Ludicrously Large-Scale Stochastic Energy Optimization in SMS++

Antonio Frangioni[1]

Rafael Durbano Lobato[2]

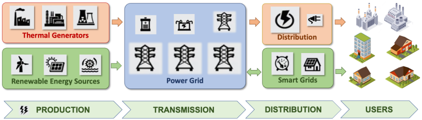
. . . & many others

[1]Dipartimento di Informatica, Università di Pisa, frangio@di.unipi.it

[2]Chamber of Electric Energy Commercialization of Brazil, rafael.lobato@gmail.com

The second HEXAGON workshop on power grids

September 10, 2025 – Benevento, Italy



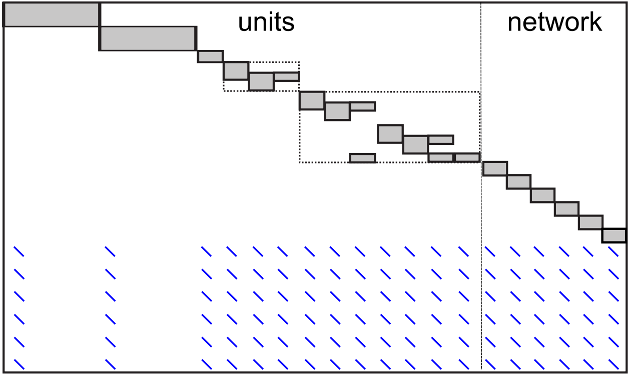
Outline

Energy Optimization bottom-up: Unit Commitment

- I don't need to convince you that energy optimization is important, but which of the many energy optimization problems??
- Operational level = Unit Commitment[1]: schedule generating units over time horizon (hours / 15m in day / week) to satisfy (forecasted) demand
- 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, OTS, ...) and reliability (primary / secondary reserve, $n-1$ units, ...)

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

Many Different Structures Already



- Many different structures: thermal units[2], hydro units[3], Energy Communities[4], stochastic[5], AC-OPF[6], OTS[7], ...

[2] Bacci, F., Gentile, Tavlaridis-Gyparakis "New MINLP Formulations for the Unit Commitment Problem [...]" *OR* 2024

[3] van Ackooij et. al. "Shortest path problem variants for the hydro unit commitment problem" *Elec. Notes Disc. Math.* 2018

[4] Fioriti, F., Poli "Optimal Sizing of Energy Communities with Fair Revenue Sharing [...]" *Applied Energy* 2021

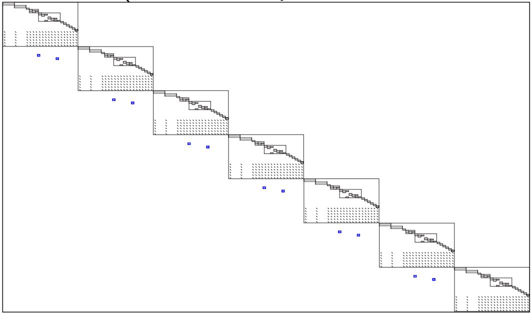
[5] Scuzziato, Finardi, F. "Comparing Spatial and Scenario Decomposition for Stochastic [...]" *IEEE Trans. Sust. En.* 2018

[6] Bienstock, Escobar, Gentile, Liberti "[...] formulations for the alternating current optimal power flow" *Ann. O.R.*, 2022

[7] Numan et. al. "The role of optimal transmission switching in enhancing grid flexibility: A review" *IEEE Access*, 2023

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 (one UC per stage)
- Perfect structure for Stochastic Dual Dynamic Programming[8,9]
- SDDP needs dual variables, but Lagrangian dual convexifies[10,11]

[8] Pereira, Pinto "Multi-stage stochastic optimization applied to energy planning" *Math. Prog.*, 1991

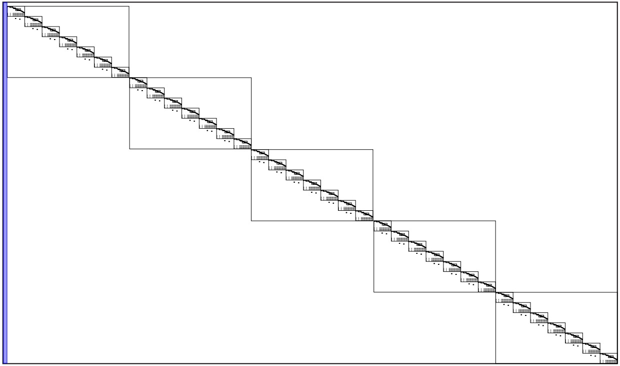
[9] van-Ackooij, Warin "On conditional cuts for Stochastic Dual Dynamic Programming" *EURO J. on Comp. Opt.*, 2020

[10] Lemaréchal, Renaud "A geometric study of duality gaps, with applications" *Math. Prog.* 2001

[11] F. "About Lagrangian Methods in Integer Optimization" Annals of O.R., 2005

Energy System Investment

- Investment on generating units / transmission lines
- Using stochastic independent representative years to evaluate system cost



- Very few investment variables, can be taken continuous, identical copies
- Would be perfect for Benders'-like[12,13], if given dual information

[12] Geoffrion "Generalized Benders Decomposition" *JOTA*, 1972

[13] van Ackooij, F., de Oliveira "Inexact Stabilized Benders' Decomposition Approaches [. . .]" *COAP*, 2016

Mathematics of convex investment problems

- Set N of generation / distribution units, κ_i identical copies of each $i \in N$
- For $\kappa = [\,\kappa_i\,]_{i \in N}$, investment cost $F(\,\kappa\,)$ ("easy") and operational cost

$$
\begin{aligned}
O(\,\kappa\,) = \min \; & \textstyle\sum_{i \in N} \sum_{j=1}^{\kappa_i} c_i(\,x_{i,j}\,) \\
\text{s.t. } & x_{i,j} \in X_i && j = 1, \ldots, \kappa_i \;,\; i \in N \\
& \textstyle\sum_{i \in N} \sum_{j=1}^{\kappa_i} A_i x_{i,j} \geq d
\end{aligned}
\tag{1}
$$

- Everything convex $\Longrightarrow$ all $i \in N$ produce identically at optimality $\Longrightarrow$

$$
\begin{aligned}
O(\,\kappa\,) = \min \; & \textstyle\sum_{i \in N} \kappa_i c_i(\,x_i\,) \\
\text{s.t. } & x_i \in X_i && i \in N \\
& \textstyle\sum_{i \in N} \kappa_i A_i x_i \geq d
\end{aligned}
\tag{2}
$$

- Extends to stochastic setting (S = scenarios, $\mathcal{N}$ = nonanticipativity)

$$
\begin{aligned}
O(\,\kappa\,) = \min \; & \textstyle\sum_{i \in N} \kappa_i \sum_{s \in S} \pi^s c_i^s(\,x_i^s\,) \\
\text{s.t. } & x_i^s \in X_i^s && i \in N \;,\; s \in S \\
& \textstyle\sum_{i \in N} \kappa_i \sum_{s \in S} A_i^s x_i^s \geq d \;\;,\;\; x \in \mathcal{N}
\end{aligned}
\tag{3}
$$

A triply clever trick

- Investment problem $\min\{\, F(\kappa) + O(\kappa) \;:\; \kappa \in K \,\}$: extremely hard as even (2) / (3) hard ((1) harder), since convexity assumption untrue
- Lagrangian relaxation triply clever:

  $$\phi(\lambda, \kappa) = \lambda d + \textstyle\sum_{i \in N} \kappa_i \min\{\, c_i(x_i) - \lambda A_i x_i \;:\; x_i \in X_i \,\}$$

  - decomposes into (many, easier, smaller) independent subproblems
  - automatically convexifies c and X[11]
  - $\phi(\lambda, \kappa)$ is concave in λ and affine in κ
- Convexified version: $\underline{O}(\kappa) = \max\{\, \phi(\lambda, \kappa) \;:\; \lambda \geq 0 \,\} = \phi(\lambda^*(\kappa), \kappa)$
- Convexified Investment Problem: $\min\{\, F(\kappa) + \underline{O}(\kappa) \;:\; \kappa \in K \,\}$ possibly the best trade-off between computability and accuracy
- Crucial: $[\, c_i(x_i^*(\lambda^*(\kappa))) - \lambda^*(\kappa) A_i x_i^*(\lambda^*(\kappa)) \,]_{i \in N} \in \partial \underline{O}(\kappa)$[14]
  $\Longrightarrow$ can use bundle methods[??] or stabilised Benders' ones[15]

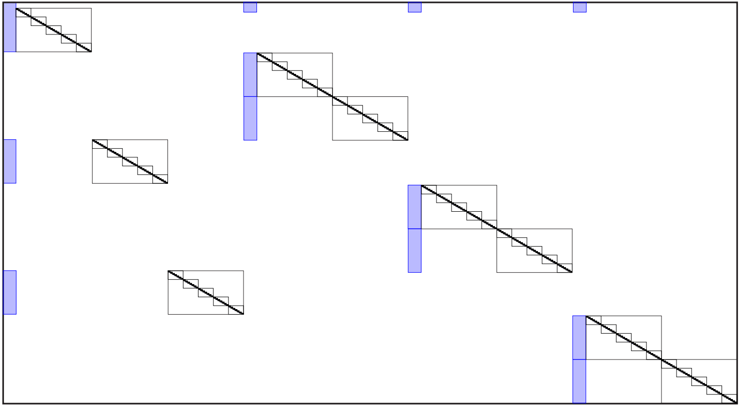
---

[14] van Ackooij, Oudjane "On supply and network investment in power systems" *4OR*, 2024

[15] van Ackooij, F., de Oliveira "Inexact Stabilized Benders' Decomposition Approaches [...]" *COAP*, 2016

Strategic Energy System Investment ("the Big Kahuna")

- Long-term (30y) optimal (cost, pollution, CO_2 emissions, ...) planning of production/transmission investments considering multi-level uncertainties scenarios (technology, economy, politics, ...)



- Many scenarios, huge size, multiple nested structure $\Longrightarrow$ multiple nested Benders' or Lagrangian decomposition and/or SDDP$^{??}$

Outline

A HUGE LOT OF
ELBOW GREASE,
BLODSHED AND TEARS

or

Quite a lot of elbow grease and

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

"For algorithm developers, from algorithm developers"

- Open source (LGPL3)
- 1 "core" repo, 1 "umbrella" repo, 12+ problem and/or algorithmic-specific repos (public, more in development), tests & tools
- Extensive Doxygen documentation https://smspp.gitlab.io
- But no real user manual as yet (except myself)

What SMS++ is

- A core set of C++-20 classes implementing a modelling system that:
  - explicitly supports the notion of Block ≡ nested structure
  - separately provides "semantic" information from "syntactic" details (list of constraints/variables ≡ one specific formulation among many)
  - allows exploiting specialised 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: a community-building effort as much as a (suite of) software product(s)

What SMS++ is not

- An algebraic modelling language: `Block` 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 0.5.2, lots of further development ahead, significant changes in (parts) of interfaces actually expected (although current `Block` / `Solver` very thoroughly tested)
- Interfaced with many existing solvers: `Cplex`, `SCIP`, `MCFClass`, `StOpt` (but the list recently grew with `Guorbi`, `HiGHS` and `LEMON`)
- Ripe with native structure-exploiting solvers: `LagrangianDualSolver` and `SDDPSolver` for now (although the list should hopefully grow)

Outline

It starts deceivingly simple

```
ThermalUnitBlock : Block
```

$$\min \sum_{i \in P} c^i(\mathbf{p}^i, \mathbf{u}^i) = \sum_{i \in P} \left(c^i(\mathbf{u}^i) + \sum_{t \in \mathscr{T}} c^i_t(p^i_t) \right)$$

$$\bar{p}^i_{\min} u^i_t \leqslant p^i_t \leqslant \bar{p}^i_{\max} u^i_t \quad t \in \mathscr{T}$$
$$p^i_t \leqslant p^i_{t-1} + u^i_{t-1} \Delta^i_+ + (1 - u^i_{t-1}) \bar{l}^i \quad t \in \mathscr{T}$$
$$p^i_{t-1} \leqslant p^i_t + u^i_t \Delta^i_- + (1 - u^i_t) \bar{u}^i \quad t \in \mathscr{T}$$
$$u^i_t \geqslant u^i_r - u^i_{r-1} \quad t \in \mathscr{T}, \quad r \in [t - \tau^i_+, t-1]$$
$$u^i_t \geqslant 1 - u^i_{r-1} - u^i_r \quad t \in \mathscr{T}, \quad r \in [t - \tau^i_-, t-1]$$
$$u^i_t \in \{0, 1\} \quad t \in \mathscr{T}$$

```
T   a[] b[] c[]  Dp[]
p_min[] p_max[] Dm[]
```

- For a man with a hammer everything is a nail

It starts deceivingly simple

`ThermalUnitBlock : Block`

$$\min \sum_{i\in P} c^i(\mathbf{p}^i,\mathbf{u}^i) = \sum_{i\in P}\left(c^i(\mathbf{u}^i) + \sum_{t\in\mathscr{T}} c^i_t(p^i_t)\right)$$

$$\bar{p}^i_{\min} u^i_t \leqslant p^i_t \leqslant \bar{p}^i_{\max} u^i_t \quad t\in\mathscr{T}$$

$$p^i_t \leqslant p^i_{t-1} + u^i_{t-1}\Delta^i_+ + (1-u^i_{t-1})\bar{l}^i \quad t\in\mathscr{T}$$

$$p^i_{t-1} \leqslant p^i_t + u^i_t\Delta^i_- + (1-u^i_t)\bar{u}^i \quad t\in\mathscr{T}$$

$$u^i_t \geqslant u^i_r - u^i_{r-1} \quad t\in\mathscr{T},\quad r\in\left[t-\tau^i_+, t-1\right]$$

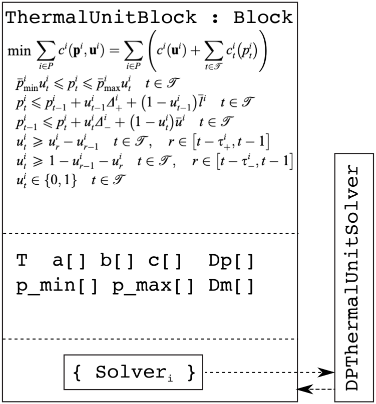
$$u^i_t \geqslant 1 - u^i_{r-1} - u^i_r \quad t\in\mathscr{T},\quad r\in\left[t-\tau^i_-, t-1\right]$$

$$u^i_t \in \{0,1\} \quad t\in\mathscr{T}$$

```
T   a[] b[] c[]  Dp[]
p_min[] p_max[] Dm[]
```

- For a man with a solver everything is a `Block` (call me blockhead )
- `Block` = abstract class representing the general concept of "(fragment of) mathematical model with a well-understood semantic"
- Each `:Block` a model with specific structure: `ThermalUnitBlock : Block` = a single-(thermal)-unit commitment problem

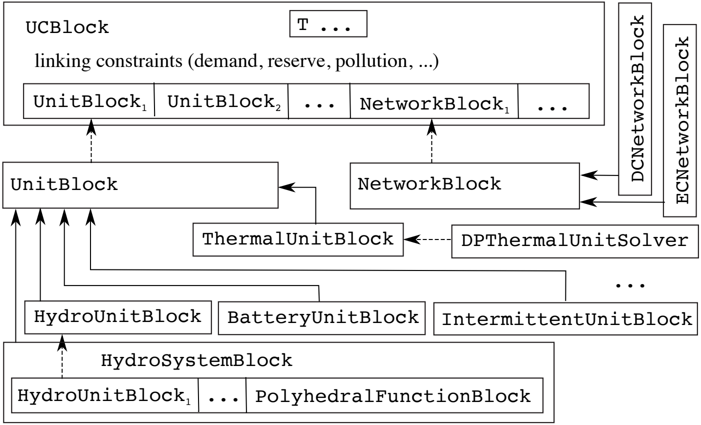
A Block ∃ (≈) because a (specialised) Solver ∃



- Any number of Solver can be attached to a Block
- Any specific :Block (e.g., ThermalUnitBlock) can have specialised $\Longrightarrow$ fast :Solver (e.g., DPThermalUnitSolver[16])
- Can be wrapper classes to efficient existing (C++) libraries

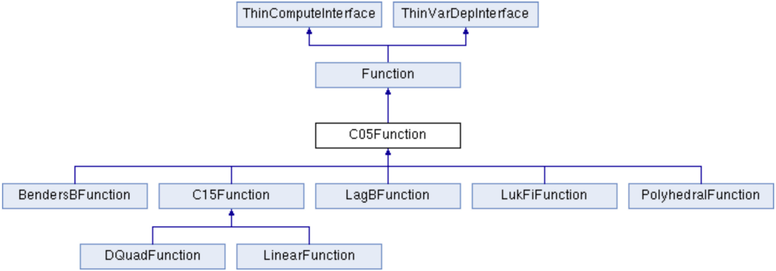
[16] F., Gentile "Solving Nonlinear Single-Unit Commitment Problems with Ramping Constraints" *Op. Res.*, 2006

A Block is (almost) always just a (small) part



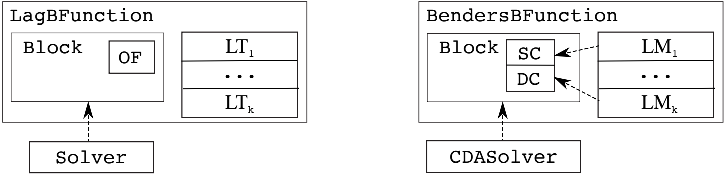
- A Block can have any # of sub-Block, recursively (Block *); e.g., UCBlock : Block has k :UnitBlock and T :NetworkBlock recursively
- Problem data split between them (energy constraints only in UCBlock)

Another necessary step: Function



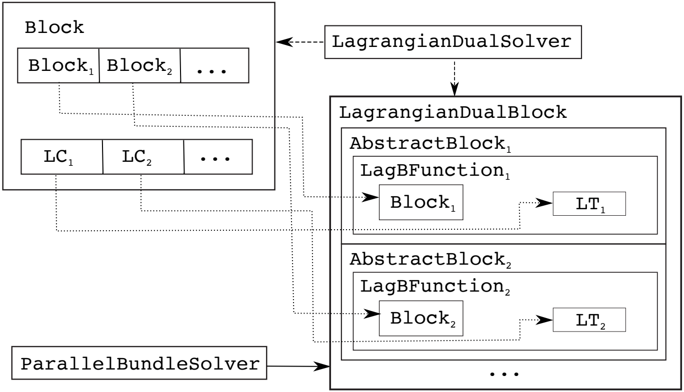
- Real-valued `Function` of a set of `[Col]Variable` (a single $\mathbb{R}$ / $\mathbb{Z}$)
- Must be `compute()`-d w.r.t. the current value of the `[Col]Variable`, possibly a costly operation (`:ThinComputeInterface`)
- `C05Function` / `C15Function` have (not necessarily continuous) 1st / 2nd order information (vertical / diagonal linearizations)
- Local / global pools of linearizations
- "Easy" `Function` (linear, quadratic, polyhedral, ...) with no overhead

LagBFunction & BendersBFunction



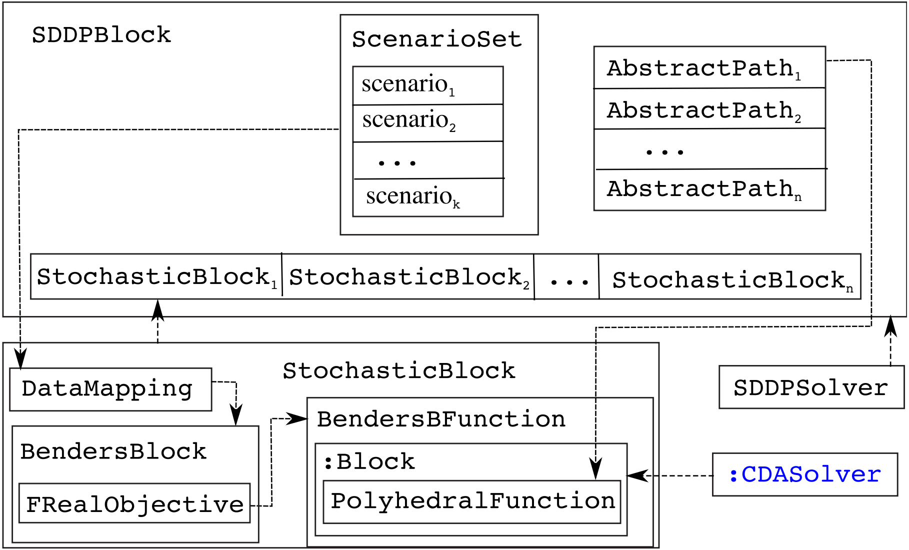
- LagBFunction $\equiv$ dual function $\varphi(\lambda) = \min\{ f(x) + (\lambda LT)x \, : \, x \in X \}$ for (almost) any Block (B) $\min\{ f(x) \, : \, x \in X \}$
- BendersBFunction $\equiv$ value function
  $v(y) = \min\{ f(x) \, : \, g(x) \leq LMy \, : \, x \in X \}$
  for (almost) any Block (B) $\min\{ f(x) \, : \, g(x) \leq 0 \, , \, x \in X \}$
- Both are :Block and :C05Function, with (B) being the only sub-Block
- Use generic [CDA]Solver to compute() ($\approx$ just call its compute())
- Store pools of primal / dual Solution corresponding to linearizations
- Any change in (B) is mapped in changes of F-values / the pools

All this ↦ LagrangianDualSolver



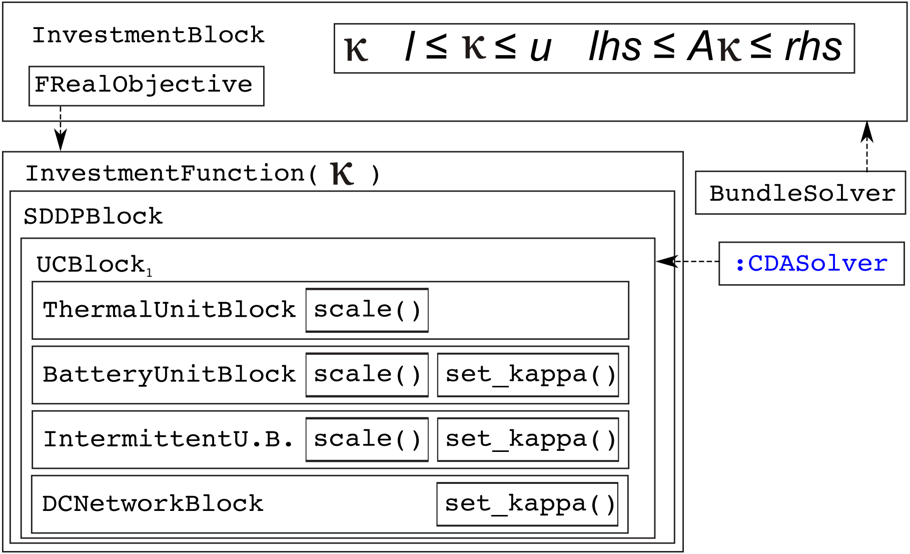
- Forms (hidden) `LagrangianDualBlock`, attaches parallel[17] `Solver`
- Provides primal (convexified ≡ "better"[10,11]) and dual solutions
- Good foundations for heuristic approaches[18,19] & the next steps

[17] Cappanera, F. "[...] Parallelization of [...] Algorithm for Multi-Commodity Flow Problems" *INFORMS JoC*, 2003
[18] Borghetti, F., Lacalandra, Nucci "Lagrangian [...] for Hydrothermal Unit Commitment", *IEEE Trans. Power Sys.* 2003
[19] Scuzziato, Finardi, F. "Solving Stochastic [...] Unit Commitment with [...] Lagrangian Solutions" *IJEPES*, 2021



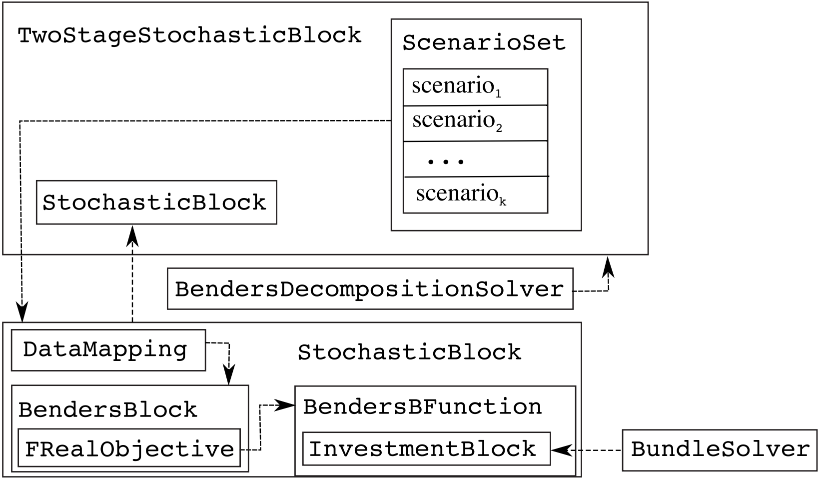
- SDDPSolver: wrapper for StOpt[20] + SDDPGreedySolver (simulator)

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



- Scaling a :Block a general concept, may be upcasted to base Block

Strategic Investment Problem in SMS++-speak



- Not all here yet, `TwoStageStochasticBlock` still under active development, `BendersDecompositionSolver` yet to come
- Clearly extremely challenging problem, need all the help we can get

SMS++ support to (coarse-grained) parallel computations

- `Block` can be (write) `lock()`-ed to ensure atomic changes
- `lock()`-ing a `Block` automatically `lock()`s all inner `Block` (recursively)
- Analogously for `read_lock()`, any # of concurrent reads
- `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)
- Write starvation not handled yet
- `Solver`'s `compute()` must be thread-safe (`std::recursive_mutex`)
- `Solver`/`ThinComputeInterface` can be "lent ID" (solving a sub-`Block`)
- `Solver`'s `list<Modification>` under an "active guard" (`std::atomic`)
- General `State` of `Solver` for checkpointing (and reoptimization)
- New `Change` concept: `Modification` + data, automatic undo_`Change`, can be de/serialize-d on `netCDF` file as everything $\Longrightarrow$ message-passing distributed `Solver` available one day (soon-ish?)

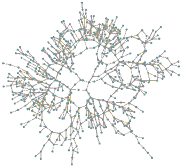
Outline

QUITE SOME ELBOW GREASE AND netCDF FILES MANGLING

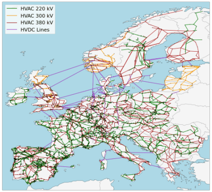
or

Some elbow grease and

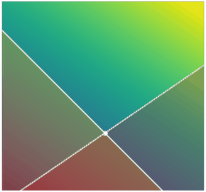
PyPSA



PyPSA-Eur



Linopy



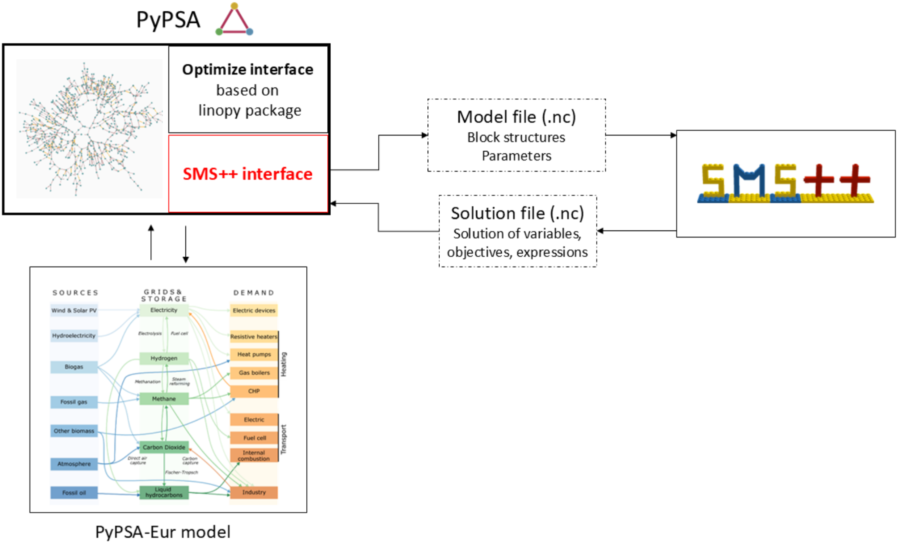
https://pypsa.org

+

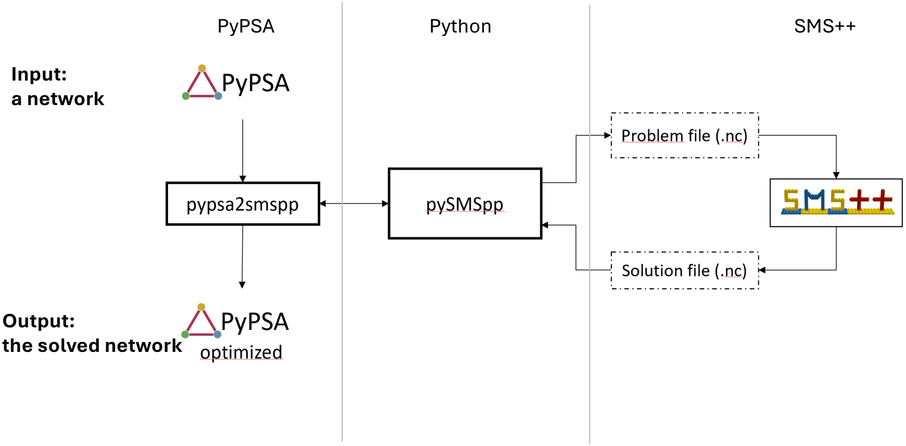
RESILIENT

https://resilient-project.github.io

What the project wants to do



- Modelling prowess of PyPSA + solution prowess of SMS++

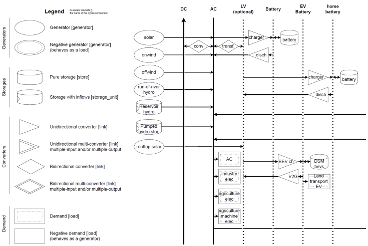


- Requires careful mapping of PyPSA network $\mapsto$ `SMS++:Block`
- Especially important for freshly developed PyPSA stochastic extension

A glimpse of the mapping

Equation	Symbol	Generator	Link	Line	Storage unit	Store	Condition	Example
Size bound	SB	X	X	X	X	X		$\underline{G}_{i,r} \leq G_{i,r} \leq \overline{G}_{i,r}$
Modularity	MD	X	X	X	X	X		$G_{i,r} = \underline{G}^{mod}_{i,r} n_{i,r}$
Power bound	PB	X	X	X	X	X		$\underline{g}_{i,r,t} G_{i,r} \leq g_{i,r,t} \leq \overline{g}_{i,r,t} G_{i,r}$
Power unit commitment	PB_{UC}	X	X				committable	$\delta_{i,r,t} \underline{g}_{i,r,t} G_{i,r} \leq g_{i,r,t} \leq \delta_{i,r,t} \overline{g}_{i,r,t} G_{i,r}$
Minimum time	MT	X	X					$\sum_{t'=t}^{t+T_{minup}} \delta_{k,t'} \geq T_{minup} (\delta_{k,t} - \delta_{k,t-1})$
Total energy produced	PSUM	X	X					$E^{sum}_{min} \leq \sum_{t \in T} w^G_t g_{i,r,t} \leq E^{sum}_{max}$
Start up/shut down cost	SC	X	X					$suc_{k,t} \geq suc_k (\delta_{k,t} - \delta_{k,t-1})$
Ramp up/down	RUD	X	X					$(g_{i,r,t} - g_{i,r,t-1}) \leq ru_{i,r} G_{i,r}$
Kirchhof's law	KL			X				$\sum_l C_{l,c} x_l p_{l,t} = 0$
Line losses	LL			X				$P^{loss}_l = \alpha_l + \beta_l p_{l,t}$
Energy storage level	ESL				X	X		$e_{i,s,t} = e_{i,s,t-1} + w^S_t h_{i,s,t}$
Energy storage bound	ESB				X	X		$0 \leq e_{i,s,t} \leq E_{i,s} e^{max}_{i,s}$
Initial energy level	IEL_S				X	X		$e_{i,s,0} = e_{i,s,init}$
Cyclic energy level	IEL_{CS}				X	X	cyclic_state_of_charge	$e_{i,s,0} = e_{i,s,\|T\|}$

	A	B	C	D	E
1	Technology name	Category	Physical compone	Option	Carrier
2	co2 atmosphere	co2	N		co2
3	Co2 storage	co2	Y		co2 stored
4	Sequestration link	co2	N		co2 sequestred
5	Sequestration store (e.g. underground)	co2	Y/N		co2 sequestred
6	CO2 vent co2 from storages	co2	?	co2_vent	co2 vent
7	CO2 pipelines	co2	Y	co2_network	CO2 pipeline
8	Allam (gas) cycle	electricity	Y	allam	allam
9	Direct Air Capture	co2	Y	dac	co2
10	Conventional generators	electricity	Y	conventional_generation	electricity
11	Haber-Bosch process	ammonia	Y	ammonia	Haber-Bosch
12	Ammonia cracker	ammonia	Y	ammonia	ammonia cracker
13	Ammonia storage	ammonia	Y	ammonia	ammonia store
14	Electricity distribution	electricity	Y	electricity_distribution_grid	low voltage
15	rooftop solar	electricity	Y	electricity_distribution_grid	solar rooftop



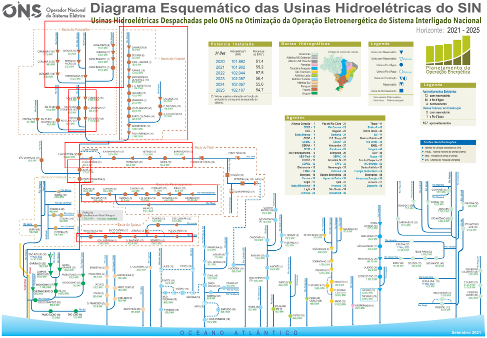
- . . . but no-one needs bother besides us (or new features developers)

Outline

Seasonal Storage Valuation – some results I

- `SDDPSolver` requires convex problem: any of the above $\Longrightarrow$ useful tricks (continuos relaxation forward, Lagrangian relaxation backward)

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



- Out-of-sample simulation: 1000 scenarios

	Continuous	Lagrangian
Cost: Avg. / Std.	4.6023e+9 / 1.3608e+9	4.5860e+9 / 1.3556e+9

- 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

	Continuous	Lagrangian
Cost: Avg. / Std.	1.3165e+11 / 2.194e+10	1.2644e+11 / 2.167e+10
time	25m	7h30m

- 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 single node (France, guess why . . .)
- 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

	Continuous	Lagrangian
Cost: Avg. / Std.	3.951e+11 / 1.608e+11	3.459e+11 / 8.903e+10
time	5h43m	7h54m

- 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 to be trusted

Seasonal Storage Valuation – some (recent) results IV

- Two dedicated top-level servers with (each) 2 AMD Epyc 9654 (2.4Ghz, 96 cores, 192 threads, 384MB cache) with 1.5TB RAM (DDR5-4800)
- 78 stages, 37 scenarios (uncertainties in demand, inflow, intermittent generation), 84 time steps
- 23 nodes (Balkans, Baltics, Benelux, Britain, Eastern Europe, France, Germany, Italy, Portugal, Scandinavia, Switzerland, Spain, "outside") + 49 lines, 98 thermals, 18 hydros, 60 intermittent, 22 batteries
- Deterministic equivalent would be $36^{78} \approx$ `2.5e+121` nodes, each `6.2e+5` variables (`2.5e+5` binary) and `9.3e+5` constraints (do the math ...)
- 37 processes (1 per scenario), `ParallelBundleSolver` with 5 threads

Seasonal Storage Valuation – a glimpse to the dynamic

- A sample of evolution, continuous VS Lagrangian relaxation
- Using generated SDDP cuts for future-value-of-water as iteration nature

# cuts	Continuous	Lagrangian	gap (%)
50	95030821478	94327497967	0.75
100	92798901757	92381611565	0.45
200	92857020055	93146422115	-0.31
300	92500105416	92231190639	0.29
400	92434329595	92297148794	0.15
500	92446972425	92572975779	-0.14
best	92434329595	92231190639	0.22
Time	18h20m	9d19h	

- . . . but gap is 2e+8 €: not a bad gain for 10 days' work
- Time to boldly take the last step

Energy System Investment Problem – some results I

- 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) 6.510e+12
  optimized (investment + operational) 5.643e+12
- This is ≈ 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

Energy System Investment Problem – 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) 3.312e+12
  optimized (investment + operational) 1.397e+12
- This is ≈ 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)

Energy System Investment Problem

- The true version: value-of-water recomputed anew for each investment
- Still simplified: only one scenario (long way to go, but `TwoStageStochasticBlock` and `BendersDecompositionSolver` currently under active development, we'll get there eventually)
- EU scenario: 14 nodes (France, Germany, Italy, Switzerland, Eastern EU, Benelux, Iberia, Britain, Balkans, Baltics, Denmark, Finland, Sweden, Norway), 28 lines, 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
- Requires extensive support for checkpointing and restarts (but less than on CINECA machines that had 24h time limit)

Energy System Investment Problem: first steps

- Huge problem, so three steps approach
  - solve the Seasonal Storage Valuation with initial system (no investment)
  - solve Energy System Investment Problem with fixed value-of-water function out of SDDP (simulation-based optimization)
  - improve investment by dynamically recomputing value-of-water at every iteration

- Original system cost: (operational) 3.467e+12

  Optimized cost: operational 4.505e+11 + investment 2.284e+11 = total 6.789e+11

- Half an order of magnitude saving (suspect most value of lost load), 511% better investing on just 4 lines and 10 hydrogen power plants

- Running time: 15h18m for future cost function of the original system, 5h18m simulation-based investment problem (74 threads max)

The Little-Big Kahuna results

- Starting from previous solution, optimize with variable value-of-water
  - iteration 0: op. 4.505e+11 + inv. 2.284e+11 = total 6.789e+11 (1.8h) (very sparse investment decision)

The Little-Big Kahuna results

- Starting from previous solution, optimize with variable value-of-water
  - iteration 0: op. 4.505e+11 + inv. 2.284e+11 = total 6.789e+11 (1.8h) (very sparse investment decision)
  - iteration 1: op. 6.670+10 + inv. 5.635e+12 = total 5.702e+12 (22h) (almost completely dense investment decision)

The Little-Big Kahuna results

- Starting from previous solution, optimize with variable value-of-water
  - iteration 0: op. 4.505e+11 + inv. 2.284e+11 = total 6.789e+11 (1.8h) (very sparse investment decision)
  - iteration 1: op. 6.670+10 + inv. 5.635e+12 = total 5.702e+12 (22h) (almost completely dense investment decision)
  - iteration 2: op. 1.505e+12 + inv. 2.221e+11 = total 1.727e+12 (21h) (less dense investment decision)

The Little-Big Kahuna results

- Starting from previous solution, optimize with variable value-of-water
  - iteration 0: op. 4.505e+11 + inv. 2.284e+11 = total 6.789e+11 (1.8h) (very sparse investment decision)
  - iteration 1: op. 6.670+10 + inv. 5.635e+12 = total 5.702e+12 (22h) (almost completely dense investment decision)
  - iteration 2: op. 1.505e+12 + inv. 2.221e+11 = total 1.727e+12 (21h) (less dense investment decision)
  - iteration 3: op. 2.286e+11 + inv. 7.263e+11 = total 9.549e+11 (20h) (less dense investment decision)

The Little-Big Kahuna results

- Starting from previous solution, optimize with variable value-of-water
  - iteration 0: op. 4.505e+11 + inv. 2.284e+11 = total 6.789e+11 (1.8h) (very sparse investment decision)
  - iteration 1: op. 6.670+10 + inv. 5.635e+12 = total 5.702e+12 (22h) (almost completely dense investment decision)
  - iteration 2: op. 1.505e+12 + inv. 2.221e+11 = total 1.727e+12 (21h) (less dense investment decision)
  - iteration 3: op. 2.286e+11 + inv. 7.263e+11 = total 9.549e+11 (20h) (less dense investment decision)
  - iteration 4:

The Little-Big Kahuna results

- Starting from previous solution, optimize with variable value-of-water
  - iteration 0: op. 4.505e+11 + inv. 2.284e+11 = total 6.789e+11 (1.8h) (very sparse investment decision)
  - iteration 1: op. 6.670+10 + inv. 5.635e+12 = total 5.702e+12 (22h) (almost completely dense investment decision)
  - iteration 2: op. 1.505e+12 + inv. 2.221e+11 = total 1.727e+12 (21h) (less dense investment decision)
  - iteration 3: op. 2.286e+11 + inv. 7.263e+11 = total 9.549e+11 (20h) (less dense investment decision)
  - iteration 4: **nope, sorry, still running**
- Already a factor of 2 better than original system (no investment)
- Using LPs in SDDP (many numerical issues), Lagrangian will be better and will be able to use way more threads (`ParallelBundleSolver`)
- Will improve over the fixed value-of-water, just not there as yet

The Big Kahuna results

The Big Kahuna results

- Roll of drums . . .

The Big Kahuna results

- Roll of drums . . . suspenseful pause . . .

The Big Kahuna results

- Roll of drums . . . suspenseful pause . . .
- JUST KIDDING: still in the design phase, forget about it (for today)
- Clearly extremely challenging problem, need all the help we can get
- But we are getting there, thanks to SMS++

Outline

Conclusions

- Want to solve ludicrously large problem? `SMS++` is here for you
- Allows exploiting multiple nested heterogeneous structure, $\approx$ the only framework designed for huge-scale (stochastic or not) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop very hard to start

Conclusions

- Want to solve ludicrously large problem? `SMS++` is here for you
- Allows exploiting multiple nested heterogeneous structure, ≈ the only framework designed for huge-scale (stochastic or not) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop very hard to start
- Hefty, possibly unrealistic, sough-after impacts:
  - 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

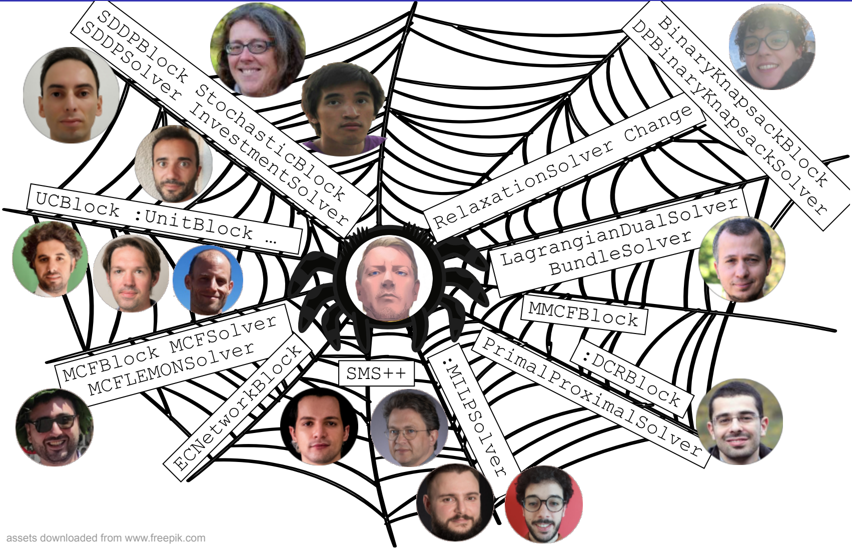
Conclusions

- Want to solve ludicrously large problem? SMS++ is here for you
- Allows exploiting multiple nested heterogeneous structure, ≈ the only framework designed for huge-scale (stochastic or not) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop very hard to start
- Hefty, possibly unrealistic, sough-after impacts:
  - 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 community-building effort as much as a software product:

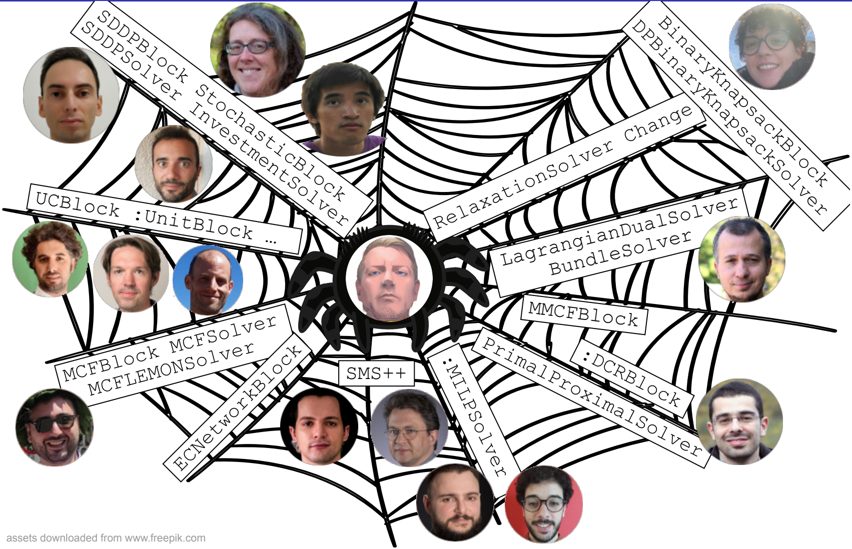
Conclusions

- Want to solve ludicrously large problem? `SMS++` is here for you
- Allows exploiting multiple nested heterogeneous structure, $\approx$ the only framework designed for huge-scale (stochastic or not) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop very hard to start
- Hefty, possibly unrealistic, sough-after impacts:
    - 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 community-building effort as much as a software product: is it working?

. . . see it for yourself

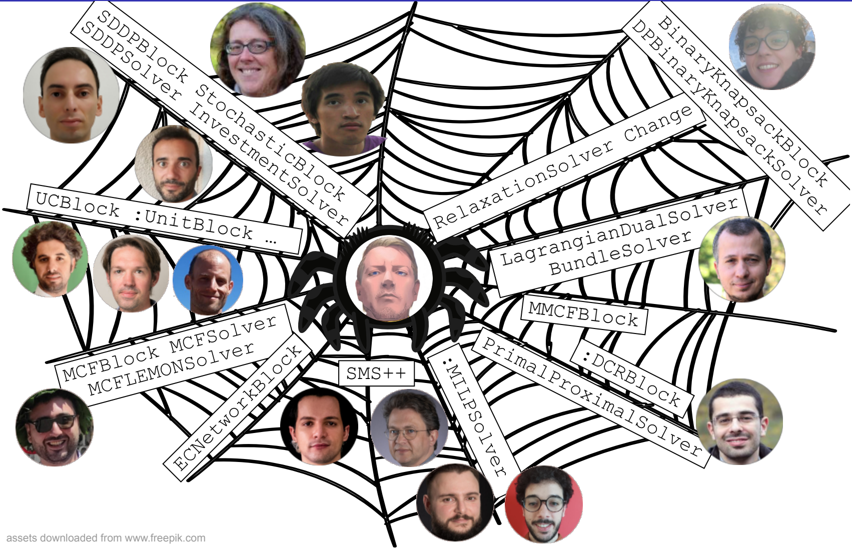


- Not yet a thriving community, but labouring to reach critical mass



- Not yet a thriving community, but labouring to reach critical mass
- Plenty of room in the Hotel California

...see it for yourself



- Not yet a thriving community, but labouring to reach critical mass
- Plenty of room in the spiderweb, be my guest