An Open-Source C++ Modelling Environment for Stochastic Programming

Tim Hultberg[1] Michal Kaut[2] Alan King[3]

[1]EUMETSAT, Germany

[2]Molde University College, Molde, Norway
michal.kaut@himolde.no

[3]IBM Research, Yorktown Heights, New York

APMOD 2008, 30 May 2008

Outline

Motivation

Why open source?

- No *restrictions* on use, so students can have it on their machines.
- It is possible to *adjust* the tools to any particular needs.

Why modelling in C++?

- Why use programming language, instead of a modelling tool?
  - We have the power and flexibility of a general programming language at our disposal.
  - We can, for example, solve an SP as a part of some heuristic, or generate special cuts as a part of a solution procedure.
  - The price we pay is the ease of use, compared to the traditional AML's (algebraic modelling languages) like AMPL or GAMS.
- Why C++?
  - All the tools we build upon are in C++.

Motivation

Why open source?

- No *restrictions* on use, so students can have it on their machines.
- It is possible to *adjust* the tools to any particular needs.

Why modelling in C++?

- Why use programming language, instead of a modelling tool?
  - We have the power and flexibility of a general programming language at our disposal.
  - We can, for example, solve an SP as a part of some heuristic, or generate special cuts as a part of a solution procedure.
  - The price we pay is the ease of use, compared to the traditional AML's (algebraic modelling languages) like AMPL or GAMS.
- Why C++?
  - All the tools we build upon are in C++.

Motivation

Why open source?

- No *restrictions* on use, so students can have it on their machines.
- It is possible to *adjust* the tools to any particular needs.

Why modelling in C++?

- Why use programming language, instead of a modelling tool?
  - We have the power and flexibility of a general programming language at our disposal.
  - We can, for example, solve an SP as a part of some heuristic, or generate special cuts as a part of a solution procedure.
  - The price we pay is the ease of use, compared to the traditional AML's (algebraic modelling languages) like AMPL or GAMS.
- Why C++?
  - All the tools we build upon are in C++.

Motivation

Why open source?

- No *restrictions* on use, so students can have it on their machines.
- It is possible to *adjust* the tools to any particular needs.

Why modelling in C++?

- Why use programming language, instead of a modelling tool?
  - We have the power and flexibility of a general programming language at our disposal.
  - We can, for example, solve an SP as a part of some heuristic, or generate special cuts as a part of a solution procedure.
  - The price we pay is the ease of use, compared to the traditional AML's (algebraic modelling languages) like AMPL or GAMS.
- Why C++?
  - All the tools we build upon are in C++.

Why stochastic programming?

Outline

The COIN-OR project

Computational **In**frastructure for **O**perations **R**esearch (COIN-OR):

- A collection of ca. 30 open-source projects developing software for the operations research (OR) community:
  - solvers for LP, (M)IP, NLP, MINLP problems
  - model classes, interfaces and other support routines
- The goal of the project is "*to create for mathematical software what the open literature is for mathematical theory*". (according to `http://www.coin-or.org/`)
- Most of the projects are written in C++, the rest in C and Fortran. All projects provide `configure` and `make` scripts for POSIX environment, and many also provide VC++ project files.
- Mostly licensed under the *Common Public License* (CPL).

COIN-OR Open Solver Interface (OSI)

- A class including a complete interface to LP and (M)IP problems.
  - all the problem data
  - input and output routines (MPS etc.)
  - solution routines (interface for a solver)
- Derived classes for connected to different solvers; the supported solvers are CPLEX, FortMP, GLPK, MOSEK, OSL, SoPlex, and Xpress-MP, plus all the COIN-OR solvers.
- It is very easy to change the solver: everything we need to do is to change two lines in the code:
  - include a different header file
  - change the class of our OSI object

  and then link it with a different library – all of which can be done using build targets in a makefile or an IDE.

Stochastic Modelling Interface (SMI) – Alan King

Interface for stochastic-programming models. It implements:

- a scenario-tree structure, using an SMPS-like format
- SMPS file reader
- scenario generation from discrete random variables
- generation of an OSI object of the deterministic equivalent
- once the problem has been solved, accessing solution by stage and scenario
- nested Benders solver – coming soon

Stochastic Modelling Interface (SMI) – Alan King

Interface for stochastic-programming models. It implements:

- a scenario-tree structure, using an SMPS-like format
- SMPS file reader
- scenario generation from discrete random variables
- generation of an OSI object of the deterministic equivalent
- once the problem has been solved, accessing solution by stage and scenario
- nested Benders solver – coming soon

Stochastic Modelling Interface (SMI) – Alan King

Interface for stochastic-programming models. It implements:

- a scenario-tree structure, using an SMPS-like format
- SMPS file reader
- scenario generation from discrete random variables
- generation of an OSI object of the deterministic equivalent
- once the problem has been solved, accessing solution by stage and scenario
- nested Benders solver – coming soon

Can be used directly as a solver for SP models given in SMPS format.

Stochastic Modelling Interface (SMI) – Alan King

Interface for stochastic-programming models. It implements:

- a scenario-tree structure, using an SMPS-like format
- SMPS file reader
- scenario generation from discrete random variables
- generation of an OSI object of the deterministic equivalent
- once the problem has been solved, accessing solution by stage and scenario
- nested Benders solver – coming soon

Can be used directly as a solver for SP models given in SMPS format.
To use it as an SP modelling tool, we need:

- make an OSI object of the core (1-scenario) model
- structure/shape of the scenario tree
- associate all the variables and constraints with stages

Formulation of **L**inear **O**ptimization **P**roblems in **C++** (FlopC++) – Tim Hultberg

- An LP/MIP modelling environment implemented as C++ classes
- The code is almost as readable as in traditional AML's:

```
MP_model ::getDefaultModel ();
MP_set S(numS), D(numD); // sources and destinations
MP_subset <2> Links(S,D); // sparse subset of S×D – needs data
MP_data SUPPLY(S), DEMAND(D), COST(Links); // needs data
MP_variable x(Links);
MP_constraint supply(S), demand(D);
supply(S) = sum ( Links(S,D), x(Links) ) <= SUPPLY(S);
demand(D) = sum ( Links(S,D), x(Links) ) >= DEMAND(D);
minimize ( sum(Links, COST(Links) * x(Links)) );
x.display ("Optimal solution:");
```

- We can have several variants of a model by defining the common parts as a class derived from **MP_model** and then model the variants as its derived classes.

Formulation of Linear Optimization Problems in C++ (FlopC++) – Tim Hultberg

- An LP/MIP modelling environment implemented as C++ classes
- The code is almost as readable as in traditional AML's:

```
MP_model ::getDefaultModel ();
MP_set  S(numS), D(numD); // sources and destinations
MP_subset <2> Links(S,D);   // sparse subset of S×D – needs data
MP_data  SUPPLY(S), DEMAND(D), COST(Links);   // needs data
MP_variable  x(Links);
MP_constraint  supply(S), demand(D);
supply(S) = sum ( Links(S,D), x(Links) ) <= SUPPLY(S);
demand(D) = sum ( Links(S,D), x(Links) ) >= DEMAND(D);
minimize  ( sum(Links, COST(Links) * x(Links)) );
x.display ("Optimal solution:");
```

- We can have several variants of a model by defining the common parts as a class derived from **MP_model** and then model the variants as its derived classes.

Formulation of Linear Optimization Problems in C++ (FlopC++) – Tim Hultberg

- An LP/MIP modelling environment implemented as C++ classes
- The code is almost as readable as in traditional AML's:

```
MP_model ::getDefaultModel ();
MP_set S(numS), D(numD); // sources and destinations
MP_subset <2> Links(S,D);   // sparse subset of S×D – needs data
MP_data SUPPLY(S), DEMAND(D), COST(Links);   // needs data
MP_variable x(Links);
MP_constraint supply(S), demand(D);
supply(S) = sum ( Links(S,D), x(Links) ) <= SUPPLY(S);
demand(D) = sum ( Links(S,D), x(Links) ) >= DEMAND(D);
minimize ( sum(Links, COST(Links) * x(Links)) );
x.display ("Optimal solution:");
```

- We can have several variants of a model by defining the common parts as a class derived from **MP_model** and then model the variants as its derived classes.

Outline

FlopC++ for stochastic programming

- A stochastic-programming extension to FlopC++ is under development, but it is not ready yet.
- For now, we can do the same as in standard AML's, i.e. formulate the deterministic equivalent—with the usual disadvantages.
- An alternative is to write C++ classes for models at each stage and then construct the SP model by making objects for each node of the scenario tree – more on that on the next slide. . .
- The above approach can also be combined with SMI:
  - Write the *core* (deterministic) model in FlopC++ and pass it to SMI as an OSI object.
  - Pass to SMI the stages of all variables and constraints.
  - Access the scenario solutions from within FlopC++.

FlopC++ for stochastic programming

- A stochastic-programming extension to FlopC++ is under development, but it is not ready yet.
- For now, we can do the same as in standard AML's, i.e. formulate the deterministic equivalent—with the usual disadvantages.
- An alternative is to write C++ classes for models at each stage and then construct the SP model by making objects for each node of the scenario tree – more on that on the next slide. . .
- The above approach can also be combined with SMI:
  - Write the *core* (deterministic) model in FlopC++ and pass it to SMI as an OSI object.
  - Pass to SMI the stages of all variables and constraints.
  - Access the scenario solutions from within FlopC++.

FlopC++ for stochastic programming

- A stochastic-programming extension to FlopC++ is under development, but it is not ready yet.
- For now, we can do the same as in standard AML's, i.e. formulate the deterministic equivalent—with the usual disadvantages.
- An alternative is to write C++ classes for models at each stage and then construct the SP model by making objects for each node of the scenario tree – more on that on the next slide...
- The above approach can also be combined with SMI:
  - Write the *core* (deterministic) model in FlopC++ and pass it to SMI as an OSI object.
  - Pass to SMI the stages of all variables and constraints.
  - Access the scenario solutions from within FlopC++.

FlopC++ for stochastic programming

- A stochastic-programming extension to FlopC++ is under development, but it is not ready yet.
- For now, we can do the same as in standard AML's, i.e. formulate the deterministic equivalent—with the usual disadvantages.
- An alternative is to write C++ classes for models at each stage and then construct the SP model by making objects for each node of the scenario tree – more on that on the next slide. . .
- The above approach can also be combined with SMI:
  - Write the *core* (deterministic) model in FlopC++ and pass it to SMI as an OSI object.
  - Pass to SMI the stages of all variables and constraints.
  - Access the scenario solutions from within FlopC++.

We believe that the last option is the best one, providing we can find some way to auto-generate the information needed by SMI.

Object-oriented model using FlopC++

- Create a class **StageNode** for a model in one node of the scenario tree. This class should include:
  - pointer to parent and (if needed) children nodes
  - variables and constraints defined in the node's model
  - an **MP_expression** with the node's objective function
  - the node's probability
- Since the model is typically different in the first and last stage of the scenario tree, create derived classes **RootNode**, **MidStageNode** and **LeafNode**, leaving only the common features in **StageNode**.
- In addition, we need a description of the scenario tree structure.
- With these classes, one can generate a stochastic model in FlopC++ alone. However, without any link to SMI, we again end up solving the deterministic equivalent. . .

Object-oriented model using FlopC++ and SMI

- Based on the previous model, but we only create nodes for the core (1-scenario) model.
- Each node model must have a method to report all its variables and constraints. This way, we can get the stage information for all variables and constraints.
  - At the moments, the lists of variables and constraints have to be created manually in the constructor of the object – there should be a more elegant way of doing this. . .
- In addition, the reporting of the objective function has to be changed, as one node of the core model corresponds to many nodes in the scenario tree. We solve this by passing the scenario data and variable values to the reporting function as parameters.

Object-oriented model using FlopC++ and SMI *cont.*

Current status:

- An example implementing a simple investment model has been added to the SMI project, as file `investment.cpp`.
- In the example, all the classes are being defined as a part of the model. In addition, the list of variables and constraints in each node has to be filled manually.

Future work:

- The next step is to integrate all the base classes and methods into FlopC++ and/or SMI, so that the users will only need to write the model-specific part.
- Another issue is handling of expected-value-type constraints: they can not be modelled using this approach—and if if we could express them, Smi won't handle them.

Object-oriented model using FlopC++ and SMI *cont.*

Current status:

- An example implementing a simple investment model has been added to the SMI project, as file `investment.cpp`.
- In the example, all the classes are being defined as a part of the model. In addition, the list of variables and constraints in each node has to be filled manually.

Future work:

- The next step is to integrate all the base classes and methods into FlopC++ and/or SMI, so that the users will only need to write the model-specific part.
- Another issue is handling of expected-value-type constraints: they can not be modelled using this approach—and if if we could express them, Smi won't handle them.

Summary

- We have presented a modelling tool for stochastic programs, implemented as a C++ library.
- The tool is based on two open-source projects from the Coin-OR repository: FlopC++ and SMI.
- Combining the two projects allows for a natural-style (ALM-like) modelling, both of the core model and the stochastic structure.

- While the proposed approach is already usable, more work is needed to make it easier to use, as well as more flexible.

The End