Antonio Frangioni
Dipartimento di Informatica, Università di Pisa
plan4res webinar – May 20, 2021

Outline

1. SMS++: design goals
2. SMS++: basic components
3. SMS++: existing `Block` and `Solver`
4. SMS++: (some of) the missing pieces
5. Conclusions

Outline

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

Open source (LGPL3)

Public as of February 8, 2021, but some 8+ years in the making

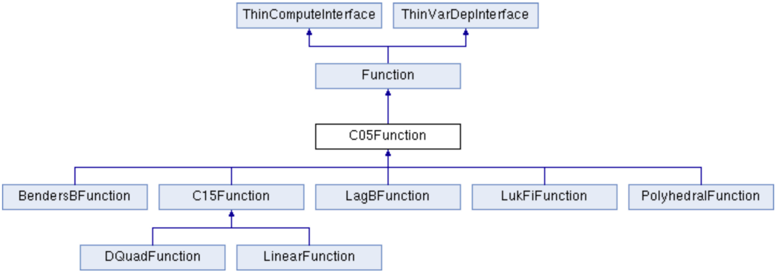
- A core set of C++-17 classes implementing a modelling system that:
  - explicitly supports the notion of Block ≡ nested structure
  - separately provides "semantic" information from "syntactic" details (list of constraints/variables ≡ one specific formulation among many)
  - allows exploiting specialised 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

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

Outline

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

- Abstract concepts, thought to be extended (a matrix, a function, any algebraic constraint, a matrix constraint, a PDE constraint, bilevel program, vector-valued objective, ...)
- Know which `Block` they belongs to
- Very basic operations (fixed/unfixed, relaxed/enforced, minimised/maximised, ...)
- Fundamental design decision: "names" are memory addresses $\Longrightarrow$ copying makes a different one
- `Modification` issued each time everything changes
- Currently a few "very basic" implementations, such as
  - `ColVariable:Variable`: "value = one single real" (possibly $\in \mathbb{Z}$)
  - `RowConstraint:Constraint`: "$l \leq$ a real $\leq u$" (has dual variable)
  - `FRowConstraint:RowConstraint`/`FRealObjective:RealObjective`: "a real" given by a `Function`



- Depends from a set of `Variable` , must be evaluated
- Evaluation possibly costly, approximate computation supported
- `C05Function`/`C15Function` deal with 1st/2nd order information, general concept of "linearization", local/global pool for reoptimization
- Arbitrary hierarchy of `:Function` possible/envisioned
- Currently `LinearFunction`, `DQuadFunction`, a few others

- Solver = general interface between a Block and algorithms solving it
  - any # of multiple optimal and sub-optimal solutions produced on demand, certificates of unboundedness/unfeasiblility (rays)
  - time/resource limits for solutions, but restarts (reoptimization)
  - flexible user interface via extendable events
  - lazily reacts to changes in the data of the Block via Modification
- Any # of Solver attached to a Block and to each sub-Block (recurs.)
- :Solver for a specific :Block can use the physical representation
- A general-purpose Solver uses the abstract representation
- However, Variable are always present to interface with Solver
- Dynamic Variable/Constraint can be generated on demand (user cuts/lazy constraints/column generation)
- CDASolver:Solver is "Convex Duality Aware": bounds are associated to dual solutions (possibly, multiple)

- Most Block components can change (but not all)
- Any change is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification object
- However, two different kinds of Modification (what changes):
  - physical Modification, only specialized Solver concerned
  - abstract Modification, only Solver using it concerned

  (specialized Solver disregard abstract Modification and vice-versa)
- Abstract Modification used to keep both representations in sync
  $\Longrightarrow$ a single change may trigger more than one 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)
- GroupModification to (recursively) pack many Modification together
  $\Longrightarrow$ different "channels" in Block

Support to (coarse-grained) Parallel Computation

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

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

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

Main Existing :Block

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

Main "Basic" :Solver

- MCFSolver: templated p.o.c. wrapper to MCFClass[1] for MCFBlock
- DPSolver for ThermalUnitBlock (still needs serious work)
- MILPSolver: constructs matrix-based representation of any "LP" Block + CPXMILPSolver:MILPSolver and SCIPMILPSolver:MILPSolver wrappers for Cplex and SCIP (to be improved)
- BundleSolver:CDASolver: SMS++-native version of[2] (still shares some code, dependency to be removed), optimizes any (sum of) C05Function, several (but not all) state-of-the-art tricks
- SDDPSolver: wrapper for SDDP solver StOpt[3] using StochasticBlock, BendersBFunction and PolyhedralFunction
- SDDPGreedySolver: greedy forward simulator for SDDPBlock
- Regularly new entries (latest BinaryKnapsackDPSolver)

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

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

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

Our Masterpiece: LagrangianDualSolver

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

Outline

- A relaxation-agnostic Branch-and-X `Solver`
- Many other forms of (among many other things):
  - `Variable` (`Vector/MatrixVariable`, `FunctionVariable`, ...)
  - `Constraint` (`SOCConstraint`, `SDPConstraint`, `PDEConstraint`, `BilevelConstraint`, `EquilibriumConstraint`, ...)
  - `Objective` (`RealVectorObjective`, ...)
  - `Function` (`AlgebraicFunction`, ...)
- Better handling of many things (groups of stuff, `Modification`, ...)
- Interfaces with many other general-purpose solvers (`GuRoBi`, `OSISolverInterface`, `Couenne`, OR-tools `CP-SAT Solver`, ...)
- Many many many more `:Block` and their specialised `:Solver`
- Translation layers from real modelling languages (AMPL, JuMP, ...)
- In a word: users/mindshare – chicken-and-egg problem

Outline

- SMS++ is there, actively developed, lasting legacy of plan4res
- Currently mostly useful for “extreme” use cases
- Will be more useful after having attracted mindshare:
  - improve collaboration and code reuse, reduce huge code waste
  - significantly increase the addressable market of decomposition
  - a much-needed step towards higher uptake of parallel methods
  - the missing marketplace for specialised solution methods
  - a step towards a reformulation-aware modelling system[4]
- A system to help developers of advanced optimization algorithms and users of highly demanding models to meet
- Whether you are a developer or a user, give it a look

[4] F., Perez Sanchez “Transforming Mathematical Models Using Declarative Reformulation Rules” *LNCS*, 2011

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