SMS++: a Structured Modelling System with Applications to Energy Optimization

Antonio Frangioni[1] Rafael Durbano Lobato[2]

[1]Dipartimento di Informatica, Università di Pisa
[2]Department of Applied Mathematics, State University of Campinas

PGMO DAYS 2018
November 21, 2018 – Paris

Outline

PGMO

- Project "Consistent Dual Signals and Optimal Primal Solutions" (2012–2018): initial implementation of SMS++ (Ph.D. Thesis[5])
- Project "Advanced Modeling Tools for Decomposition Methods to Energy Optimization Problems" (2016):
  develop general Benders' decomposition code
- Superseded by Project "Multilevel Heterogeneous Distributed Decomposition for Energy Planning with SMS++" (to start RSN):
  generic multi-level decomposition (arbitrary combination of Benders', Lagrange and whatever)

The plan4res H2020 project

- The plan4res project (www.plan4res.eu):

  > "An end-to-end planning and operation tool, composed of a set of optimization models based on an integrated modelling of the pan-European Energy System"

- Plus IT infrastructure, plus lots of data, plus 3 case studies

- An accurate depiction of long-term effects of strategic choices on the pan-European Energy System ≡

  modelling the next 30 years with 1h timescale

  and huge amounts of uncertainty over everything

- An unfeasibly large optimization problem

The plan4res H2020 project

- The plan4res project (www.plan4res.eu):

  "An end-to-end planning and operation tool, composed of a set of optimization models based on an integrated modelling of the pan-European Energy System"

- Plus IT infrastructure, plus lots of data, plus 3 case studies

- An accurate depiction of long-term effects of strategic choices on the pan-European Energy System ≡

  modelling the next 30 years with 1h timescale

  and huge amounts of uncertainty over everything

- An unfeasibly large optimization problem with lots of structure

- Schedule a set of generating units to satisfy the demand at each node of the transmission network at each time instant of the horizon
- Two versions: simulation (only costs) and operation (also schedules)
- Three natural sources of structure: unit, time, and network
- Relaxing demand constraints decomposes by unit and network: one problem per unit across all horizon, a network problem per instant
- Indeed, Lagrangian Relaxation historically[1] the go-to approach for both simulation and operations[2,3]

---

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

[2] Borghetti, F., Lacalandra, Nucci "Lagrangian Heuristics Based on Disaggregated Bundle Methods for Hydrothermal Unit Commitment", *IEEE Transactions on Power Systems*, 2003

[3] Dubost, Gonzalez, Lemaréchal "A primal-proximal heuristic applied to french unit-commitment problem" *Math. Prog.* 2005

UC Lagrangian approaches

- A lot of network structure:
  - Dynamic Programming[4] for simple single thermal units, but not for complex ones[5]
  - Min-Cost Flows[6] for simple hydro valleys, but not for complex ones[7]
  - Laplacian of graph[8] for simple network constraints, but not for complex ones[9]
  - other stuff (ROR hydro, solar/wind, small-scale storage, demand response, smart grids, ...) usually "easy"
- Efficient algorithms for simple cases
- At least some hope for complex cases (real-world operations)

---

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

5 Tavlaridis-Gyparakis "Decomposition Techniques for Large-Scale Energy Optimization Problems" *Ph.D. Thesis*, 2018

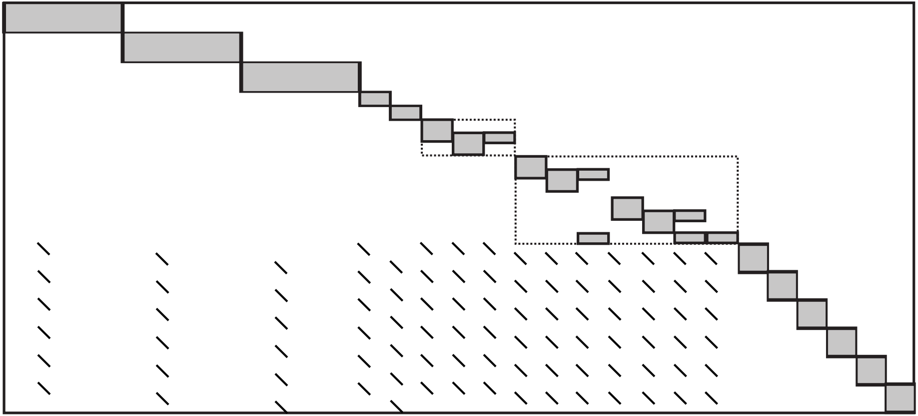
6 F., Manca "A Computational Study of Cost Reoptimization for Min Cost Flow Problems" *INFORMS JOC*, 2006

7 Sahraoui, Bendotti, D'Ambrosio "Real-world hydro-power unit-commitment [...]" *Energy*, 2017

8 F., Serra Capizzano "Spectral Analysis of (Sequences of) Graph Matrices" *SIMAX*, 2001

9 Bienstock, Chertkov, Harnett "Chance-constrained optimal power flow [...]" *SIAM Review* 2014

- A lot of network structure spread around (≈ multicommodity flow)



- Nontrivial linking constraints

Can We Deal With Such a Structure?

- Of course we can, in fact with several different approaches:
  - Lagrangian decomposition[10] and related methods[11], even in parallel[12]
  - Structured Interior-Point methods[13]
  - Structured active-set (simplex) methods[14]
  - Structured Dantzig-Wolfe decomposition[15,16]
  - ...
- Unclear which is better for the application at hand, since have to be solved many times with changing data ≡ reoptimization

---

10 F., Gallo "A Bundle type dual-ascent approach to linear multicommodity min cost flow problems" *INFORMS JOC*, 1999

11 Grigoriadis, Khachiyan "An exponential function reduction method for block angular convex programs" *Networks*, 1995

12 Cappanera, F. "Symmetric and asymmetric parallelization of a cost-decomposition algorithm [...]" *INFORMS JOC*, 2003

13 Castro "Solving difficult multicommodity problems through a specialized interior-point algorithm" *Ann. OR*, 2003

14 McBride "Progress made in solving the multicommodity flow problem" *SIOPT*, 1998

15 F., Gendron "A stabilized structured Dantzig-Wolfe decomposition method" *Math. Prog.*, 2013

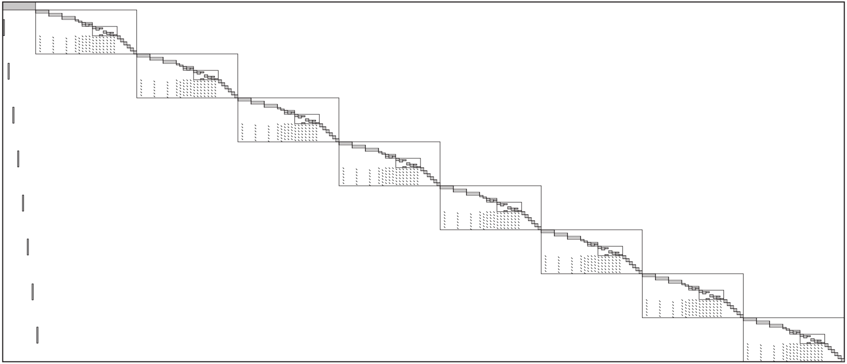
16 Mamer, McBride "A decomposition-based pricing procedure for large-scale linear programs [...]" *Man. Sci.*, 2000

Mid Level: Seasonal storage valuation

- Unit-Commitment is a short-term problem, lacks long-term strategies
- Issue: cost of water (none) / minimum reservoir volume (very low)
  $\Longrightarrow$ lot of water used early on $\Longrightarrow$ no water most of the year
- Hydro production most useful for peak shaving every day
- Computing value of water left in the reservoirs at horizon end
  $\equiv$ the value function of UC w.r.t. max water constraints
  (naturally convex if Lagrangian relaxation = convexification[17] used)
- A mid-term (1y) stochastic program: uncertain inflows, demands, ...
- Stochastic dual dynamic programming[18,19]
  with multiple EUC inside (in principle $\approx$ 365)

[17] Lemaréchal, Renaud "A geometric study of duality gaps, with applications" *Math. Prog.* 2001
[18] Pereira, Pinto "Multi-stage stochastic optimization applied to energy planning" *Math. Prog.*, 1991
[19] van-Ackooij, Warin "On conditional cuts for Stochastic Dual Dynamic Programming" arXiv:1704.06205, 2017

A Picture is Worth 1000 words



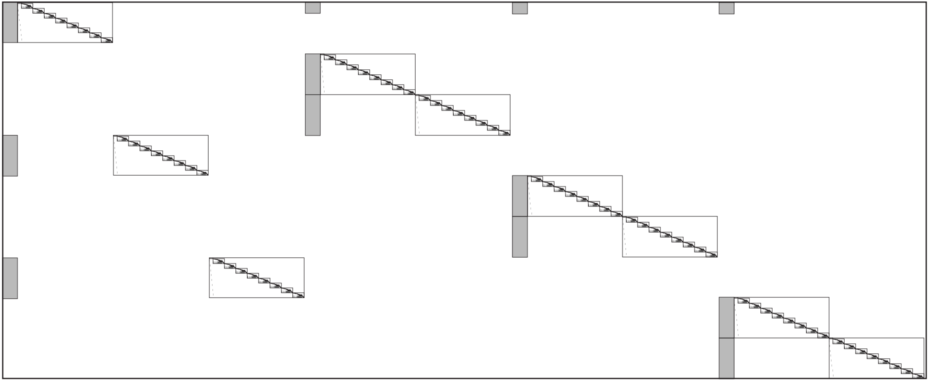
- Standard structure for (DP +) Benders' decomposition
- Whenever you do Benders' you can do Lagrange[20] and vice-versa[21]
- Very many different variants, which is best/feasible??

[20] Guignard, Kim "Lagrangean decomposition: a model yielding stronger lagrangean bounds" *Math. Prog.* 1987

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

- The energy system changes all the time, but modifications slow, extremely costly, with huge inertia
- Demand and production subject to very significant uncertainties: climate = RES production + demand, shifts in consumption patterns (EV, cryptocurrencies, . . .), new technologies (shale, LED, . . .), geo-political factors (energy security), economical factors (boost or boom), regulatory factors (EU energy market, . . .), political factors (CO_2 emission treaties, nuclear power, . . .), . . .
- Planning long-term evolution very hard, yet necessary
- 20/30 years, 2/5 years steps (multi-level recourse), many scenarios

A Picture is Worth 1000 words



- Huge size, multiple nested structure
- Still OK for either Benders or Lagrange
- Benders + DP + Benders + Lagrange + Graph or . . . ???

How Do you Solve Such a Thing?

- Modeling system: easily construct a huge, flat = unstructured matrix to be passed to a general-purpose, flat solver
- Some solvers offer one-level decomposition (Benders, CG = DW)
- Attempts at automatically recovering structure from a matrix[22], but only one level and anyway conceptually awkward
- Only one tool (that I know of) for multiple nested structure[23,24], but only solves continuous problems by Interior Point methods
- Nothing for multilevel, heterogeneous approaches (such as, but not only, decomposition), e.g., allowing specialized solvers for each block
- So far

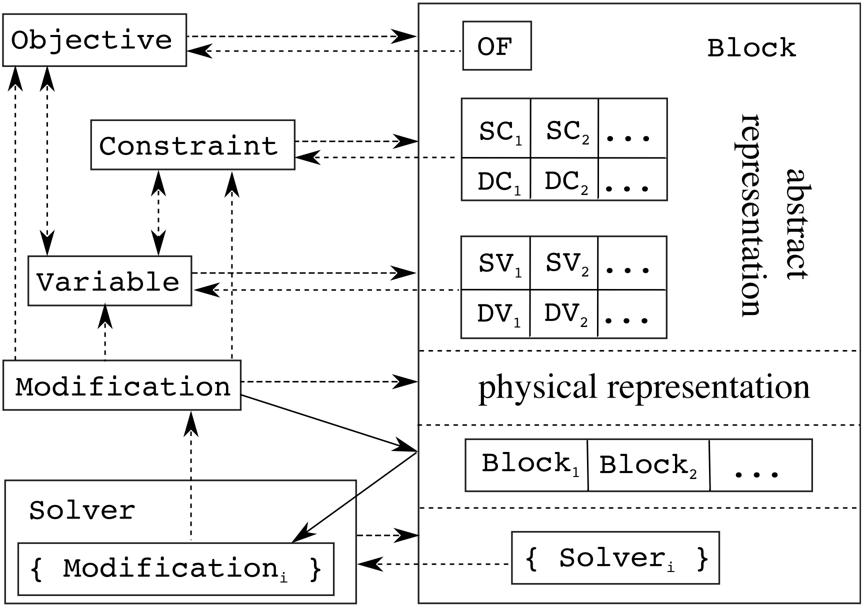
[22] Furini, Lübbecke, Traversi et. al. "Automatic Dantzig–Wolfe reformulation of mixed integer programs" *Math. Prog.* 2015
[23] Gondzio, Grothey "Exploiting Structure in Parallel Implementation of Interior Point Methods [. . .]" *Comput. Man. Sci.*, 2009
[24] Colombo et al. "A Structure-Conveying Modelling Language for Mathematical [. . .] Programming" *Mathe. Prog. Comp.*, 2009

Outline

What We Want

- A modelling language/system which:
  - 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 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++11/14/17/20`)
  - tried with Julia/JuMP, but could not handle well C++ interface

The Core SMS++


Objective
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., 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)
- Abstract representation of a Block:
  - one Objective (but possibly vector-valued)
  - 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::MCFBlock inside)

Variable

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

Constraint

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

Objective

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

Function

- `Function` only deals with (real) values
- Depends from a set of `Variable` (`:ThinVarDepInterface`)
- Must be evaluated w.r.t. the current value of the `Variable`, possibly a costly operation (`:ThinComputeInterface`)
- Approximate computation supported in a quite general way[25] (since `:ThinComputeInterface`, and that does)
- Asynchronous evaluation still not defined
- `FunctionModification[Variables]` for "easy" changes $\Longrightarrow$ reoptimization (shift, adding/removing "quasi separable" `Variable`)

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

C05Function and C15Function

- C05Function/C15Function deal with 1st/2nd order information (not necessarily continuous)
- General concept of "linearization" (gradient, convex/concave subgradient, Clarke subgradient, ...)
- Multiple linearizations produced at each evaluation (local pool)
- Global pool of linearizations for reoptimization:
  - convex combination of linearizations
  - "important linearization" (at optimality)
- C05FunctionModification[Variables/LinearizationShift] for "easy" changes $\Longrightarrow$ reoptimization (linearizations shift, some linearizations entries changing in simple ways)
- C15Function supports Hessians, unclear how much reoptimization possible/useful

- Generic concept of "something depending on a set of `Variable`"
- Specific implementation demanded to derived classes for efficiency
- "Abstract" STL-like iterator and const-iterator for access
- Other specific methods to describe/search 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 evaluate"
- Specific provisions for the fact that the computation can:
  - end in several ways (OK, error, stopped, ...) and be resumedx
  - be influenced by `int`/`double`/`std::string` parameters which can be gathered in a `ComputeConfig:Configuration` object (flexible)
- 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"
- Changes may be "explicit" (a `Modification` issued) or "implicit" (changing a `Variable` value do not trigger a `Modification`)
- Asynchronous `compute()` not done yet, TBD soon with Cray™ help: changes in this interface will do the trick everywhere

Block and Solver

- Any # of Solver attached to a Block to solve it
- :Solver for a specific :Block can use the physical representation
  $\Longrightarrow$ no need for explicit Constraint
  $\Longrightarrow$ abstract representation of Block only constructed on demand
- However, Variable are always present (interface with Solver)
- A general-purpose Solver uses the abstract representation
- Dynamic Variable/Constraint can be generated on demand (user cuts/lazy constraints/column generation)
- For a Solver attached to a Block:
  - Variable not belonging to the Block are constants
  - Constraint not belonging to the Block are ignored

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

Solver

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

Block and Modification

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

Modification

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

Solution

- Block produces Solution object, possibly using its sub-Blocks'
- Solution can read() its own Block and write() itself back
- Solution is Block-specific rather than Solver-specific
- Solution may save dual information
- Solution may save only a specific subset of primal/dual information
- Linear combination of Solution supported $\Longrightarrow$ "less general" Solution may (automatically) convert in "more general" ones
- Like Block, Solution are tree-structured complex objects

- Block a tree-structured complex object $\Longrightarrow$
  Configuration for them a (possibly) tree-structured complex object
- But also SimpleConfiguration<T>:Configuration
  (T an int, a double, a std::pair<>, ...)
- BlockConfiguration:Configuration sets (recursively):
  - which dynamic Variable/Constraint are generated, how (Solver, time limit, parameters ...)
  - which Solution is produced (what is saved)
  - a bunch of other Block parameters
- BlockSolverConfiguration:Configuration sets (recursively) which Solver are attached to the Block and their ComputeConfiguration:Configuration
- Both can be set (recursively) at once

R^3Block

- Often reformulation crucial, but also relaxation or restriction: `get_R3_Block()` produces one, possibly using sub-`Block`s'
- Obvious special case: copy (clone) should always work
- Available R^3Blocks `:Block`-specific, a `:Configuration` needed
- R^3Block completely independent (new `Variable`/`Constraint`), useful for algorithmic purposes (branch, fix, solve, . . .)
- Solution of R^3Block useful to `Solver` for original `Block`: `map_back_solution()` (best effort in case of dynamic `Variable`)
- Sometimes keeping R^3Block in sync with original necessary: `map_forward_Modification()`, task of original `Block`
- `map_forward_solution()` and `map_back_Modification()` useful, e.g., dynamic generation of `Variable`/`Constraint` in the R^3Block
- `:Block` is in charge of all this, thus decides what it supports

Other useful stuff

- un_any_thing() template functions/macros to extract (std::vector or boost::multi_array of) (std::list of) Variable/Constraint out of a boost_any and work on that
- All tree-structured complex objects (Block, Configuration, Solution) have an (almost) completely automatic factory
- All tree-structured complex objects (...) have methods to serialize/deserialize themselves to netCDF files
- All objects have ">>" std::stream operator, some (Block) also have "<<"

Closer to the ground

- `ColVariable:Variable`: "value = one single real" (possibly $\in \mathbb{Z}$)
- `RowConstraint:Constraint`: "$l \leq$ a real $\leq u$" $\Longrightarrow$ has dual variable (single real) attached to it
- `OneVarConstraint:RowConstraint`: "a real" = a single `ColVariable` $\equiv$ bound constraints
- `FRowConstraint:RowConstraint`: "a real" given by a `Function`
- `FRealObjective:RealObjective`: "value" given by a `Function`
- `LinearFunction:Function`: a linear form in `ColVariable`
- `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
- `FakeSolver:Solver`: just stashes away all `Modification`

Outline

S*MILPBlock and MILPSolver

- SimpleMILPBlock:Block: an un-structured set of FRowConstraint and one FRealObjective with only LinearFunction on an un-structured set of ColVariable, possibly with attached OneVarConstraint but no sub-Block

- StructuredMILPBlock:SimpleMILPBlock: all sub-Block can be SimpleMILPBlock (hence also StructuredMILPBlock), generic linking constraints are defined among the variables of the father Block and of the sub-Block

TBD MILPSolver:Solver: passes to Cplex any Block that only has any # of groups of ColVariable and FRowConstraint, and a FRealObjective, all with LinearFunction only

TBD MILPSolver to be transformed in "generic" MILP solver interface with a sub-class for SCIP

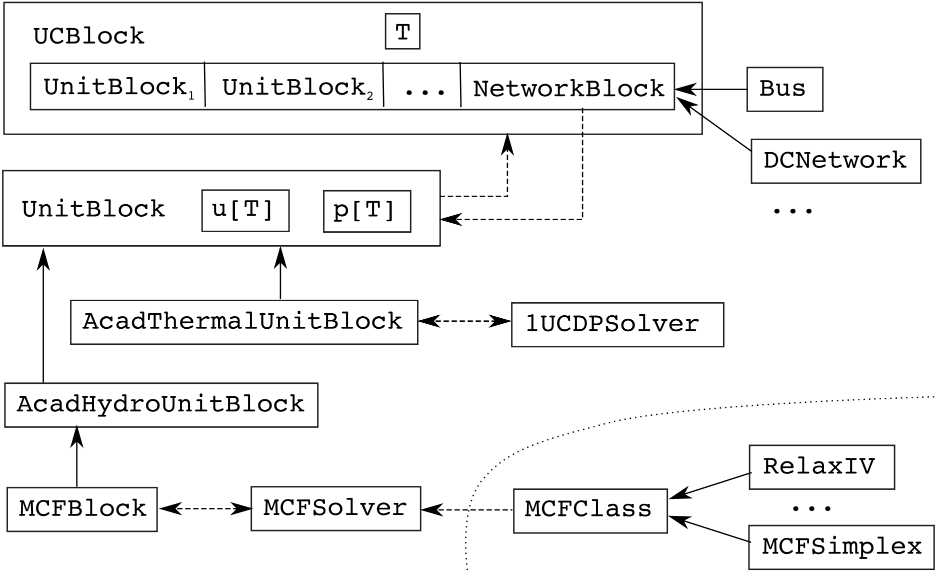
MCFBlock and MCFSolver

- MCFBlock:Block: a Min-Cost Flow Problem
- MCFSolver:Solver: solves a MCFBlock forwarding the MCFClass interface (www.di.unipi.it/optimize/Software/MCF.html) and its existing solvers (<MCFClass>)
- First complete implementation of a Block/Solver pair, with almost all mechanisms (physical/abstract Modification, R^3Block, ...) save for dynamic stuff and sub-Block
- Everything seems to fit, but testing still underway

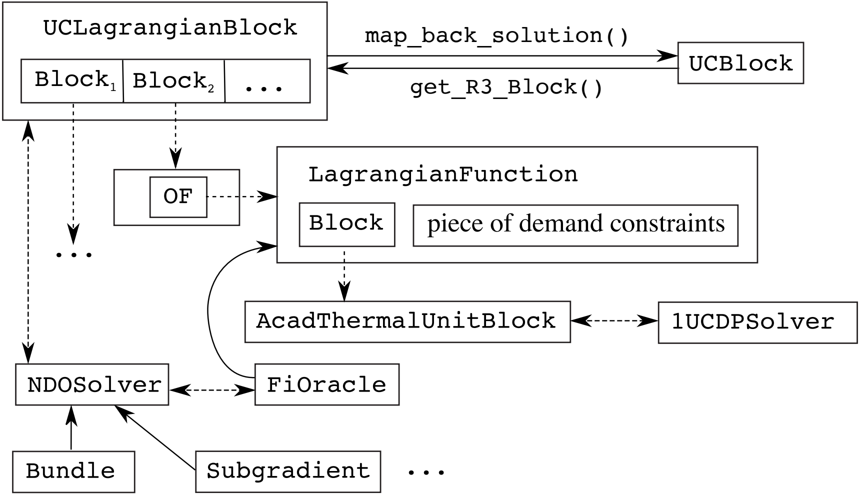
LagrangianFunction [TBD]

- LagrangianFunction:C05Function has one isolated Block + set of (say) LinearFunction to define Lagrangian term
- evaluate() = Block.get_registered_solvers()[i].solve(): asynchronous Solver $\Longrightarrow$ asynchronous Function
- Solution extracted from Block $\equiv$ linearization
- Solver provides local pool
- LagrangianFunction handles global pool
- All changes lead to reoptimization-friendly C05FModification
- BendersFunction should be quite similar

UCBlock and Companion Classes [TBD]



UCLagrangianBlock [TBD]



- Independent from details of units/network
- Multi-level decomposition now (perhaps) possible

Outline

A Lot of Work, Then Maybe Conclusions

- Current pre-beta version sitting tight on GitLab

  `gitlab.com/frangio68/sms_plus_plus_project`

  Private repository, but any interested onlooker/contributor just ask
- Two quite good, (2+1)-years, 2^{15} €/ year, post-doc positions open

  `https://www.unipi.it/ateneo/bandi/assegni/asse2018/inf/28nov2018`

  Deadline 28/11, thanks for helping disseminate
- About time, too, because a lot of work still ahead of us
- True large-scale application still to come, `Solver` to be written
- Asynchronous still to be figured out (but very relevant), good Cray™ folks will lend a helping hand here
- Clearly not for the faint of heart . . .

A Lot of Work, Then Maybe Conclusions

- Current pre-beta version sitting tight on GitLab

  gitlab.com/frangio68/sms_plus_plus_project

  Private repository, but any interested onlooker/contributor just ask
- Two quite good, (2+1)-years, 2^{15} €/ year, post-doc positions open

  https://www.unipi.it/ateneo/bandi/assegni/asse2018/inf/28nov2018

  Deadline 28/11, thanks for helping disseminate
- About time, too, because a lot of work still ahead of us
- True large-scale application still to come, `Solver` to be written
- Asynchronous still to be figured out (but very relevant), good Cray™ folks will lend a helping hand here
- Clearly not for the faint of heart . . .
  but when it'll work it will be useful in many applications

We are trying. Anyone cares to join?

Acknowledgements

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