Stochastic Extensions to FlopC++

Christian Wolf, Achim Koberstein, and Tim Hultberg

Abstract We extend the open-source modelling language FlopC++, which is part of the COIN-OR project, to support multi-stage stochastic programs with recourse. We connect the stochastic version of FlopC++ to the existing COIN class stochastic modelling interface (SMI) to provide a direct interface to specialized solution algorithms. The new stochastic version of FlopC++ can be used to specify scenario-based problems and distribution-based problems with independent random variables. A data-driven scenario tree generation method transforms a given scenario fan, a collection of different data paths with specified probabilities, into a scenario tree. We illustrate our extensions by means of a two-stage mixed integer strategic supply chain design problem and a multi-stage investment model.

1 Introduction

Stochastic programming is about optimal decision making under uncertainty [2]. Uncertain parameters are realistic in most practical decision problems [15], so it would be convenient to easily implement those problems. Modelling environments for deterministic mathematical problems are numerous and widely used. This is not the case for modelling environments for stochastic programming. An overview about existing modelling environments as well as detailed information can be found in [15, 9, 12], examples are SPiNE [14], SLP-IOR [9] and STOCHASTICS [15]. Support for modelling of stochastic programs was also added to some commercial

Christian Wolf
University of Paderborn, Paderborn, Germany e-mail: christian.wolf@dsor.de

Achim Koberstein
University of Paderborn, Paderborn, Germany e-mail: koberstein@dsor.de

Tim Hultberg
Eumetsat, Darmstadt e-mail: tim.hultberg@eumetstat.int

B. Hu et al. (eds.), *Operations Research Proceedings 2010*, Operations Researrch Proceedings, 333
DOI 10.1007/978-3-642-20009-0_53,

modelling software, e.g. GAMS, MPL, AIMMS, Xpress or Microsofts Solver Foundation.

The process of creating a solvable stochastic program instance is threefold. In the first step, the stochastic program can be implemented using a modelling language designed for that purpose. Algebraic modelling languages (AML) allow for models that are easier to build, read and thus maintain than the use of problem specific matrix generators [5].

A major difficulty for the modeller is the determination of the stochastic process of the real world and the translation into random variables with a certain distribution or statistical parameters. This leads to the second step which consists of the generation of random variates for all the random variables defined in the first step. The generation and combination process is called *scenario generation* and is an active research area of its own [11]. Kaut and Wallace state that there is no "scenario-generation method that would be best for all possible models". Therefore the scenario generation method has to be chosen with respect to the model. The AML should provide the modeller with the possibility to use his scenario generation method of choice.

In the third step, the stochastic problem defined by the core model and the scenario tree gets solved. Several different approaches are possible to reach this goal. The deterministic equivalent problem can be easily generated given the deterministic core model and the scenario tree. It can be solved with standard techniques. With a rising number of scenarios, this approach soon becomes infeasible, as the problem size grows linearly with the number of scenarios, which grows exponentially with respect to the number of (independent) random variables and the number of outcomes to each random variable. This is the so-called "curse of dimensionality".

Because of that effect decomposition methods were proposed that take advantage of the special structure of stochastic programs with recourse. There are two different types of problem formulations for stochastic programs with respect to the non-anticipativity constraints. The node-based or implicit formulation generates variables for every node in the scenario tree, whereas the scenario-based or explicit formulation introduces n copies of each decision variable for every stage, if n is the number of scenarios. All copies are then forced by the explicit non-anticipativity constraints to take the same value. Depending on the formulation different decomposition methods can be applied [6], so the modelling environment should be able to generate both formulations.

Our goal is to provide an open-source modelling environment which renders it possible to specify multistage stochastic programs with recourse and solve these programs. The environment is based on FlopC++ [8], which is part of the COIN-OR project [13]. The COIN-OR project provides open-source software for the operations research community. The storage of a stochastic program is done via the stochastic modelling interface (Smi), which is also part of COIN-OR. Our contribution is based on previous work in the direction of chaining FlopC++ and Smi together [10], which is a plausible approach, as these two projects are already present in COIN-OR. It is theoretically possible to solve all multistage stochastic programs in the deterministic equivalent form with any Osi capable solver. Furthermore spe-

cialized solver for stochastic programs can be connected, if they implement the stochastic solver interface we propose on the basis of Osi and Smi. We used an implementation of the L-shaped method with first stage integer variables [3] with this interface to solve practical problems. We also extended Smi to handle integer variables, so that we can model stochastic integer programs and solve them in the two-stage case. Furthermore the environment provides methods to compute the stochastic measures EVPI and VSS [2].

2 Stochastic Extensions

In the following, we introduce the new language constructs by means of an example, the well known investment model [2, 12].

```
MP_modell investmentModel(new OsiClpSolverInterface());
MP_data start,goal;
start() = 55; goal() = 80;
int numStage=4;int numScen=8; enum{asset1,asset2,numAssets};
```

Any Osi capable solver can be used to solve the model, it is given in the first line.

```
MP_stage T(numStage);
MP_scenario_set scen(numScen);
MP_set assets(numAssets);
```

A key concept in stochastic programming is the notion of a *stage*. The underlying time process of the model can be divided in stages, where each stage corresponds to a time period where new information becomes available and decisions have to be made after observing the new information. The set `MP_stage` specifies the special stage set, which is mandatory for every stochastic program.

```
double scenarios[numStage-1][numAssets][numScen] =
 {    // stage 2
    {1.25,1.25,1.25,1.25,1.06,1.06,1.06,1.06},//asset1
    {1.14,1.14,1.14,1.14,1.16,1.16,1.16,1.16} //asset2
 },{ // stage 3
    {1.21,1.21,1.07,1.07,1.15,1.15,1.06,1.06},//asset1
    {1.17,1.17,1.12,1.12,1.18,1.18,1.12,1.12} //asset2
 }, {// stage 4
    {1.26,1.07,1.25,1.06,1.05,1.06,1.05,1.06},//asset1
    {1.13,1.14,1.15,1.12,1.17,1.15,1.14,1.12}}};//asset2
MP_random_data returns(&scenarios[0][0][0],T,assets);
```

The random parameter `MP_random_data` is in general an algebraic combination of several other parameters or it refers to a random variable with a specific distribution, as it is the case in the example. A `ScenarioRandomVariable` is implicitly created during the initialization of `returns` to store the values for every scenario.

The expected number of entries is given via the special set `MP_scenario_set`. This is only mandatory if the problem is scenario-based. A `MP_random_data` variable can be used everywhere, where a normal `MP_data` variable has been used before, so it is fairly easy to switch from a deterministic to a stochastic model. One has to replace deterministic data with random data and assign a stage index to all second or later stage variables and constraints. No indexation over specific scenarios is required.

```
MP_variable x(T,assets),wealth(T),shortage(T),overage(T);
MP_constraint initialWealthConstr, returnConstr(T);
MP_constraint allocationConstr(T),goalConstr(T);
```

The variables and constraints of the model get initialized with the above statements.

```
initialWealthConstr() = sum(assets,x(0,assets)) == start();
allocationConstr(T) =sum(assets,x(T,assets))== wealth(T);
returnConstr(T+1) = sum(assets,returns(T+1,assets)*x(T,
   assets))== wealth(T+1);//only valid for stage 2 to 4
goalConstr(T.last()) = wealth(T.last()) == goal() +
                overage(T.last()) - shortage(T.last())
MP_expression valueFunction( -1.3*shortage(T.last()) +
                   1.1*overage(T.last()));
```

The second stage constraints are indexed over the set $T+1$, so the first stage is spared. The object `valueFunction` stores the objective function of the model. The expectation over all scenarios is added automatically by the environment.

```
investmentModel.setObjective( valueFunction );
investmentModel.solve(MP_model::MAXIMIZE);
```

The investment model is solved via the deterministic equivalent with the LP solver *Clp* through the `OsiClpSolverInterface`, but this can be changed in the first code line.

Distribution-based problems can also be specified with FlopC++ with the added `RandomVariable` hierarchy. A discrete continuous distribution with step width 0.5 in the interval $[0,3]$ can be specified with the following code.

```
RandomVariable* rPtr;
rPtr = new DiscreteContinuousRandomVariable(0,3,0.5);
```

Dependencies between different random variables can not be specified at the moment. Also inter-stage dependencies can not be modelled explicitly via random processes. It is however possible to generate variates according to sophisticated random processes with external tools and feed the data directly to a `ScenarioRandomVariable` as it was the case in the example.

Apart from file-based exchange via SMPS or OSiL [4] there is no common interface between modelling languages and solver for stochastic programs [7]. We therefore propose an interface similar to Osi that consists mainly of an Osi solver and a Smi instance that holds the model and the already generated scenario tree. A solver developer who implements this interface uses Smi as the data structure and the given solver to solve the subproblems.

3 Practical Experience

We implemented a two-stage strategic network model with binary variables on the first stage (basically a simplified version of the model given in [1]) in FlopC++. We connected a stochastic solver [3] that uses the integer L-shaped method with FlopC++ using our stochastic solver interface. Figure 1 shows that in this case the use of a specialized solver is advantageous to the use of standard solution techniques by contrasting the solution times of our stochastic solver with the solution times for the deterministic equivalent [1]. Thus the use of a modelling environment for stochastic programming in combination with an interface for stochastic solvers is highly useful, if these solvers are available. If not, the deterministic equivalent can still be generated and solved instead.

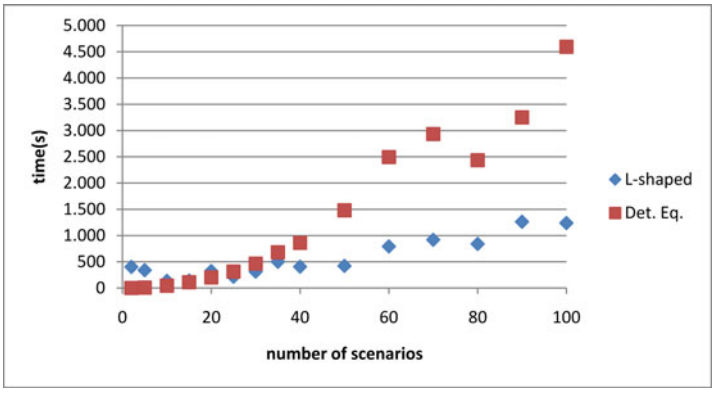

Fig. 1 Comparison of solution time of det. eq. and L-shaped method solver.

[1] All tests have been conducted on a MS Windows XP 64 bit, Intel Xeon 3,4 Ghz, 4 GB RAM with Cplex 11.0. The measured time is the average wall-clock time of three runs.

4 Conclusions and Future Work

We extended the existing open-source AML FlopC++ to support the modelling of multistage stochastic programs with recourse. This modelling framework eases the process of creating stochastic programs opposed to the manual generation of the deterministic equivalent or usage of the column based SMPS format. The environment is restricted to inter-stage independent random variables. In addition we propose an interface for stochastic solvers based on Smi and Osi. An integration of various discretization schemes for continuous random distributions or empirical data within the environment would be valuable [11] and is subject of future research.

References

1. Ralf Bihlmaier, Achim Koberstein, and René Obst. Modeling and optimizing of strategic and tactical production planning in the automotive industry under uncertainty. *OR Spectrum*, 31(2): 311–336, August 2009.
2. John R. Birge and François V. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, 1997.
3. Corinna Dohle. *Eine Implementierung des Benders-Dekompositionsverfahrens für allgemeine zweistufig stochastische Programme mit diskreten Stufe-1-Variablen.* Diploma thesis, University of Paderborn, April 2010.
4. R. Fourer, H. I. Gassmann, J. Ma, and R. K. Martin. An XML-based schema for stochastic programs. *Annals of Operations Research*, 166(1): 313–337, October 2008.
5. Robert Fourer. Modeling languages versus matrix generators for linear programming. *ACM Transactions on Mathematical Software*, 9(2): 143–183, June 1983.
6. Robert Fourer and L. Lopes. A management system for decompositions in stochastic programming. *Annals of Operations Research*, 142(1): 99–118, 2006.
7. Emmanuel Fragniere and Jacek Gondzio. Stochastic programming from modeling languages. In Stein W. Wallace and William T. and Ziemba, editors, *Applications of Stochastic Programming*, chapter 7, pages 95–113. SIAM, 2005.
8. Tim Helge Hultberg. FLOPC++ An Algebraic Modeling Language Embedded in C++. In *Operations Research Proceedings 2006*, pages 187–190. Springer Berlin-Heidelberg, 2006.
9. Peter Kall and Janos Mayer. *Stochastic linear programming: models, theory, and computation*. Springer, 2005.
10. Michael Kaut, Alan J King, and Tim Helge Hultberg. A C++ Modelling Environment for Stochastic Programming. Technical Report rc24662, IBM Watson Research Center, 2008.
11. Michael Kaut and Stein W. Wallace. Evaluation of scenario-generation methods for stochastic programming. *Stochastic Programming E-Print Series*, 14(1), 2003.
12. Miloš Kopa, editor. *On Selected Software for Stochastic Programming*. 2008.
13. Robin Lougee-Heimer. The Common Optimization Interface for Operations Research. *IBM Journal of Research and Development*, 47(1): 57–66, January 2003.
14. Christian Valente, Gautam Mitra, Mustapha Sadki, and Robert Fourer. Extending algebraic modelling languages for Stochastic Programming. *INFORMS Journal on Computing*, 21(1): 107–122, 2009.
15. S. W. Wallace and W. T. Ziemba, editors. *Applications of stochastic programming*. Society for Industrial Mathematics, 2005.