A Structure-Conveying Modelling Language for Mathematical Programming

INYS, Druskininkai, Lithuania

Marco Colombo Jacek Gondzio Andreas Grothey
Jonathan Hogg Kristian Woodsend

School of Mathematics
University of Edinburgh

6 February 2008

Outline

Optimization

An Optimization problem tries to find the solution of

$$\min_{x \in \boldsymbol{R}^n} f(x) \text{ s.t. } g(x) \leq 0$$

where $f : \boldsymbol{R}^n \longrightarrow \boldsymbol{R}, g : \boldsymbol{R}^n \longrightarrow \boldsymbol{R}^m$ sufficiently smooth

Applications:

- Engineering
- Telecommunications (Network Routing)
- Financial Planning (Asset and Liability Management)
- many more

Interested in large scale problems ($> 10^6$ variables/constraints)

Structured Problems

Large scale optimization problems are usually structured:

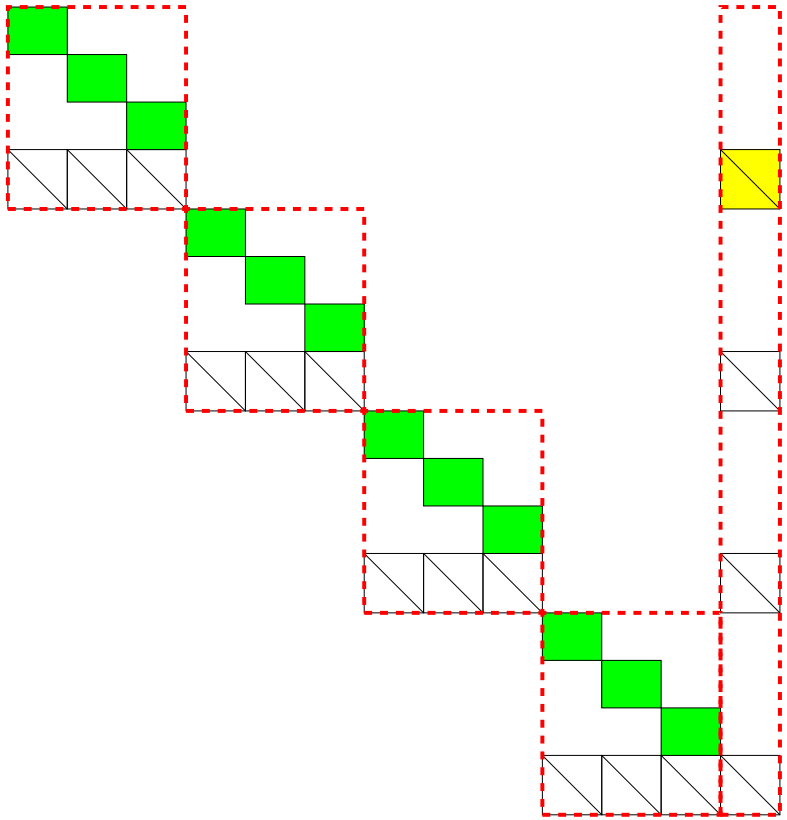
Typically appear as the result of a generation process

- discretization over time (dynamic structures)
- discretization over space (PDE-constrained optimization)
- discretization of probability space (Stochastic Programming)
- modelling inherent underlying structure (company consisting of several divisions, etc)

The structure of the problem implies a structure of the problem defining functions f, g and their derivatives

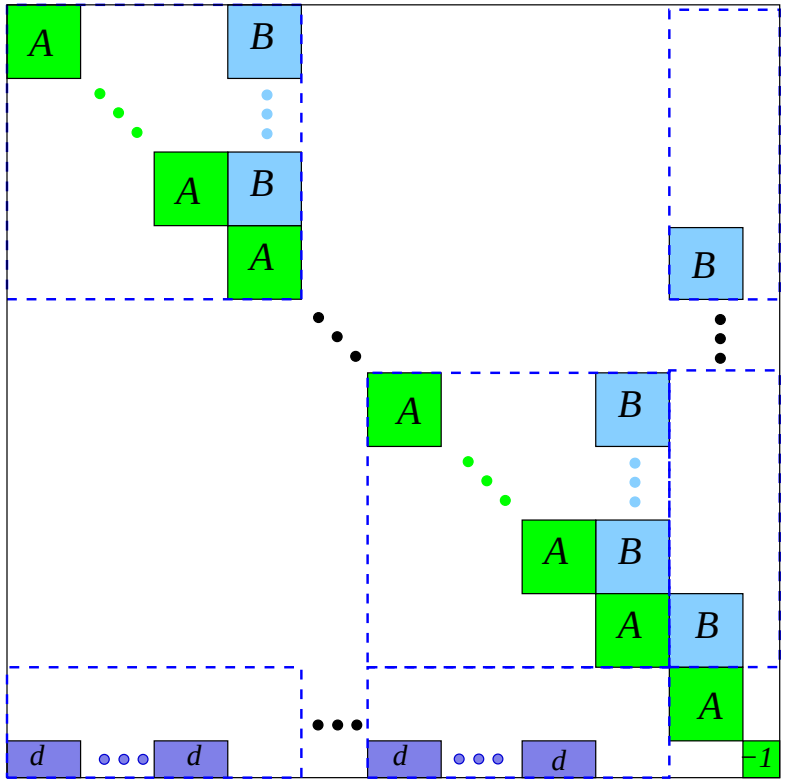
Example: Multicommodity Survivable Network Design (MSND)

Structure of ∇g:



Example: Asset Liability Management (ALM)

Structure of ∇g:



Structure Exploiting Solvers

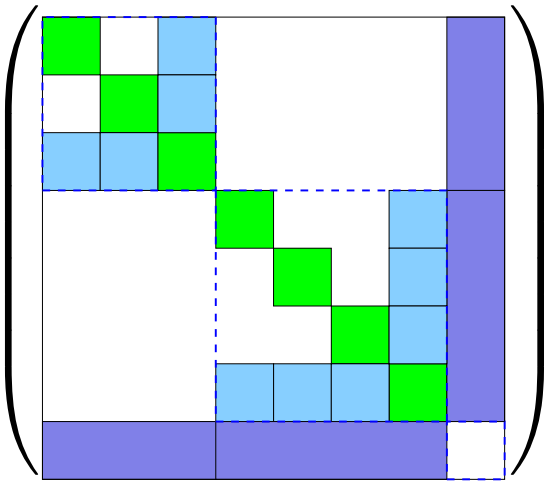
The structure of these problems can be effectively exploited

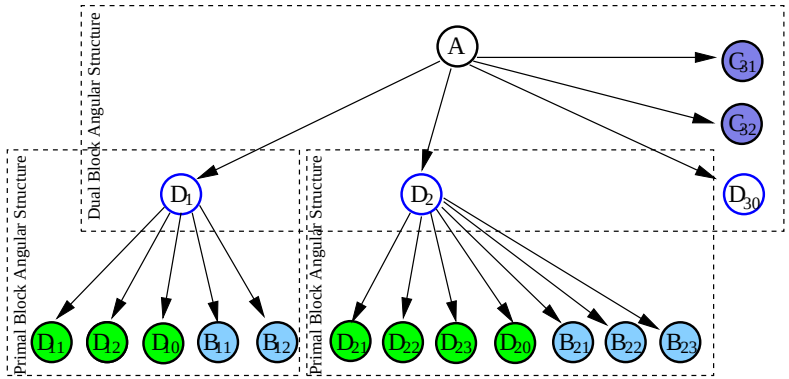
- to speed up solution time $\mathcal{O}(\text{blocks})$ vs $\mathcal{O}(n^3)$
- to parallelise the solution process

Popular structure exploiting techniques

- Decomposition (Benders', Dantzig-Wolfe, etc)
- Linear Algebra within Interior Point Methods

OOPS: Object Oriented Parallel Solver





- Tree representation of problem matrices
- Tree sparse linear algebra implementation
- Able to solve ALM problem with 1.01×10^9 variables on 1280 processors (HPCx) in under 2 hours (in 2005).
- Largest optimization problem solved by direct methods

Outline

Modelling Languages

Any solver for optimisation problems needs to evaluate functions f, g and its derivatives. Could write code to do this

- tedious,
- error prone,
- especially for large problems...

Modelling Languages

Any solver for optimisation problems needs to evaluate functions f, g and its derivatives. Could write code to do this

- tedious,
- error prone,
- especially for large problems...

Better to use a **Modelling Language:**
(AMPL, GAMS, XpressMP, AIMMS, etc)

- Concise algebraic expression of constraints and objective
- Automatic evaluation of derivatives
- Separation of **model** and **data**

Modelling Languages

Any solver for optimisation problems needs to evaluate functions f, g and its derivatives. Could write code to do this

- tedious,
- error prone,
- especially for large problems...

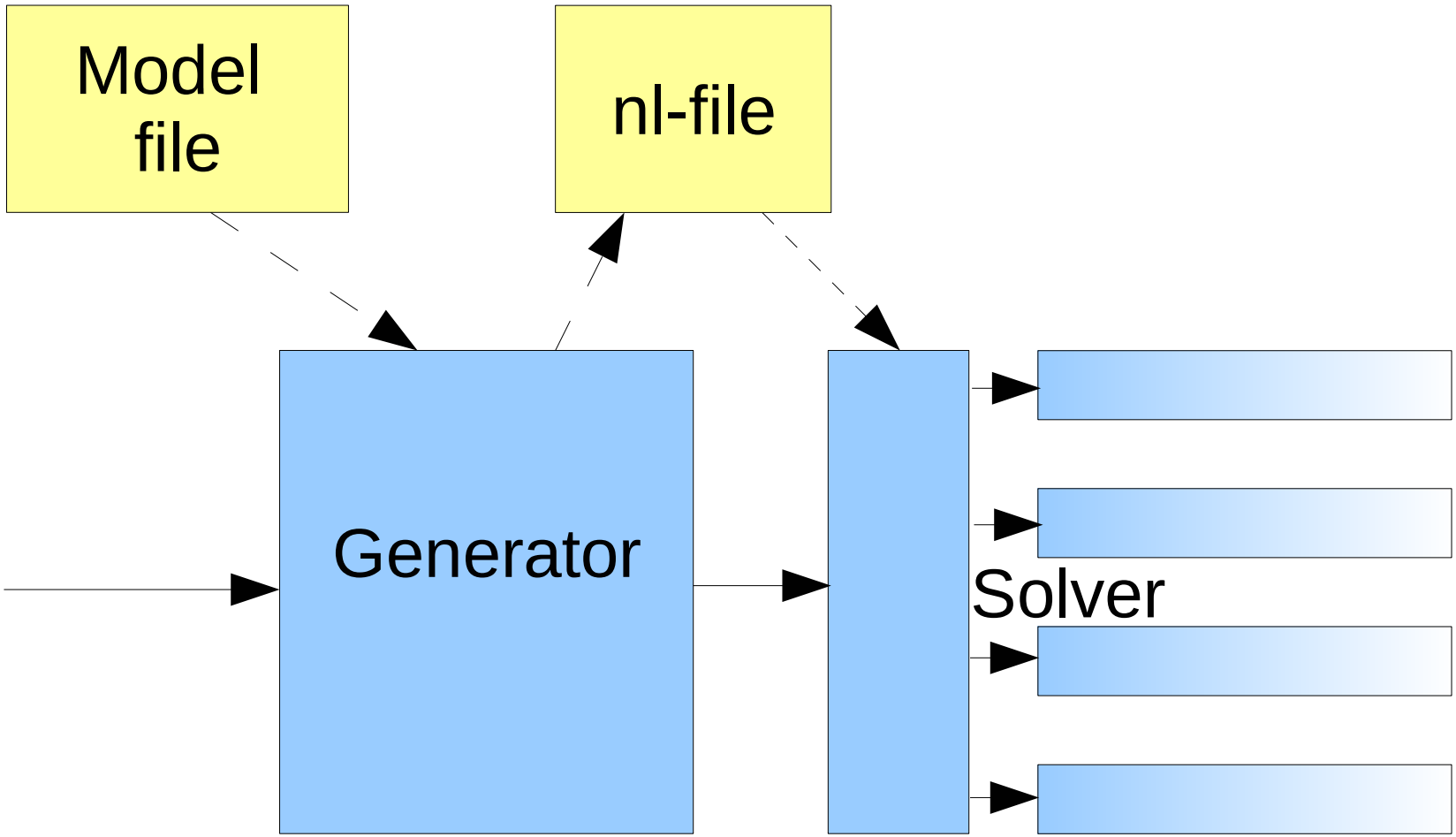
Better to use a **Modelling Language:**
(AMPL, GAMS, XpressMP, AIMMS, etc)

- Concise algebraic expression of constraints and objective
- Automatic evaluation of derivatives
- Separation of **model** and **data**

Problems

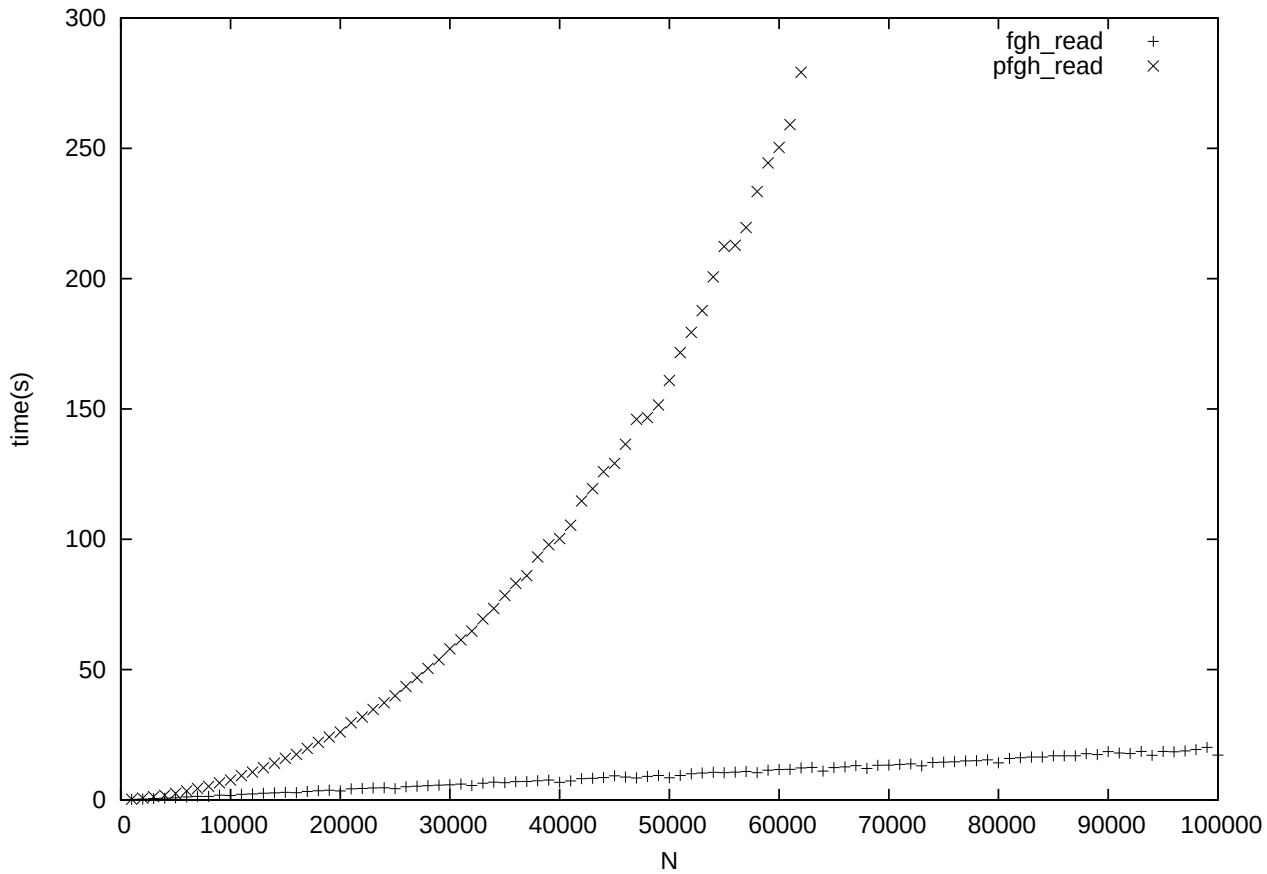
- Cannot express problem structure
  $\Rightarrow$ Structure of the problem not passed to the solver
- No attempt to parallelise the generation process

Modelling Languages: Flow of control



Modelling Languages: Generation Speed

Problem generation with AMPL: LP vs QP



$\Rightarrow$ More than ≈ 100000 variables are problematic

Outline

Example problem: Survivable Network Design

Network described by:

- nodes $i \in \mathcal{V}$, and
- directed arcs $j \in \mathcal{E}$, with base capacity Cap_j

Convenient to represent the network using node-arc incidence matrix $A \in \boldsymbol{R}^{|\mathcal{V}| \times |\mathcal{E}|}$, where:

$$A_{i,j} = \begin{cases} -1 & \text{node } i \text{ is origin of arc } j \\ 1 & \text{node } i \text{ is target of arc } j. \end{cases}$$

$x \geq 0 : Ax = b, x \in \boldsymbol{R}^{|\mathcal{E}|}$ is a feasible flow allocation for routing a demand b.

Example problem: Survivable Network Design

In the *Multicommodity Network Flow* (MCNF) problem, we have:

- a number of commodities $\mathcal{C}$
- demand vectors for the commodities $b_k, \forall k \in \mathcal{C}$
- a shared network of limited capacity $Cap_j, \forall j \in \mathcal{E}$

A feasible set of flows x_k will satisfy:

$$\begin{aligned} Ax_k &= b_k, & \forall k \in \mathcal{C} \\ \sum_{k \in \mathcal{C}} x_k &\leq Cap & \forall k \in \mathcal{C} \\ x_k &\geq 0 & \forall k \in \mathcal{C} \end{aligned}$$

Example problem: Survivable Network Design

In the *Multicommodity Survivable Network Design* (MSND) problem, we seek a network robust enough to survive the failure of a *single* arc.
What is the minimum spare capacity $s \in \boldsymbol{R}^{|\mathcal{E}|}$ required?

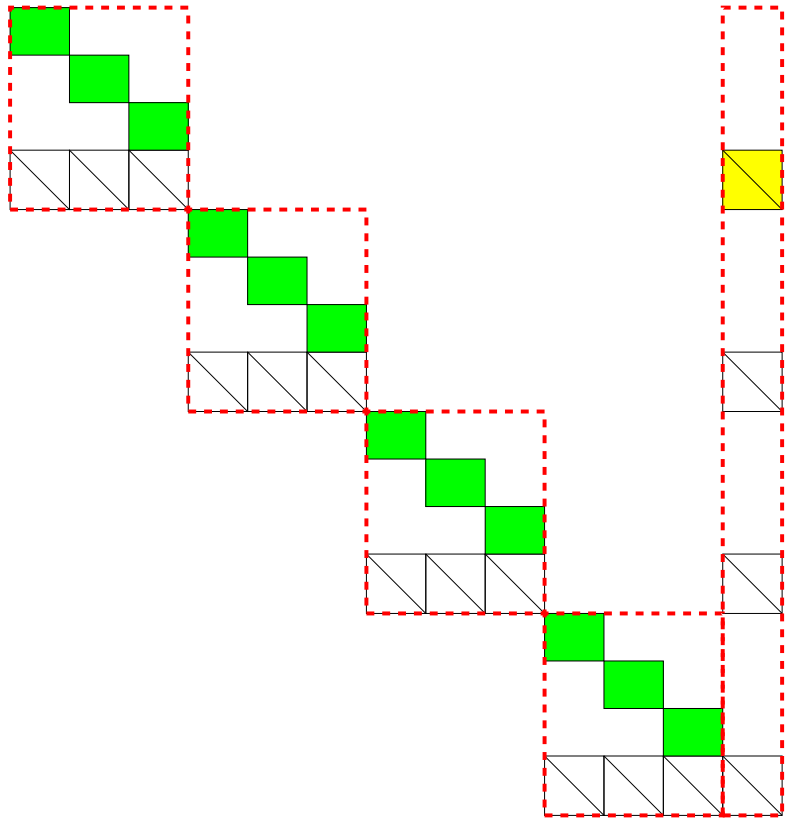
We model the network by removing individual arcs $i \in \mathcal{E}$. Let $A^{(i)}, i \in \mathcal{E}$ be A with i-th arc missing.
Then MSND problem can be written as:

$$\begin{array}{lll} \min & \sum_{i \in \mathcal{E}} c_i s_i & \\ \text{s.t.} & A^{(i)} x_k^{(i)} = b_k, & \forall k \in \mathcal{C}, i \in \mathcal{E} \\ & \sum_{k \in \mathcal{C}} x_{k,j}^{(i)} \leq Cap_j + s_j & \forall j \in \mathcal{E}, i \in \mathcal{E} \\ & x, s \geq 0 & \end{array} \tag{1}$$

Structure of MSND model

MSND with 4 commodities:

- MCNF sub-problems (red) have primal block-angular structure
  - Flow balance constraints for each network (green),
  - and arc capacity constraints (bottom border).
- Capacity constraints involve global spare capacity variables s (far right border, yellow)
- MSND problem displays a dual-block angular structure (red)



AMPL model for MSND

```
set NODES; set COMM;
set ARCS within NODES cross NODES;
...
var Flow{ARCS, COMM, ARCS} >= 0;
var sparecap{ARCS} >= 0;

subject to FlowBalanceMissingArcs
    {ma in ARCS, k in COMM, j in NODES}:
  sum{l in ARCS:l~=ma,target(l)==j} Flow[ma,k,l]
      + if (comm_source[k]==j) then comm_demand[k]
    = sum{l in ARCS:l~=ma,source(l)==j} Flow[ma,k,l]
      + if (comm_target[k]==j) then comm_demand[k];

subject to CapacityMissingArcs{ma in ARCS, l in ARCS}:
  sum{k in COMM} Flow[ma,k,l] <= basecap[l] + sparecap[l];

minimize obj: sum{i in ARCS} sparecap[i]*cost[i];
```

Criticisms of AMPL model

There are a few obvious deficiencies with this model:

- Triple indexing of flow variables and balance constraints. Complex conditions are error-prone e.g. `{l in ARCS:l~=ma,target(l)==j}` in `FlowBalanceMissingArcs`
- The structure of the problem is not apparent . . .

Outline

Structured parallel modelling language

Proposal for a Structured Parallel Modelling Language (SPML):

- Mimic the "block" nature of the problem using `block` keyword:

```
block nameOfBlock{j in NODES}: {
   ...
}
```

- Blocks can contain sets, variables and constraints, even nested blocks.
- Scope of these elements delimited by `block { ... }`
- Reference variables outside their scope using object-oriented syntax:

```
nameOfBlock[j].nameOfElement
```

MSND model in SPML

```
set NODES; set COMM;
set ARCS within (NODES cross NODES);
...
var sparecap{ARCS} >= 0;

block MCNFArcs{ma in ARCS}: {
  set ARCSDIFF = ARCS diff {ma};  % local ARCS still present

  block Net{k in COMM}: {
    var Flow{ARCSDIFF} >= 0;
    subject to FlowBalance{j in NODES}:
      sum{l in ARCSDIFF:target(l)==j} Flow[l]
        + if (comm_source[k]==j) then comm_demand[k]
      = sum{l in ARCSDIFF:ls==j} Flow[l]
        + if (comm_target[k]==j) then comm_demand[k];
  }

  subject to Capacity{l in ARCSDIFF}:
     sum{k in COMM} Net[k].Flow[l] <= basecap[l] + sparecap[l];
}
minimize obj: sum{i in ARCS} sparecap[i]*cost[i];
```

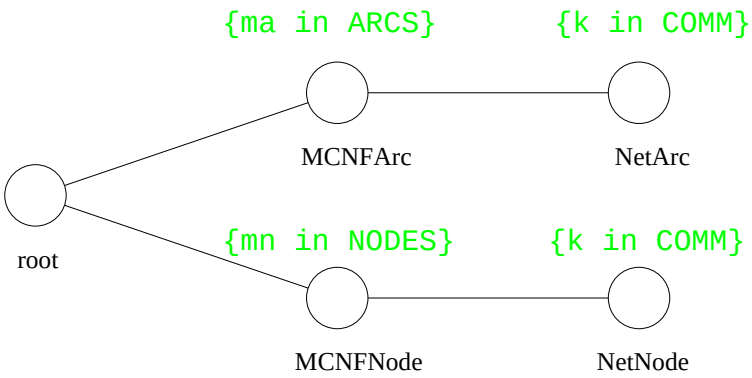
Extensions to the MSND model

The MSND model as presented here is not particularly complex.

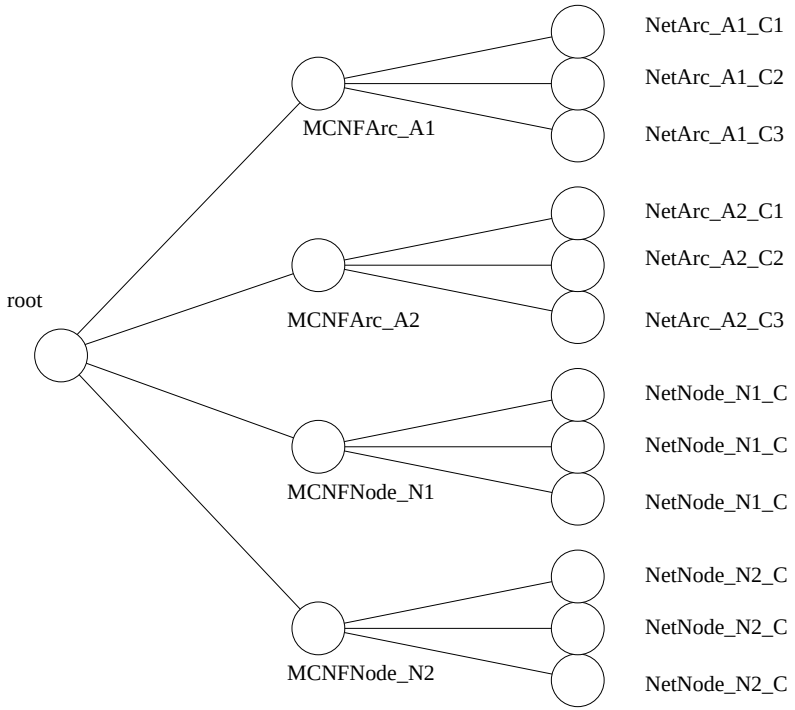
Further factors could be incorporated:

- Demand could be stochastic
  $\Rightarrow$ Introduces an extra layer of nesting into the structure.
- Design the network to be robust against *node* failure as well as *arc* failure
  $\Rightarrow$ Introduces further primal block-angular sub-problems alongside the MCNF arc-failure sub-problems.

Prototype and expanded model



"Prototype" blocks, described in the model



"Expanded" blocks, structurally the same but each different in data

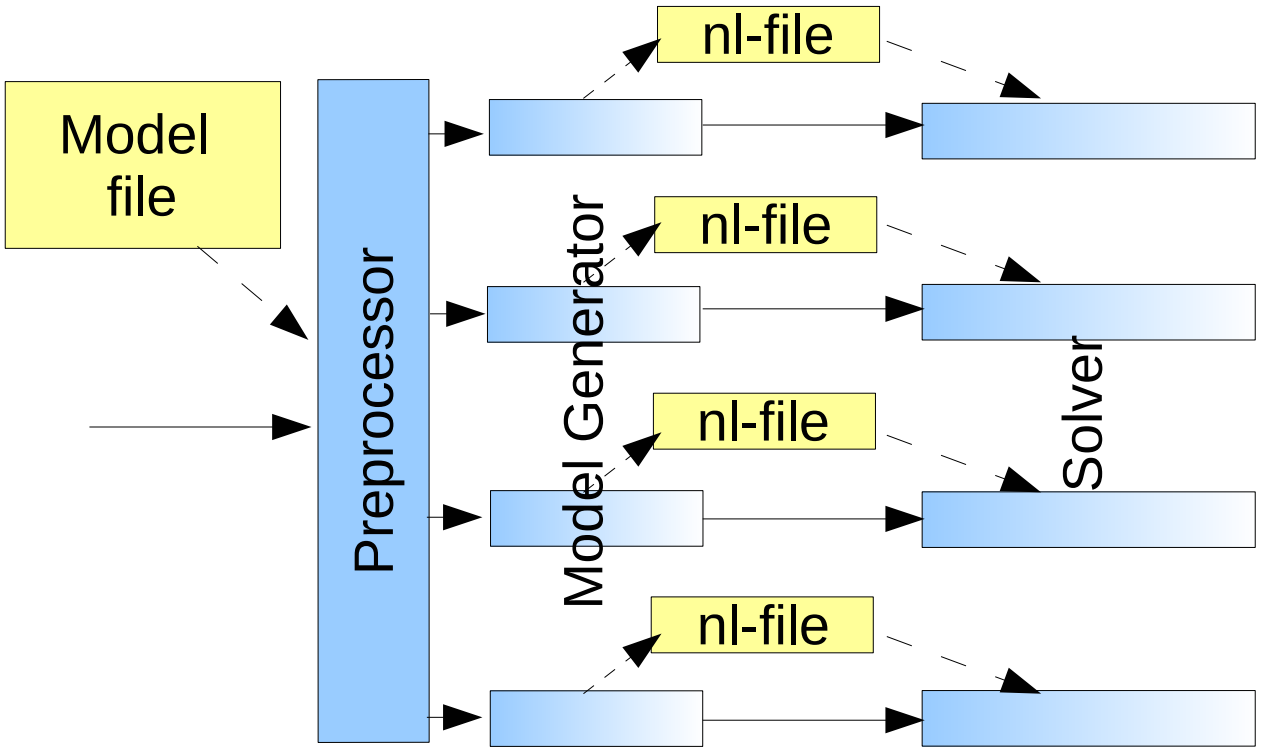
Outline

Overview of processing steps

SPML is implemented as a pre-/postprocessor to an existing modelling language (AMPL):

1. Pre-processor *parses* SPML file, and
   - analyses structure to
   - extract prototype model tree;
   - extract component dependency graph
2. Distribute problem
   - recursively expand prototype tree and assign to (groups of) processors
3. Parallel model generation
   - write sub-model AMPL files (using structure and dependency graph)
   - generate sub-model *.nl file
4. Solver is called (in parallel)

Parallelisation



Outline

Status & Future Work

Status

- Working implementation linked to OOPS (although early days...)

Future Work

- Stochastic Programming
- Link to other solvers (Decomposition solvers)
- Get rid of AMPL dependency
- Do some proper benchmarking

Conclusions

We present a Structured Parallel Modelling Language (SPML) which

- is based on standard AMPL;
- uses object-oriented extensions to construct models from sub-models;
- these models are expressed elegantly and more naturally;
- with a noticeable reduction in complexity;
- captures the inherent structure of the problem, allowing this to be conveyed to the solver;
- and parallelises the model generation.

Conclusions

We present a Structured Parallel Modelling Language (SPML) which

- is based on standard AMPL;
- uses object-oriented extensions to construct models from sub-models;
- these models are expressed elegantly and more naturally;
- with a noticeable reduction in complexity;
- captures the inherent structure of the problem, allowing this to be conveyed to the solver;
- and parallelises the model generation.

Thank you!