Multiple Nested Structures: the Curse (or Blessing?) of Applied Mathematics

Antonio Frangioni[1]

[1]Dipartimento di Informatica, Università di Pisa

Calcolo Scientifico e Modelli Matematici:
Alla Ricerca delle Cose Nascoste
Attraverso le Cose Manifeste 2.0
17 Maggio 2018 – Como

Outline

Structure in Optimization Problems

- General optimization problem $(P) \;\; \min \{ \, c(x) \;:\; x \in X \, \}$: clearly unsolvable if $c(\cdot)$ and X are "any" function/set
- To do anything one needs assumptions on the structure of $c(\cdot)/X$
- Many different cases, most of them hard
- Let's take it easy: strong structure $\equiv$ easy problem:

  Linear Program $(P) \;\; \min \{ \, cx \;:\; Ax = b \;,\; 0 \leq x \leq u \, \}$

  $A \in \mathbb{R}^{n \times m}$ (sparse), $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, $u \in \mathbb{R}^m$, $m > n$
- Structure $\Longrightarrow$ useful properties: dual problem to (P)

  $(D) \;\; \max \{ \, yb - wu \;:\; yA + z - w = c \;,\; z \geq 0 \;,\; w \geq 0 \, \}$

  fundamental tool for solving (P)

Solving LPs is (Structured, non)Linear Algebra

- Karush-Kuhn-Tucker optimality conditions ($diag(V) = v$, $e =$ all 1s)

$$(KKT) \quad \begin{cases} Ax = b \;,\; x + s = u \;,\; yA + z - w = c & \text{(linear)} \\ XZe = 0 \;,\; SWe = 0 & \text{(nonlinear)} \\ [\, x \,,\, s \,,\, z \,,\, w \,] \geq 0 & \text{(inequalities)} \end{cases}$$

- Interior Point methods for LP: "slacken + linearize":

  i) $\mu > 0$, $(KKT_\mu) \equiv (KKT)$ except $XZe = \mu e \;,\; SWe = \mu e$
  $\Longrightarrow$ $(2m\mu)$-optimal solution

  ii) feasible $[\, \bar{x} \,,\, \bar{s} \,,\, \bar{z} \,,\, \bar{w} \,] > 0$, $v = \bar{v} + \Delta v$ (stepsize ensures $v > 0$) $\Longrightarrow$

$$\begin{cases} A\Delta x = 0 \;,\; \Delta x + \Delta s = 0 \;,\; \Delta yA + \Delta z - \Delta w = 0 \\ \bar{X}\bar{Z}e + \bar{X}\Delta z + \bar{Z}\Delta x = \mu e \;,\; \bar{S}\bar{W}e + \bar{S}\Delta w + \bar{W}\Delta s = \mu e \end{cases}$$

  ignore second-order terms $\equiv$ Newton's method for nonlinear equations

- $[\, \bar{x} \,,\, \bar{s} \,,\, \bar{z} \,,\, \bar{w} \,]$ satisfies (KKT_μ): one iteration, $\mu \searrow$ (fast), repeat
- Many improvements (infeasible method, predictor corrector, . . .)

Solving LPs is Structured Linear Algebra

- Boils down to Reduced KKT or Normal equations ($\Theta > 0$ diagonal)

$$\begin{bmatrix} -\Theta & A^T \\ A & 0 \end{bmatrix} \qquad\qquad A\Theta A^T$$

$m + n \times m + n$, sparse $\qquad$ $n \times n$, a lot less sparse

- Special case of Saddle-Point system, lots of applications (physics, engineering, economy, computer science, ...), very active research[1]

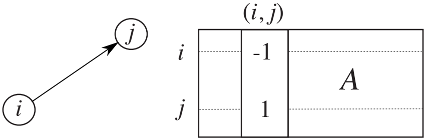
- Specific twists in the LP case:
  - large size: $m \approx 10^{6+}$, $n \approx 10^{5+}$ ...
  - must be solved many times, but rather inexactly (at the first iterations)
  - fixed nonzero structure (A) and variable data (Θ)
  - special evolution of data over the iterations
  - no discretization, no underlying smooth operator

- Ultimate performances require assumptions on (structure of) A

[1] Benzi, Golub, Liesen "Numerical solution of saddle point problems" *Acta Numerica* 14, 1–137, 2005

Going Deeper: More Assumptions ($\equiv$ Structure)

- A = *node-arc incidence matrix* of directed graph G



$\equiv$ min-cost flow problem

$\equiv$ $A \Theta A^T$ weighted Laplacian of G (an M-matrix)

- Can tell a lot on the system by looking at the graph[2,3]
- Can do a lot about the system by working on the graph:
  - preconditioners are (chordal) sub-graphs, can be obtained by efficient graph algorithms (Kruskal[4], Prim[5], . . .)
  - projection in algebraic multigrid is merging nodes[6]
  - projection and preconditioning is a unique graph-based process[7]

---

2 Cvetković, Doob, Sachs "Spectra of graphs", 1980 — Brouwer, Haemers "Spectra of Graphs", 2012

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

4 F., Gentile "New Preconditioners for KKT Systems of Network Flow Problems" SIOPT, 2004

5 F., Gentile "Prim-based Support-Graph preconditioners for Min-Cost Flow Problems" *CO&A*, 2006

6 Dell'Acqua, F., Serra Capizzano "Computational Evaluation of Multi-Iterative Approaches [. . .]" *CALCOLO*, 2015

7 Dell'Acqua, F., Serra Capizzano "Accelerated Multigrid for Graph Laplacian Operators" *Appl. Math. & Comp.*, 2015

Lots of Fun for Lots of Different People

- These systems have been approached by many different angles:
  - graph theory
  - computer science
  - numerical linear algebra
  - optimization
  - physics . . .
- Lots of ingenuity, theoretical results, implementations
- Applied mathematics at its best: focus on one structure with relevant applications, drill it down until it cries

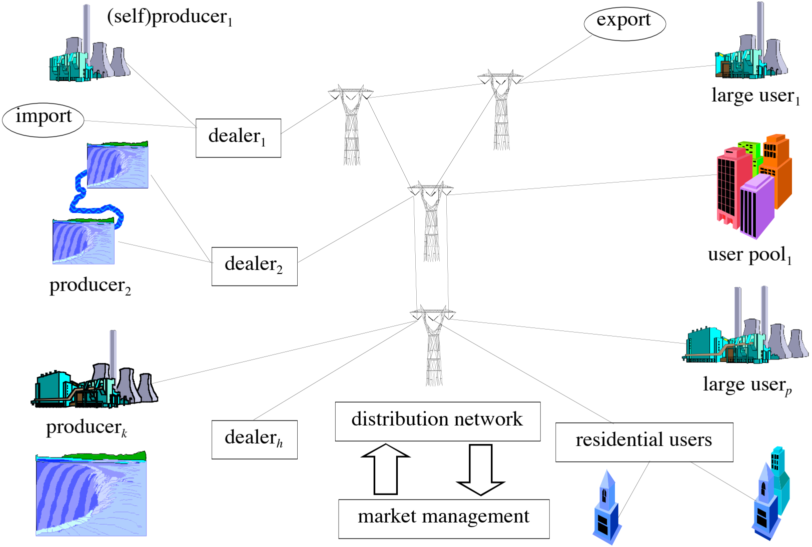
Lots of Fun for Lots of Different People

- These systems have been approached by many different angles:
  - graph theory
  - computer science
  - numerical linear algebra
  - optimization
  - physics ...
- Lots of ingenuity, theoretical results, implementations
- Applied mathematics at its best: focus on one structure with relevant applications, drill it down until it cries
- Is this always enough?

Outline

Motivation: The Unit Commitment (UC) problem

- Schedule a set of generating units $\mathcal{U}$ over a set of time instants $\mathcal{T}$ to satisfy the (forecasted) demand d_t at each $t \in \mathcal{T}$



The Unit Commitment problem

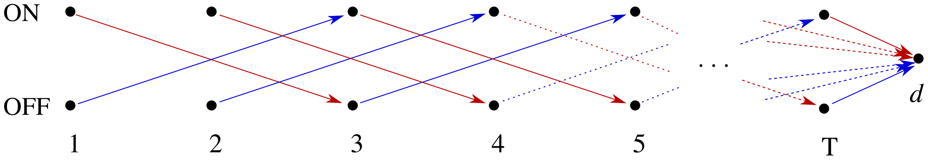
- Gazzillions €€€ / $$$, enormous amount of research[8,9]
- What has it to do with networks? More than it would seem
- Different types of production units, different constraints:
  - Thermal (comprised nuclear): min/max production, min up/down time, ramp rates on production increase/decrease, start-up 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)

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

9 The plan4res project: https://www.plan4res.eu/

Thermal Units

- Again, what did this have to do with graphs, please?
- Specialized DP algorithms for thermal single-Unit Commitment[10]
  $\equiv$ shortest path on appropriate acyclic graph

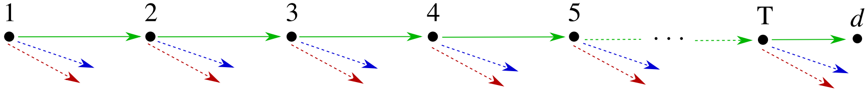


- Not too many nodes $2(T = |\mathcal{T}|)$, but rather dense: $O(T^2)$ arcs
- $(\,(\,t\,,\mathrm{ON}\,)\,,(\,\tau\,,\mathrm{OFF}\,)\,) \equiv$ startup at t and shutdown at $\tau > t$ . . .
- Costs require another nested DP per arc, $O(T^3)$ overall
- Hence, (strong but large) formulation as a flow problem[11]

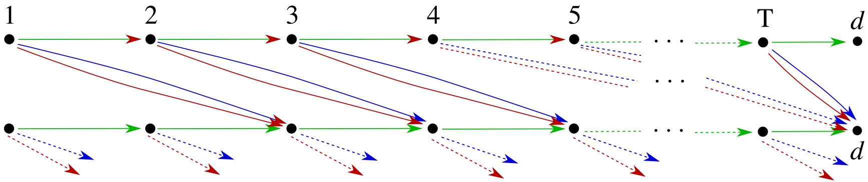
[10] F., Gentile "Solving Nonlinear Single-Unit Commitment Problems with Ramping Constraints" *OR*, 2006

[11] F., Gentile "New MIP Formulations for the Single-Unit Commitment Problems with Ramping Constraints" *IASI 15-06*, 2015

- Water flowing over time is a flow problem (surprise!)



- Quite skinny graph, $O(T)$ nodes and arcs, too
- Turbining/spilling arcs produce/not energy (max reservoir capacity)
- However, hydro units are often whole interconnected hydro valleys



- All in all (without pumping) still a flow problem, on a structured graph (composition of lines with a reverse tree)

The Network

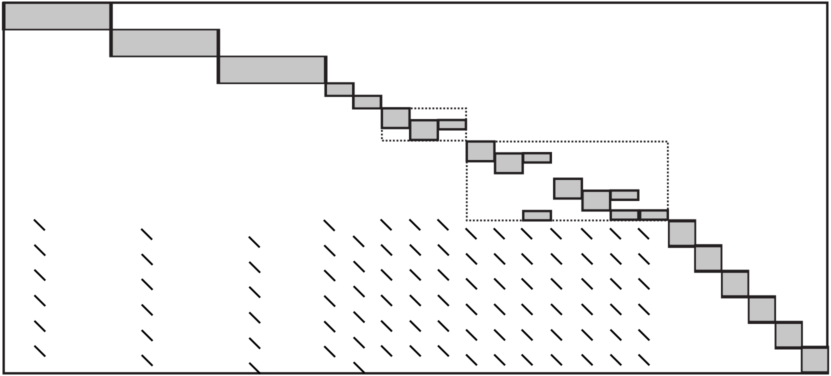
- The transmission/distribution network is a graph (surprise!) nodes are zones/buses, arcs are links (bi-directed)
- Kirchhoff's current law: $Af = n$ ($f =$ flows, $n =$ net injection)
- Kirchhoff's voltage law + Ohm's law for AC current $\Longrightarrow f = \gamma^T A^T \theta$ ($\theta =$ voltage angles, $\gamma =$ arc susceptances $= 1/$ impedence)
- AC $\Longrightarrow$ currents and voltages are periodic $\equiv$ complex numbers
- DC approximation: $|\,\theta_i - \theta_j\,| \ll 1$ $(i,j) \in A$ (small phase differences between neighbours) $\Longrightarrow$ can linearize the trigonometric functions
- $A\Gamma A^T\theta = n$ (Laplacian!) $+$ $\underline{f} \leq \gamma^T A^T \theta \leq \bar{f}$ (capacity)
- Fixing one reference voltage $A\Gamma A^T$ nonsingular:

$$\underline{f} \leq \gamma^T A^T (\,A\Gamma A^T\,)^{-1} n \leq \bar{f}$$

- True AC version nonlinear nonconvex, rater hard ...

Putting it all together

- Not a single flow but a multicommodity flow (of sorts)



- Many blocks, either A or $A\Gamma A^T$, but of rather different shape and size
- Nontrivial linking constraints

Can We Deal With Such a Structure?

- Of course we can, in fact with several different approaches:
  - Lagrangian decomposition[12] and related methods[13], even in parallel[14]
  - Structured Interior-Point methods[15]
  - Structured active-set (simplex) methods[16]
  - Structured Dantzig-Wolfe decomposition[17,18]
  - ...
- Most can exploit the "inner" graph structure of (the many) A(s)
- Significantly more complex: two-level approaches ($\equiv$ more fun)

---

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

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

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

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

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

17 F., Gendron "A stabilized structured dantzig-wolfe decomposition method" *Math. Prog.*, 2013

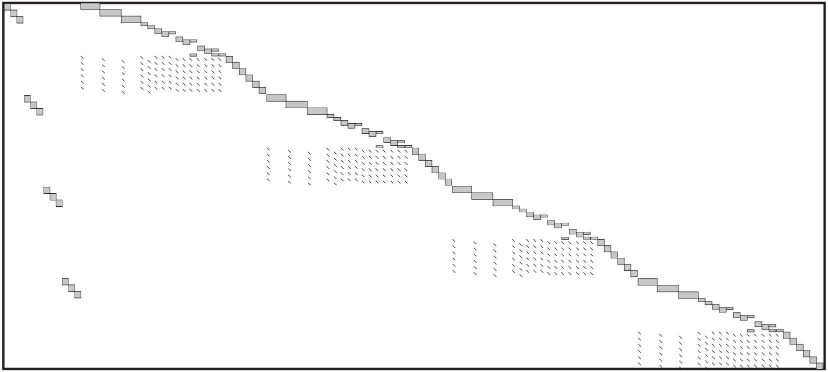
18 Mamer, McBride "A decomposition-based pricing procedure for large-scale linear programs [...]" *Man. Sci.*, 2000

Is All Well in Structure Land, Then?

- Maybe if this were the end, but it is just the beginning
- Data is uncertain: demand, wind/solar production, units/network state . . . which cannot be ignored (increased RES penetration . . .)
- Unit commitment is decided in advance (here-and-now) but actual dispatch can be changed in real time (recourse)
- Many methods to represent uncertainty: Stochastic Optimization[19], Robust Optimization, Chance-Constrained Optimization, hybrid[20]
- Simplest approach scenario-based: each $\approx$ a full UC
  $\Longrightarrow$ yet another two-level structure
- Cons: size increases of a factor # scenarios (which should be large)

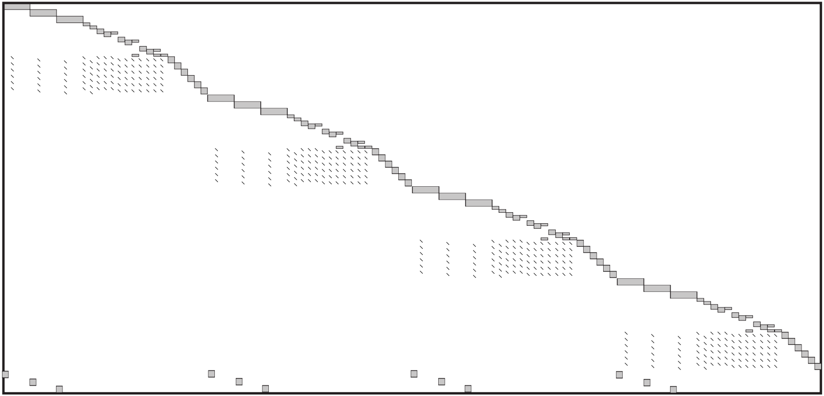
[19] Scuzziato, Finardi, F. "Comparing Spatial and Scenario Decomposition for Stochastic [. . .]" *IEEE Trans. Sust. En.*, 2018
[20] van Ackooij, F., de Oliveira "Inexact Stabilized Benders' Decomposition Approaches, with Application [. . .]" *CO&A*, 2016

Scenario-based Structure



- Perfect structure for Benders' decomposition
- Benders' decomposition with Lagrangian decomposition inside[21]
- . . . with (different) graph structure(s) inside

[21] van Ackooij, Malick "Decomposition algorithm for large-scale two-stage unit-commitment" *Ann. OR*, 2016



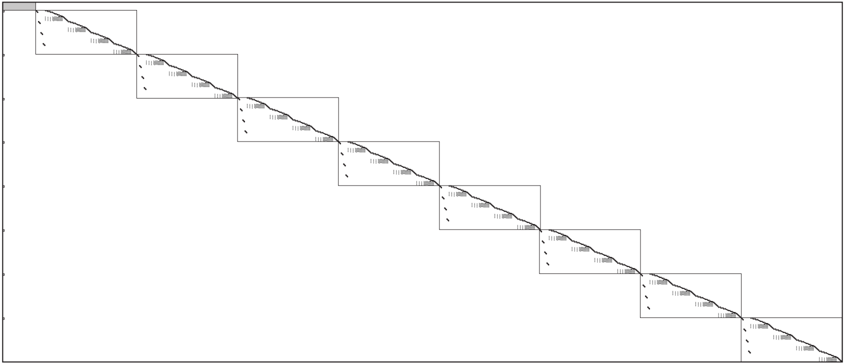
- Or was it the perfect structure for Lagrangian decomposition?
- Lagrangian decomposition with Lagrangian decomposition inside ...
- Which is better? Very hard to say beforehand[19]

OK, This is Two-Level Decomposition, Then?

- 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 $\Longrightarrow$ no water most of the year
- Hydro production most useful for peak shaving every day
- Computing value of water left in the reservoirs at T
  $\equiv$ solving a parametric (uncertain) UC problem
  for each (significant) day of the year
- Can approximate it by dual variables/Lagrangian multipliers of minimum reservoir volume constraints
- Better a piecewise linear representation (cutting-plane model)
- Then, stochastic dual dynamic programming[22] (another graph)

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

Complete Tactical Problem

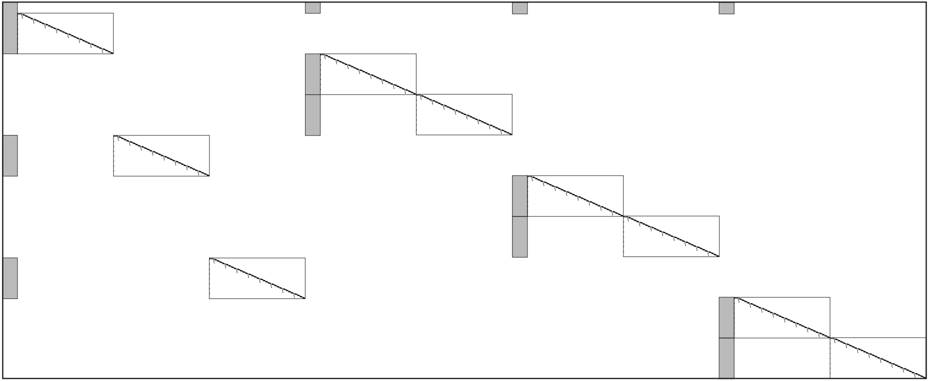


- This is not really how you'd do that (integer variables)
- Still OK for Benders-like decomposition
- Benders + Benders + Lagrange + Graph or
  Benders + Lagrange + Lagrange + Graph or
  Lagrange + Benders + Lagrange + Graph or ...

OK, But This Surely is The End, Right?

- 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

Complete Strategic Problem



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

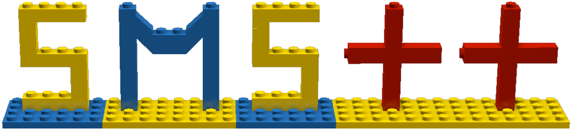
How Do you Solve Such a Thing?

- 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[23], but only one level and anyway conceptually awkward
- Only one tool (that I know of) for multiple nested structure[24,25], 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

[23] Gamrath, Lübbecke "Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs" *LNCS*, 2010
[24] Gondzio, Grothey "Exploiting Structure in Parallel Implementation of Interior Point Methods [. . .]" *Comput. Man. Sci.*, 2009
[25] Colombo et al. "A Structure-Conveying Modelling Language for Mathematical [. . .] Programming" *Mathe. Prog. Comp.*, 2009

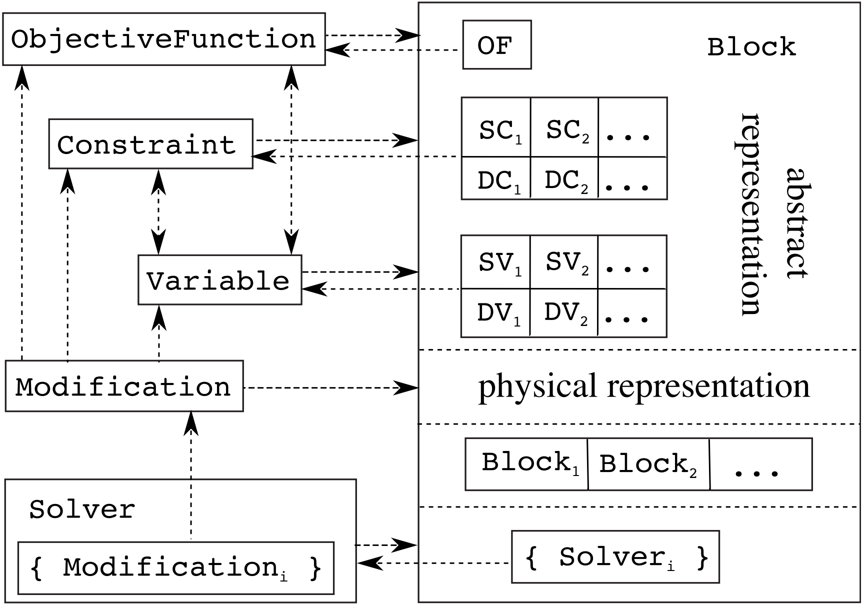
Outline

What We Want



- A modelling language/system which:
  - explicitly supports the notion of block $\equiv$ nested structure
  - separately provides "semantic" information from "syntactic" details (list of constraints/variables)
  - allows exploiting specialised solvers on blocks with specific structure
  - caters all needs of complex methods: dynamic generation of constraints/variables, modifications in the data, reoptimization
- C++ library: set of "core" classes, easily extendable
- Why C++? A number of reasons:
  - all serious solvers are written in C/C++
  - we all love it (especially `C++11/14`)
  - tried with Julia/JuMP, but could not handle well C++ interface

The Core SMS++


ObjectiveFunction
OF
Block
Constraint
SC1 SC2 ...
DC1 DC2 ...
Variable
SV1 SV2 ...
DV1 DV2 ...
abstract representation
Modification
physical representation
Block1 Block2 ...
Solver
{ Modificationi }
{ Solveri }

Block

- Block = abstract class representing the general concept of "a part of a mathematical model with a well-understood identity"
- Each `Block::` a model with specific structure (e.g., `Block::BinKnapsackBlock` = a 0/1 knapsack problem)
- Physical representation of a `Block`: whatever data structure is required to describe the instance (e.g., a, b, c)
- Abstract representation of a `Block`:
  - one (for now) `ObjectiveFunction`
  - any # of groups of (pointers) to (static) `Variable`
  - any # of groups of `std::list` of (pointers) to (dynamic) `Variable`
  - any # of groups of (pointers) to (static) `Constraint`
  - any # of groups of `std::list` of (pointers) to (dynamic) `Constraint`

  groups of `Variable`/`Constraint` can be single (`std::list`) or `std::vector` (...) or `boost::multi_array` thanks to `boost::any`
- Any # of sub-`Block`s (recursively), possibly of specific type (e.g., `Block::MMCFBlock` can have k `Block::MCFBlock`s inside)

Variable

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

Constraint

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

ObjectiveFunction

- Abstract concept, perhaps to be extended (vector-valued ...)
- Either minimized or maximized
- Keeps the set of `Variables` it depends from
- Can be evaluated w.r.t. the current value of the `Variables` (but its value depends on the specific form)
- `ObjectiveFunction::RealObjectiveFunction` implements "value is an extended real"
- Knows which `Block` it belongs to
- Same fundamental design decision ...
  (but there is no such thing as a dynamic `ObjectiveFunction`)
- `Modification::OFModification` (change verse)

Block and Solver

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

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

Solver

- Solver = interface between a Block and algorithms solving it
- Each Solver attached to a single Block, from which it picks all the data, but any # of Solvers can be attached to the same Block
- Solutions are written directly into the Variables of the Block
- Individual Solvers can be attached to sub-Blocks of a Block
- Tries to cater for all the important needs:
  - optimal and sub-optimal solutions, provably unbounded/unfeasible
  - time/resource limits for solutions, but restarts (reoptimization)
  - any # of multiple solutions produced on demand
  - lazily reacts to changes in the data of the Block via Modifications
- Heavily slanted towards RealObjectiveFunction (optimality guarantees being upper and lower bounds)
- Derived CDASolver is "Convex Duality Aware": bounds are associated to dual solutions (possibly, multiple)
- Something relevant may be missing, asynchronous calls not clear yet

Block and Modification

- Most Block components can change, but not all:
  - set of sub-Blocks
  - number and shape of groups of Variables/Constraints
- Any change is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification object
- anyone_there() $\equiv \exists$ interested Solver (Modification needed)
- However, two different kinds of Modification (what changes):
  - physical Modification, only specialized Solvers concerned
  - abstract Modification, only Solvers using it concerned
- Abstract Modification on Variable/Constraint must always be issued, even if no Solver, to keep both representations in sync
- A single change may trigger more than one Modification
- A Solver will disregard a Modification it does not understand (there must always be another one it understands)
- A Block may refuse to support some changes (explicitly declaring it)

Modification

- Almost empty base class, then everything has its own derived ones
- Each change to `Block`/`Variable`/`Constraint` ... produces a `Modification`, and a smart pointer is passed to the `Block`
- The `Block` funnels it to the interested `Solvers` (above, if any)
- Heavy stuff can be attached to a `Modification` (e.g., added/deleted dynamic `Variable`/`Constraints`)
- Each `Solver` has the responsibility of cleaning up its list of `Modifications` (smart pointers → memory will finally be released)
- `Modifications` processed in the arrival order to ensure consistency
- `Solvers` are supposed to reoptimize to improve efficiency, which is easier if you can see all list of changes at once (lazy update)
- A `Solver` may optimize the changes (`Modifications` may cancel each outer out ...), but its responsibility

Solution and Configuration

- Block produces one Solution, possibly using its sub-Blocks'
- A Solution can read() its own Block and write() itself back
- Solution is Block-specific rather than Solver-specific
- Solution may save dual information
- Solution may save only a specific subset of the primal/dual solution
- Block, Solution are tree-structured complex objects
- Configuration for them a (possibly) tree-structured complex object but also Configuration::SimpleConfiguration (an int)
- Configuration::BlockConfiguration sets (recursively):
  - which dynamic Variable/Constraints are generated, how (Solver, time limit ...)
  - which Solvers attached to each sub-Block
  - which Solution is produced ...

R^3Block

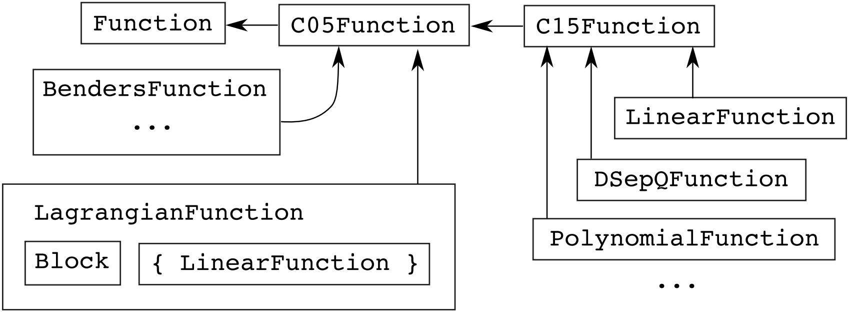
- Often reformulation crucial, but also relaxation or restriction: `get_R3_Block()` produces one, possibly using sub-Blocks'
- Obvious special case: copy (clone), should always work
- Available R^3Blocks `Block::`-specific, a `Configuration` needed
- R^3Block completely independent (new `Variable`/`Constraints`),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

Outline

First Basic Implementations

- `Variable::ColVariable` implements "value = one single real", possibly restricted to $\mathbb{Z}$, with (possibly infinite) bounds
- `Modification::ColVariableModification` (change bounds, type)
- `Constraint::RowConstraint` implements "$l \leq$ a real $\leq u$"
- Has dual variable attached to it (single real)
- `Modification::RowConstraintModification` (change l, u)
- `RowConstraint::FRowConstraint`: "a real" given by a `Function`
- `RealObjectiveFunction::FRealObjectiveFunction`: "value" given by a `Function`

Function



- `Function` only deals with (real) values
- Approximate computation supported in a quite general way[26]
- Asynchronous evaluation still not defined
- Handles set of `Variables` upon which it depends
- `FunctionModification[Variables]` for "easy" changes ⟹ reoptimization (shift, adding/removing "quasi separable" `Variables`)

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

C05Function

- 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

LagrangianFunction

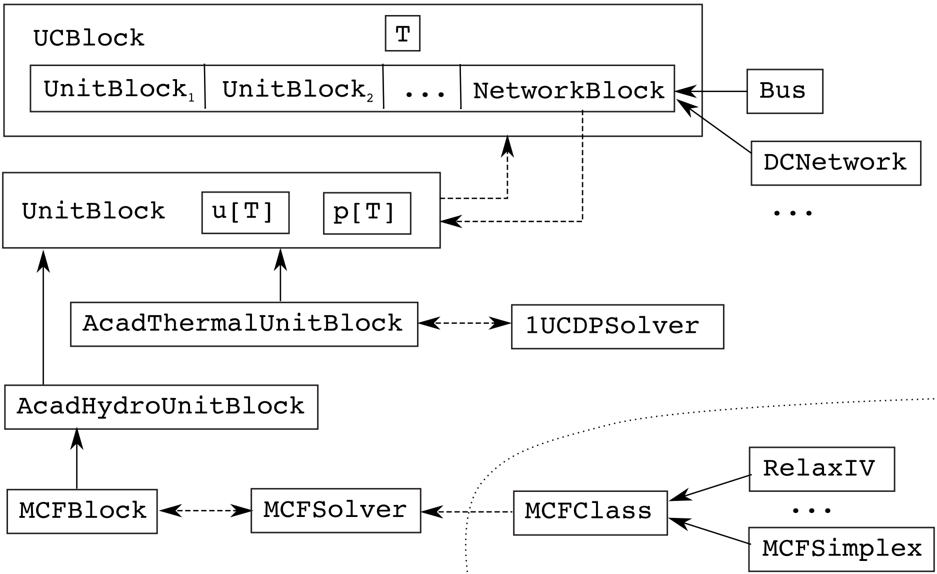
- C05Function::LagrangianFunction has one isolated Block + set of (so far) LinearFunction to define Lagrangian term
- evaluate() = Block.get_registered_solvers()[i].solve(): asynchronous Solver $\Longrightarrow$ asynchronous Function
- Solutions extracted from Block $\equiv$ linearizations
- Solver provides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

Other useful stuff

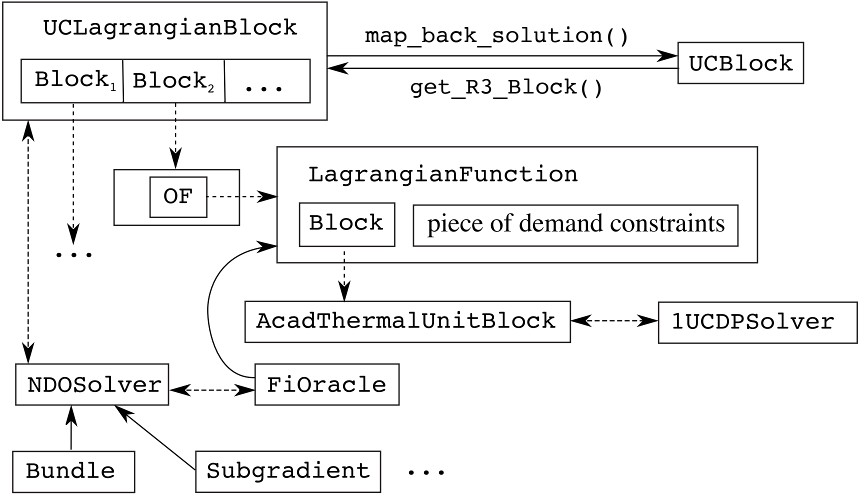
- `un_any_thing()` template functions/macros to extract (`std::vector` or `boost::multi_array` of) (`std::list` of) `Variable`/`Constraints` out of a `boost_any` and work on that
- `Solution::ColVariableSolution` uses the abstract representation of any `Block` that only have (`std::vector` or `boost::multi_array` of) (`std::list` of) `ColVariables` to read/write the solution
- `Solution::RowConstraintSolution` uses the abstract representation of any `Block` that only have (...) `RowConstraints` to read/write the dual solution
- Of course, `Solution::CVFRSolution` ...
- `Solver::MILPSolver` solves with `Cplex` any `Block` that only has (...) `ColVariables`, `FRowConstraints` and `FRealObjectiveFunction` with `LinearFunctions` (uses the abstract representation)

Outline

UCBlock and Companion Classes



UCLagrangianBlock



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

Outline

A Lot of Work, Then Maybe Conclusions

- Alpha version, not all the features you have seen are complete
- Design principles have kept evolving, new ideas continue to crop up
- Core nicely general, but only success in applications validate it
- Heavily slanted towards optimization, useful for numerical analysis?
- Really $\neq$ from all I've seen so far, had to invent almost everything
- Overhead still largely unknown (although C++ efficient)
- Asynchronous still to be figured out (but very relevant)
- Clearly not for the faint of heart . . .

A Lot of Work, Then Maybe Conclusions

- Alpha version, not all the features you have seen are complete
- Design principles have kept evolving, new ideas continue to crop up
- Core nicely general, but only success in applications validate it
- Heavily slanted towards optimization, useful for numerical analysis?
- Really $\neq$ from all I've seen so far, had to invent almost everything
- Overhead still largely unknown (although C++ efficient)
- Asynchronous still to be figured out (but very relevant)
- Clearly not for the faint of heart . . .
  but when it'll work it will be useful in many applications
- Implementing general, flexible methods for heterogeneous, multi-level structured problems is highly complex, have to make the tools first

We are trying. Someone 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