacities of range / subsets of arcs / nodes ...)

- Modification reach the Block (in fact Observer), from there dispatched to all attached Solver and to father Block (recursively)
- In general tells what changes but not how (go read the Block)
- `GroupModification` to (recursively) pack many Modification together ➞ different “channels” in Block
The whole SMS++ shebang (simplified)
“Abstract” Modification and the “Janus dilemma”

- Block are “Janus”, have “two faces” that must be kept in sync
- Nontrivial as abstract representation does not “know” the physical one
- Anyway (only) two different kinds of Modification (what changes):
  - physical Modification, only specialized Solver concerned
  - abstract (representation) Modification, only Solver using it concerned
- Abstract Modification used to keep both representations in sync
  \[ \Rightarrow \text{a single change may trigger more than one Modification} \]
  \[ \Rightarrow \text{concerns Block() mechanism to avoid this to repeat} \]
  \[ \Rightarrow \text{parameter in changing methods to avoid useless Modification} \]
- Specialized Solver disregard abstract Modification and vice-versa
- A Block declares which abstract changes it supports
  (throws std::exception if the wrong things change)
Change (new, beta)

- New concept to be introduced somehow after 5.2.0
- Change $\approx$ Modification + data: tells both what changes and how
- Can be de/serialize-d, not smart pointer
- Change need be defined for each :Block, typically just calling chg_*( ) methods in specialised interface
- Very nifty concept: undo-Change can be produced when Change apply()-ed to Block
- Not all Change can be undo-Change-d (may be too complex)
- Can be used for very general algorithmic schemes (B&X, . . . )
- AbstractChange possible, still to be implemented
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 de/serialize Block / Change capabilities
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 $\Rightarrow$ “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** \(\Rightarrow\) **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 final cleanup
**R³Block**

- Often reformulation crucial, but also relaxation or restriction: `get_R3_Block()` produces one, possibly using sub-Blocks’

- Obvious special case: copy (clone) should always work

- Available R³Blocks :Block-specific, a :Configuration needed

- R³Block completely independent *(new Variable/Constraint)*, useful for algorithmic purposes (branch, fix, solve, …)

- Solution of R³Block useful to Solver for original Block: `map_back_solution()` (best effort in case of dynamic Variable)

- Sometimes keeping R³Block 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 R³Block

- :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 Constraint/Variable in a Block

- FakeSolver:Solver stashes away all Modification, UpdateSolver:Solver immediately forwards/R^3Bs them

- ...
Outline

1. **SMS++ : design goals**

2. **Important motivation: energy optimization**
   - Drilling down: Thermal Units
   - Scaling up: longer-term problems

3. **SMS++ : basic components**

4. **SMS++ : existing Block and Solver**

5. A glimpse to some results

6. Conclusions (Part I)
Main Existing Block:

- MCFBlock/MMCFBlock: single/multicommodity flow
- BinaryKnapsackBlock (actually mixed-integer)
- CapacitatedFacilityLocationBlock (didactic)
- UCBlock for UC, several UnitBlock and NetworkBlock for components
- 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 wrapper to MCFClass\(^{27}\) for MCFBlock
- **ThermalUnitDPSolver** for ThermalUnitBlock (state-of-the-art)
- **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), Gurobi and HiGHS next
- **[Parallel]BundleSolver** : CDASolver: SMS++-native bundle code, still shares some code with\(^{28}\) (dependency to be removed), optimizes any (sum of) C05Function, several (but not all) state-of-the-art tricks
- **SDDPSolver**: wrapper for SDDP solver St0pt\(^{29}\) using StochasticBlock, BendersBFunction and PolyhedralFunction
- **SDDPGreedySolver**: greedy forward simulator for SDDPBlock

\(^{27}\) [https://github.com/frangio68/Min-Cost-Flow-Class](https://github.com/frangio68/Min-Cost-Flow-Class)

\(^{28}\) [https://gitlab.com/frangio68/ndosolver_fioracle_project](https://gitlab.com/frangio68/ndosolver_fioracle_project)

\(^{29}\) [https://gitlab.com/stochastic-control/St0pt](https://gitlab.com/stochastic-control/St0pt)
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)
Outline

1. SMS++: design goals

2. Important motivation: energy optimization
   - Drilling down: Thermal Units
   - Scaling up: longer-term problems

3. SMS++: basic components

4. SMS++: existing Block and Solver

5. A glimpse to some results

6. Conclusions (Part I)
Unit Commitment – some results

- ThermalUnitBlock provides different formulations, one “exact”

- ThermalUnitDPSolver provides efficient solution of 1-UC problems

- LagrangianDualSolver (using BundleSolver) + ThermalUnitDPSolver provides best trade-off between bound tightness and computational cost as size of the instance grows

<table>
<thead>
<tr>
<th>units</th>
<th>3bin time</th>
<th>3bin gap</th>
<th>DP time</th>
<th>DP gap</th>
<th>pt time</th>
<th>pt gap</th>
<th>LR time</th>
<th>LR gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.21</td>
<td>1.03</td>
<td>78.56</td>
<td>0.67</td>
<td>1.00</td>
<td>0.53</td>
<td>0.46</td>
<td>0.67</td>
</tr>
<tr>
<td>20</td>
<td>0.90</td>
<td>0.93</td>
<td>480.02</td>
<td>0.51</td>
<td>2.58</td>
<td>0.27</td>
<td>0.83</td>
<td>0.51</td>
</tr>
<tr>
<td>50</td>
<td>4.18</td>
<td>0.81</td>
<td>3836.78</td>
<td>0.08</td>
<td>9.92</td>
<td>0.09</td>
<td>1.19</td>
<td>0.08</td>
</tr>
</tbody>
</table>

- Just changing few lines in the BlockConfig and BlockSolverConfig
**Seasonal Storage Valuation – some results I**

- **SDDPSolver** requires convex problem: any of the above.

- Brazilian hydro-heavy system: 53 hydro (3 cascade), 98 thermal (coal, gas, nuclear), stochastic inflows (20 scenarios).

- Out-of-sample simulation: 1000 scenarios.

<table>
<thead>
<tr>
<th></th>
<th>Cont. relax.</th>
<th>Lag. relax.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost: Avg. / Std.</td>
<td>4.6023e+9 / 1.3608e+9</td>
<td>4.5860e+9 / 1.3556e+9</td>
</tr>
</tbody>
</table>

- Only 0.4% better, but **just changing a few lines in the Configuration** (Lagrangian about 4 times slower, but can be improved).
Seasonal Storage Valuation – some results II

- Single node (Switzerland)
- 60 stages (1+ year), 37 scenarios, 168 time instants (weekly UC)
- Units: 3 intermittent, 5 thermals, 1 hydro
- Out-of-sample simulation: all 37 scenarios to integer optimality

<table>
<thead>
<tr>
<th></th>
<th>Cont. relax.</th>
<th>Lag. relax.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost: Avg. / Std.</td>
<td>1.3165e+11 / 2.194e+10</td>
<td>1.2644e+11 / 2.167e+10</td>
</tr>
<tr>
<td>Time:</td>
<td>25m</td>
<td>7h30m</td>
</tr>
</tbody>
</table>

- Much longer, but:
  - simulation cost ≈ 30m per scenario, largely dominant
  - save 4% just changing a few lines in the configuration
  - LR time can be improved (ParallelBundleSolver not used)
Seasonal Storage Valuation – some results III

- A different single node (France)
- 60 stages (1+ year), 37 scenarios, 168 time instants (weekly UC)
- 83 thermals, 3 intermittent, 2 batteries, 1 hydro
- Out-of-sample simulation: all 37 scenarios to integer optimality

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.95e+11 / 1.608e+11</td>
<td>3.459e+11 / 8.903e+10</td>
</tr>
<tr>
<td>Time:</td>
<td>5h43m</td>
<td>7h54m</td>
</tr>
</tbody>
</table>

- Time not so bad (and 3h20m on average simulation per scenario) using ParallelBundleSolver with 5 threads per scenario
- That’s 14% just changing a few lines in the configuration
- Starts happening regularly enough (and lower variance) to be believable
**Investment Layer – some results**

- **Simplified version**: solve SDDP only once, run optimization with fixed value-of-water function + simulation (SDDPGreedySolver)

- **EdF EU scenario**: 11 nodes (France, Germany, Italy, Switzerland, Eastern Europe, Benelux, Iberia, Britain, Balkans, Baltics, Scandinavia), 20 lines

- **Units**: 1183 battery, 7 hydro, 518 thermal, 40 intermittent

- **78 weeks hourly (168h)**, 37 scenarios (demand, inflow, RES generation)

- **Investments**: 3 thermal units + 2 transmission lines.

- **Average cost**: original (operational) $6.510 \times 10^{12}$
  
  optimized (investment + operational) $5.643 \times 10^{12}$

  This is $\approx 1$ Trillion Euro, 15%

- **Running time**: ??? hours for value-of-water functions (EdF provided) + 10 hours (4 scenarios in parallel + ParallelBundleSolver with 6 threads) for the investment problem
Investment Layer – some results II

- Simplified version (fixed value-of-water with continuous relaxation)
- Same 11 nodes, 19 lines
- **Less** units: 7 hydros, 44 thermals, 24 batteries, and 42 intermittent
- **More** investments: 82 units + 19 transmission lines.
- 78 weeks hourly (168h), 37 scenarios (demand, inflow, RES generation)
- Average cost: original (operational) \(3.312 \times 10^{12}\) 
  optimized (investment + operational) \(1.397 \times 10^{12}\)
- This is \(\approx 2\) Trillion Euro, 137%
- Running time: 48 hours for value-of-water functions (2 nodes = 96 cores) 
  + 5h 20m to solve the investment problem (1 nodes = 48 core)
Investment Layer – some results III

- Same simplified version as above

- EdF EU scenario: 14 nodes (France, Germany, Italy, Switzerland, Eastern Europe, Benelux, Iberia, Britain, Balkans, Baltics, Denmark, Finland, Sweden, Norway), 28 lines

- Units: 62 thermals, 54 intermittent, 8 hydros, 39 batteries

- 78 weeks hourly (168h), 37 scenarios (demand, inflow, RES generation)

- Investments: 99 units of all kinds + all transmission lines

- Average cost: original (operational) $3.465 \times 10^{12}$
  
  optimized (investment + operational) $4.708 \times 10^{11}$

  one order of magnitude saving (suspect most value of lost load)

  636% better investing on just 4 lines and 10 hydrogen power plants

- Running time: 7 hours on 48 cores, 375GB of RAM
Investment Layer – the (Little-)Big Kahuna results

- **The true version**: value-of-water recomputed anew for each investment
- But still **simplified**: only one scenario (long way to go)
- As usual, SDDP with Continuous or Lagrangian
- **One node** (48 core, 375Gb) not enough, must either MPI-distribute over many or run on larger nodes (48 core, 800Gb of RAM suffice)
- After \( \approx 648h \) (several time-outs&resumes, maintenance breaks, …) simulation-based: investment + operational \( 4.708e+11 \)
  SDDP-based: investment + operational \( 4.537e+11 \) (17 billion€ saving)
- Perhaps better idea: warm-start SDDP-based from simulation-based, got an even better \( 4.325e+11 \) to start with in 24h (avoid Cplex)
- warm-started SDDP-based currently running, no more results to show
- Two small-ish (\( \approx 10000h \)) CINECA grants to debug&test, a much bigger one needed to run the real Big Kahuna (one more decomposition level)
- But we are getting there, thanks to SMS++
Conclusions
(Part I)
Real-world (energy) optimization models are large and very complex.

Even writing them right is challenging, not to mention solving them.

Normal modelling languages/systems don’t take this seriously enough.

Perhaps wise: taking it seriously $\rightsquigarrow$ complex requirements.

Especially true for natively supporting sophisticated algorithms (decomposition, tomorrow), particularly parallel ones.

SMS++ is there, actively developed.

The only game in town for the extremely challenging use cases.

Only serious C++ programmers need apply, exercises next.