Stochastic MiniZinc*

Andrea Rendl[1], Guido Tack[1], and Peter J. Stuckey[2]

[1] National ICT Australia (NICTA) and Faculty of IT, Monash University, Australia
andrea.rendl@nicta.com.au, guido.tack@monash.edu
[2] National ICT Australia (NICTA) and University of Melbourne, Victoria, Australia
pstuckey@unimelb.edu.au

Abstract. Combinatorial optimisation problems often contain uncertainty that has to be taken into account to produce realistic solutions. However, existing modelling systems either do not support uncertainty, or do not support combinatorial features, such as integer variables and non-linear constraints. This paper presents an extension of the MINIZINC modelling language that supports uncertainty. Stochastic MINIZINC enables modellers to express combinatorial stochastic problems at a high level of abstraction, independent of the stochastic solving approach. These models are translated automatically into different solver-level representations. Stochastic MINIZINC provides the first solving technology agnostic approach to stochastic modelling we are aware of.

1 Introduction

In contrast to deterministic optimisation problems where all problem parameters are known a priori, *stochastic* optimisation problems deal with parameters that are *uncertain*, such as customer demand, resource capacities or travel times. This uncertainty has to be taken into account to provide realistic solutions.

Several stochastic modelling and solving systems have been established in recent years, such as AIMMS [16], AMPL [21,22] or GAMS [10]. These systems provide a strong support for stochastic linear problems on continuous variables, however have only limited or no support for problems with integer variables and non-linear constraints. Moreover, these systems force the modeller to commit to a particular solving approach at the modelling stage. This poses significant limitations to modellers who are interested in formulating stochastic *combinatorial* problems and solving them using different solving techniques.

For combinatorial problems, expressive high-level modelling languages have been developed, such as ESRA [2], Essence [4], Essence' [7], OPL [23], ZINC [11] and MINIZINC [12,13]. The benefit of high-level modelling is that users can focus on the problem formulation without committing to a particular solving approach. Some modelling systems, including MINIZINC, perform automated translations from the high-level model for different backend solvers, such as CP, MIP or SMT solvers. This way problems can be solved using different solving approaches without any background

* NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

knowledge of the respective technique. Unfortunately, none of the existing combinatorial modelling languages provides means for dealing with uncertainty. Stochastic extensions for OPL have been proposed in [25], however, they have never been made available [26].

In this work we present Stochastic MINIZINC, an extension of MINIZINC that supports uncertainty. It allows the user to augment deterministic models to stochastic models without the need to commit to a particular solving strategy. To solve these stochastic models, we present transformations from Stochastic MINIZINC to deterministic MINIZINC for three different stochastic solving approaches: scenario-based [19] and policy-based [24] deterministic equivalents, as well as progressive hedging [15]. These are the only known stochastic solving techniques that can deal with both integer decision variables and non-linear constraints. Our transformations generate standard MINIZINC, enabling the use of any backend solver technique that supports the MINIZINC tool-chain.

2 Background

Stochastic optimisation deals with problems where some parameters are *uncertain*. Uncertain parameters become known at some point in time, which divides the problem into different *stages*: the stage *before* the parameter is known, and the stage *after* it is known. Typically, decisions have to be made *before* the uncertain parameters become known. For instance, a car factory has to decide how many cars to produce before the actual demand is known. These 'beforehand'-decisions are called *first stage* decisions. The *second stage* decisions are made *after* the uncertain parameters are known. In the car factory, after the demand is known (and the production is completed), decisions may have to be made to deal with overproduction or shortage, depending on the actual demand. The aim of second stage decisions is to compensate for 'bad' first stage decisions with respect to the uncertain parameters. This is referred to as *recourse*.

The values of uncertain parameters can often be *estimated*, e.g. from historical data or simulations, resulting in a set of possible outcomes called *scenarios*. All approaches discussed here assume a finite number S of scenarios. Each scenario has a *weight* $w_1, \ldots, w_S$ reflecting its likeliness. In the car factory example, we may have three different scenarios for the demand, $d_1 = 13,000$, $d_2 = 16,000$ and $d_3 = 22,000$, with weights $w_1 = 1$, $w_2 = 6$ and $w_3 = 3$.

Stochastic events may happen repeatedly, resulting in *multiple* stages. For instance, the car factory may have to decide its production every quarter, taking into account the surplus/shortage from the previous quarter. The yearly production plan then contains four stochastic events (one for each quarter), dividing the problem into five stages, where decisions at stage i influence the decisions at stage $i + 1$. Problems with only one stochastic event are called *two-stage* problems, as opposed to *multi-stage* problems with more than one event.

The stages of a stochastic optimisation problem divide its objective into different parts. For instance, in the car factory example, the first stage objective is to minimise production costs, while the second stage objective is to minimise storage costs (in case of overproduction) and unmet demand penalties (in case of underproduction). The first stage objective is *independent* of the stochastic parameters. The objectives in

later stages do depend on the stochastic parameters. The overall objective therefore requires a probabilistic interpretation over all scenarios. Here, we focus on the *expected* value of the objective. Given an objective function $f(V)$ of the original problem formulation, the stochastic objective then becomes $\mathbb{E}[f(V)]$. Other interpretations, such as optimising for the worst case, can be realised in a similar manner.

Only three stochastic solving approaches exist for non-linear integer stochastic problems, all of which reformulate the problem into a deterministic model. The *Scenario-based Deterministic Equivalent* [20,19] is a single model that expands the stochastic model for each scenario and stage. All stochastic parameters and each second (and higher-)stage variable is copied for each scenario, as well as all constraints involving higher stage parameters or variables. *Policy-based Search* [24] treats stochastic parameters as decision variables and uses *and-or-search* to explore all possible scenarios. Finally, *Progressive Hedging* [15] solves each scenario to optimality in isolation, and then iteratively adapts the objective function to minimise the gap between the first stage variables.

3 Stochastic MINIZINC

The design of Stochastic MINIZINC follows four objectives. (1) The stochastic extension is *conservative*, the stochastic model can be run, debugged, and solved deterministically, without changing the model. (2) The model is *agnostic of the solving approach*, so that the user does not have to commit to a specific stochastic approach during modelling. (3) Basic knowledge of stochastic optimisation should suffice to formulate a stochastic problem. (4) The stochastic extensions are lightweight additions to the language. As a result, Stochastic MINIZINC is a simple extension of standard MINIZINC that includes stochastic annotations.

A stochastic MINIZINC problem specification has three parts: a core problem model, a deterministic and a stochastic data specification. The core model is a standard MINIZINC model, augmented with annotations to mark stochastic parameters and stages. The deterministic data defines deterministic, first stage parameters. The stochastic data is given as a list of deterministic data files, one per scenario. Note that the core model combined with the deterministic data and a single scenario is a valid, deterministic MiniZinc model for that scenario. From the scenario list and a list of scenario weights, a combined stochastic data specification can be generated. Alternatively, the combined representation can be specified directly.

3.1 Stochastic Annotations in MINIZINC

The annotation `::stage(n)` associates a variable or parameter with stage n. A parameter in stage 1 is known from the outset and not stochastic. Variables and parameters without stage annotations belong to the first stage. The objective function is annotated to identify how the probabilistic nature of the scenarios is aggregated. We introduce an annotation `::expected` to optimise the *expected* value over all scenarios. The annotation `::scenario_weights` identifies the weights that reflect the likeliness of each scenario. The scenario weights have to be given as an array of the same length as the number of scenarios.

3.2 An Example: The stochastic Vehicle Routing Problem

We illustrate Stochastic MINIZINC by formulating a stochastic variant of the vehicle routing problem (VRP) [9]. In the deterministic VRP, the aim is to find tours for m vehicles to serve n customers, minimising travel costs. In the stochastic variant of the VRP, the travel times are uncertain, and the aim is to find a vehicle-customer assignment that minimises the expected travel times. This means that the vehicle-customer assignments are the first stage decisions, and the optimal tours for each vehicle are the second stage decisions.

Fig. 1 shows a stochastic VRP model based on the classical VRP formulation [9], omitting the redundant predecessor variables for brevity. In line 13 we annotate the stochastic parameter, and in lines 16-18 we annotate the stochastic variables with their stages. The objective is annotated with `expected` since we want to find the optimal solution wrt. the expected arrival times of each vehicle. Note that all parameters, variables, constraints and the objective are defined deterministically, and the model can thus be solved as such. For instance, the data sets `d1.dzn` and `d2.dzn` in the bottom left each correspond to a single scenario.

The stochastic data `d_stoch.dzn` is generated from the scenarios `d1.dzn` and `d2.dzn` using the specification in the bottom left. Each parameter has an added dimension for the scenario. The `distance` is now three-dimensional, the first dimension indexing the scenario. The transformations in the next section link the stochastic parameter back to the model using the scenario as an index.

4 Transformations

This section shows how Stochastic MINIZINC can be implemented by transformation into standard MINIZINC. We consider three different formulations: a scenario-based deterministic model, a policy-based search, and a version of progressive hedging. Since the transformations generate standard MINIZINC, all solvers that support MINIZINC can be used. For policy-based search and progressive hedging, the backend solvers need to support search combinators [18]. The results of transforming the VRP from Fig. 1 can be found at [14].

We only present the two-stage version of the transformations, multi-stage problems are a straightforward generalisation [17]. In addition to the first and second stage sets of decision variables V_1 and V_2, we use C_1 and C_2 for the sets of first and second stage constraints, p for the set of stochastic parameters, and o for the original objective function. The transformations rely on a substitution operation $substitute(S, [x_1/y_1, \ldots, x_n/y_n])$, meaning that all occurrences of each x_i in S are simultaneously replaced by y_i.

4.1 The Scenario-based Deterministic Equivalent

A Stochastic MINIZINC model is transformed into the deterministic equivalent by creating a copy of the second stage variables, with the stochastic parameters substituted by the concrete values for the scenario.

The objective from the stochastic model needs to be modified to represent the expected value over all scenarios. We introduce an array of variables `o` for the contribution of each scenario to the overall objective. The expected value is then computed as

```
% ============== Stochastic Vehicle Routing Problem ============= %
include "globals.mzn";
include "stochastic.mzn";

int: nC; int: nV; int: timeBudget;
set of int: VEHICLE = 1..nV;
set of int: CUSTOMER = 1..nC;
set of int: NODES = 1..nC+2*nV;
set of int: START_DEPOT_NODES = nC+1..nC+nV;
set of int: END_DEPOT_NODES = nC+nV+1..nC+2*nV;
set of int: TIME = 0..timeBudget;
array[NODES] of int: serviceTime;
array[NODES, NODES] of int: distance  :: stage(2);

% -------- variables ------------- %
array[NODES] of var VEHICLE: vehicle  :: stage(1);
array[NODES] of var NODES: successor  :: stage(2);
array[NODES] of var TIME: arrivalTime :: stage(2);

% -------- first stage constraints ---------- %
constraint forall (n in START_DEPOT_NODES)       % associate each start
   ( vehicle[n] = n-nC );                        % node with a vehicle
constraint  forall (n in END_DEPOT_NODES)        % associate each end
   ( vehicle[n] = n-nC-nV );                     % node with a vehicle

% -------- second stage constraints ---------- %
constraint forall (n in nC+nV+1..nC+2*nV-1)      % successors of end nodes
   ( successor[n] = n-nV+1 );                    % are start nodes
constraint successor[nC+2*nV] = nC+1;

constraint forall (n in START_DEPOT_NODES)       % vehicles leave the
   ( arrivalTime[n] = 0 );                       % depot at time zero
constraint circuit(successor);                   % hamiltonian circuit

constraint forall (n in CUSTOMER)                % use the same vehicle
   ( vehicle[successor[n]] = vehicle[n] );       % along a subtour
constraint forall (n in 0..nC+nV)
   ( arrivalTime[n] + serviceTime[n] + distance[n,successor[n]]
     <= arrivalTime[successor[n]] );             % time constraints

% -------- objective ------------ %
solve minimize   % expected overall travel time of each vehicle
  (sum (n in END_DEPOT_NODES) (arrivalTime[n])) :: expected;
```

```
% ====================== deterministic data ========================== %
nV = 1;  nC = 3;  timeBudget = 30;
serviceTime = [2,2,2,0,0];
```

```
% ==== d1.dzn ====== %
distance = [| 0, 4, 3, 5, 5
            | 4, 0, 2, 3, 3
            | 3, 2, 0, 2, 2
            | 5, 3, 2, 0, 0
            | 5, 3, 2, 0, 0 |];
```

```
% ==== d2.dzn ====== %
distance = [| 0, 4, 3, 5, 5,
            | 4, 0, 2, 6, 3,
            | 3, 2, 0, 2, 2,
            | 5, 6, 2, 0, 0,
            | 5, 3, 2, 0, 0  |];
```

```
% === implicit stochastic data == %
array[1..2] of string: scenarios =
     ["d1.dzn","d2.dzn"];
array[1..2] of int: weights = [1,1];
```

```
% === d_stoch.dzn === %
distance = array3d(1..2, % scenarios
                   1..5, 1..5,
        [ 0, 4, 3, 5, 5, % scenario 1
          4, 0, 2, 3, 3,
          3, 2, 0, 2, 2,
          5, 3, 2, 0, 0,
          5, 3, 2, 0, 0,

          0, 4, 3, 5, 5, % scenario 2
          4, 0, 2, 6, 3,
          3, 2, 0, 2, 2,
          5, 6, 2, 0, 0,
          5, 3, 2, 0, 0  ]);
array[1..2] of int: weights = [1,1]
             :: scenario_weights;
```

Fig. 1. A stochastic vehicle routing problem

the weighted average using the array of weights `w`. We use an integer representation for simplicity, but if the target solver supports continuous variables, a version using `float` variables could be used.

```
function var int: expected(array[int] of int: w, array[int] of var int: o) =
  sum (i in index_set(o)) (w[i]*o[i]) div sum(w);
```

```
V_1;
C_1;
array[1..S] of var int: o;
constraint forall (s in 1..S) ( let {
     substitute(V_2, [p/p[s], o/o[s]]);
} in substitute(C_2, [p/p[s], o/o[s]]) );
solve minimize expected(w,o);
```

The deterministic equivalent is then constructed by looping over all scenarios and creating a fresh set of second stage variables for each scenario using a `let` construct. The second stage constraints C_2 are moved into the loop. For both these code sections we add a scenario argument s to each second stage parameter.

4.2 Policy-based Search

Policy-based search for stochastic constraint programming [24] turns stochastic parameters into decision variables and then uses backtracking search to explore the different scenarios. Instead of copying the second stage model for each scenario as in Sect. 4.1, policy-based search implements the `forall` over all scenarios using a variant of *and-or search*. Decision variables are searched in the usual *or*-fashion, while stochastic variables represent *and*-nodes.

Our implementation adds decision variables for each stochastic parameter `p`, and a single integer variable `scenario` that selects the scenario. Element constraints connect the parameter variables to the actual parameters for the selected scenario. The original objective `o` is unchanged, but an additional expected objective `eo` is added as in Sect. 4.1.

```
V_1;
C_1;
array[1..S] of var int: os;
var 1..S: scenario;
for each array[1..S] of int: p add var int: pV = p[scenario];
substitute(V_2, [ p/pV ]);
substitute(C_2, [ p/pV ]);
var int: eo = expected(w,os);
solve two_stage(eo,o,os);
```

The *and-or* search can be implemented elegantly using search combinators [18], an expressive domain-specific language for sophisticated search strategies. The combinator `two_stage` used above can be realised as follows:

```
combinator s_bab(svar int: i, array[int] of svar int: best, var int: o) =
  post(scenario=i /\ o<best[i], and(search_stage_2,assign(best[i],o)));
combinator two_stage(var int: eo, var int: o, array[int] of var int: os) =
  bab(eo,
      let { array[1..S] of svar int: best = [ ∞ | i in 1..S] } in
      and([ search_stage_1,
            for (i,1,S, s_bab(i,best,o)),
            post(os = best) ]));
```

The main structure of the combinator is an outer branch-and-bound search (`bab` in line 4) over the expected objective, combined with an inner branch-and-bound for every scenario (`scenario_bab`, line 7). The optimum of the second stage for each scenario is collected in an array `best`, and after all scenarios have been processed, is posted back into the variables `os`, constraining the expected objective. The search strategies `search_stage_1` and `search_stage_2` can be user-defined or default searches for the first and second stage variables, respectively.

4.3 Progressive Hedging

Progressive hedging [15] solves each scenario to optimality in isolation, producing different assignments to the first stage variables for each scenario. The objective of each scenario is then augmented with a term to minimise these first stage differences between scenarios. This is iterated until the differences between first stage variables across all scenarios are sufficiently small.

The transformation of a stochastic model using progressive hedging adds weights `xw` that will be updated after each iteration, and an augmented objective `o_hedge` that accounts for the differences between the first stage variables:

```
1  V1;
2  C1;
3  var 1..S: scenario;
4  array[1..|V1|] of int: xw;
5  for each array[1..S] of int: p add var int: pV = p[scenario];
6  substitute(V2,[ p/pV ]);
7  substitute(C2,[ p/pV ]);
8  var int: o_hedge = o + hedge(xw,V1);
9  solve progressive_hedging(o_hedge);
```

```
1  combinator progressive_hedging(var int: o) =
2    restart( distance > epsilon,
3      let { array[1..S,1..|V1|] of svar int: V } in
4      or(for (i,1,S,
5              let { svar int: best = ∞ } in
6              post(scenario=i /\ o < best,
7                and([search_stage_1, search_stage_2, assign(best,o),
8                     assign(V[i],V1)]))
9         ),
10        update_weights(distance,xw,V)
11   ) );
```

The `progressive_hedging` combinator iterates over all scenarios (line 4), performing a branch-and-bound search on the modified objective `o_hedge` similar to `scenario_bab`. The results of the first stage variables V_1 are stored in the array `V` (line 8). After search in all scenarios has completed, the distance between the V_1 variables and the weights in the extended objective function are updated (line 10). The `update_weights` function requires some integration with the underlying solver, since it changes the parameters of an existing constraint. We are currently implementing this feature.

5 Related Work

The closest related approach is Stochastic OPL [25], a proposed extension of the OPL modelling language [23] with support for stochastic variables, chance constraints, and an objective function based on expectation. Similar to Stochastic MINIZINC, the extended language is compiled into the deterministic equivalent in standard OPL. Our

approach is more general, with translations into policy-based search and progressive hedging using search combinators. We also took a conservative approach to language extension, where stochastic features are represented using annotations, while the basic model is perfectly valid deterministic MINIZINC. Finally, using MINIZINC as the base language yields a solver agnostic approach, enabling the modeller to experiment with all the different backend solvers that support MINIZINC.

AIMMS [16] is a commercial modelling and solving framework, where models can be formulated using a graphical user-interface or by employing the internal programming language. It provides strong support for stochastic linear, continuous problems, but has very limited support for non-linear integer problems.

AMPL is a commercial algebraic modelling language with a number of stochastic extensions [5,6,3,20,22]. The SAMPL [22] extension, including the Stochastic Programming Integrated Environment (SPiNE) [21], has been integrated into AMPL. It has annotations for stochastic parameters and other stochastic features. However, the modeller has to formulate the deterministic (scenario-based) equivalent and thus has to commit to this approach.

GAMS is a commercial modelling and solving framework that incorporates a high-level modelling language with a stochastic extension [10]. Stochastic models contain annotations that associate parameters to random distributions and assign variables and constraints to stages. The annotation compilation must be explicitly stated by the modeller, and non-linear constraints are not supported.

Lingo [1] is a commercial optimisation framework for Microsoft Excel with support for stochastic programming. It provides a modelling language and strong support for stochastic linear problems, however, integer or non-linear problems can only be translated into their scenario-based deterministic equivalents.

PySP [27] is an open modelling and solving library based on Pyomo [8], an algebraic modelling language extending Python. PySP supports stochastic integer problems that can be solved either as scenario-based deterministic equivalents or by progressive hedging. It has no known support for non-linear constraints.

6 Conclusions

We have presented Stochastic MINIZINC, an extension of the MINIZINC modelling language that introduces syntax for modelling stochastic constraint (optimisation) problems. We have shown how to translate Stochastic MINIZINC automatically into standard MINIZINC, using three different standard approaches for dealing with uncertainty: the scenario-based deterministic equivalent, policy-based search, and progressive hedging. The resulting system enables modellers to express stochastic problems at a high level of abstraction, and to experiment with different solving approaches. Stochastic MINIZINC is the first solving technology agnostic approach to stochastic modelling we are aware of.

The presented stochastic extensions of MINIZINC are implemented, as well as the automated transformations that take fractions of a second to translate. We plan to release Stochastic MINIZINC soon.

References

1. Atlihan, M., Cunningham, K., Laude, G., Schrage, L.: Challenges in adding a stochastic programming/scenario planning capability to a general purpose optimization modeling system. In: Sodhi, M.S., Tang, C.S. (eds.) A Long View of Research and Practice in Operations Research and Management Science, International Series in Operations Research & Management Science, vol. 148, pp. 117–135. Springer US (2010)
2. Flener, P., Pearson, J., Ågren, M.: Introducing ESRA, a Relational Language for Modelling Combinatorial Problems. In: Rossi, F. (ed.) CP. LNCS, vol. 2833, p. 971. Springer (2003)
3. Fourer, R., Lopes, L.: StAMPL: A Filtration-Oriented Modeling Tool for Multistage Stochastic Recourse Problems. INFORMS Journal on Computing 21(2), 242–256 (2009)
4. Frisch, A.M., Harvey, W., Jefferson, C., Hernandez, B.M., Miguel, I.: Essence: A Constraint Language for Specifying Combinatorial Problems. Constraints 13(3), 268–306 (2008)
5. Gassmann, H., Ireland, A.: Scenario formulation in an algebraic modelling language. Annals of Operations Research 59, 45–75 (1995)
6. Gassmann, H., Ireland, A.: On the formulation of stochastic linear programs using algebraic modelling languages. Annuals of Operations Research 64, 83–112 (1996), stochastic programming, algorithms and models (Lillehammer, 1994)
7. Gent, I.P., Miguel, I., Rendl, A.: Tailoring Solver-Independent Constraint Models: A Case Study with Essence' and Minion. In: Miguel, I., Ruml, W. (eds.) SARA. Lecture Notes in Computer Science, vol. 4612, pp. 184–199. Springer (2007)
8. Hart, W.E., Watson, J.P., Woodruff, D.L.: Pyomo: modeling and solving mathematical programs in Python. Math. Program. Comput. 3(3), 219–260 (2011)
9. Kilby, P., Shaw, P.: Vehicle routing. In: Rossi, F., Beek, P.v., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 23, pp. 799–834. Elsevier Science Inc., New York, NY, USA (2006)
10. Loewe, M., Ferris, M.: Stochastic programming (SP) with EMP (GAMS) (2013), http://www.gams.com/dd/docs/solvers/empsp.pdf
11. Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P.J., de la Banda, M.G., Wallace, M.: The design of the zinc modelling language. Constraints 13(3), 229–267 (2008)
12. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) CP'07. LNCS, vol. 4741, pp. 529–543. Springer (2007)
13. NICTA: MinZinc (2014), http://www.minizinc.org
14. NICTA: Stochastic MiniZinc examples (2014), http://www.minzinc.org/stochastic/
15. Rockafellar, R.T., Wets, R.J.B.: Scenarios and policy aggregation in optimization under uncertainty. Mathematics of Operations Research 16(1), 119–147 (1991)
16. Roelofs, M., Bisschop, J.: AIMMS: The language reference 3.9 (2009)
17. Ruszczyński, A., Shapiro, A.: Stochastic Programming. Handbooks in operations research and management science, Elsevier (2003)
18. Schrijvers, T., Tack, G., Wuille, P., Samulowitz, H., Stuckey, P.J.: Search combinators. Constraints 18(2), 269–305 (2013)
19. Tarim, A., Manandhar, S., Walsh, T.: Stochastic Constraint Programming: A Scenario-Based Approach. Constraints 11(1), 53–80 (2006)
20. Thénié, J., Delft, C., Vial, J.: Automatic formulation of stochastic programs via an algebraic modeling language. Computational Management Science 4(1), 17–40 (2007)
21. Valente, C., Mitra, G., Poojari, C.: A Stochastic Programming Integrated Environement (SPiNE), pp. 115–136. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM). Philadelphia, PA: MPS, Mathematical Programming Society (2005)

22. Valente, C., Mitra, G., Sadki, M., Fourer, R.: Extending algebraic modelling languages for stochastic programming. INFORMS Journal on Computing 21(1), 107–122 (2009)
23. Van Hentenryck, P.: The OPL Optimization Programming Language. MIT Press, Cambridge, MA, USA (1999)
24. Walsh, T.: Stochastic Constraint Programming. In: van Harmelen, F. (ed.) ECAI. pp. 111–115. IOS Press (2002)
25. Walsh, T.: Stochastic OPL. Proceedings of the Workshop on Modelling and Solving with Constraints (2002)
26. Walsh, T.: personal communication (2014)
27. Watson, J.P., Woodruff, D.L., Hart, W.E.: PySP: modeling and solving stochastic programs in Python. Math. Program. Comput. 4(2), 109–149 (2012)