Stochastic OPL

Toby Walsh
Cork Constraint Computation Centre
Cork
Ireland
tw@4c.ucc.ie

1st May 2002

Abstract

To model combinatorial decision problems involving uncertainty and probability, we have proposed "stochastic constraint programming" [3]. This extends constraint programming with stochastic variables, chance constraints and optimized expectations. We propose extending the OPL modelling language [1] with these features, and show how they can be compiled away using some simple rules.

1 Introduction

Many decision problems contain uncertainty. Data about events in the past may not be known exactly due to errors in measuring or difficulties in sampling, whilst data about events in the future may simply not be known with certainty. For example, when scheduling power stations, we need to cope with uncertainty in future energy demands. As a second example, nurse rostering in an accident and emergency department requires us to anticipate variability in workload. As a final example, when constructing a balanced bond portfolio, we must deal with uncertainty in the future price of bonds. To deal with such situations, we have proposed an extension of constraint programming called *stochastic constraint programming* [3] in which we distinguish between decision variables, which we are free to set, and stochastic (or observed) variables, which follow some probability distribution. In this paper, we describe how we are extending the OPL constraint modelling language [1] to deal with stochastic variables.

2 Stochastic constraint programs

We have defined a number of models of stochastic constraint programming of increasing complexity. In an one stage stochastic constraint satisfaction problem (stochastic CSP), the decision variables are set before the stochastic variables. The stochastic variables independently take values with probabilities given by a probability distribution. This models situations where we act now and observe later. For example, we have to decide now which nurses to have on duty and will only later discover the actual work-load. We can easily invert the instantiation order if the application demands, with the stochastic variables set before the decision variables. Constraints are defined (as in traditional constraint satisfaction) by relations of allowed tuples of values. Constraints can, however, be implemented with specialized and efficient algorithms for consistency checking. In addition, we allow for both hard constraints (which are always satisfied) and "chance constraints" (which may only be satisfied in some of the possilbe worlds). Each chance constraint has a threshold, θ and must be satisfied in this or more fraction of the possible worlds. A one stage stochastic CSP is satisfiable iff there exists values for the decision variables so that, given random values for the stochastic variables, the hard constraints are always satisfied and the chance constraints are satisfied in at least the given fraction of worlds. Note that in [**?**], we only allowed for one (global) chance constraint so the definition here of stochastic constraint programming is strictly more general.

In a two stage stochastic CSP, there are two sets of decision variables, V_{d1} and V_{d2}, and two sets of stochastic variables, V_{s1} and V_{s2}. The aim is to find values for the variables in V_{d1}, so that given random values for V_{s1}, we can find values for V_{d2}, so that given random values for V_{s2}, the hard constraints are always satisfied and the chance constraints are again satisfied in at least the given fraction of worlds. Note that the values chosen for the second set of decision variables V_{d2} are conditioned on both the values chosen for the first set of decision variables V_{d1} and on the random values given to the first set of stochastic variables V_{s1}. This can model situations in which items are produced and can be consumed or put in stock for later consumption. Future production then depends both on previous production (earlier decision variables) and on previous demand (earlier stochastic variables). A m stage stochastic CSP is defined in an analogous way to one and two stage stochastic CSPs.

A stochastic constraint optimization problem (stochastic COP) is a stochastic CSP plus a cost function defined over the decision and stochastic variables. The aim is to find a solution that satisfies the stochastic CSP which minimizes (or, if desired, maximizes) the expected value of the cost function. In the future, we may also consider other possibilities (e.g. minimizing the maximal value, or maximizing the minimal value).

3 Scenario based stochastic CP

In [3], we give a semantics for stochastic constraint programs based on policies. A policy is a tree of decisions. Each path in a policy represents a different possible world (set of values for the stochastic variables), and the values assigned to decision variables in this world. To find satisfying policies, [3] presents backtracking and forward checking algorithms for finding policies which explores the implicit AND/OR graph. Stochastic variables give AND nodes (as we must find a policy that satisfies all their values) whilst decision variables give OR nodes (as we only need find one satisfying value). An alternative semantics (and solution method), which was first proposed by Armagan Tarim, comes from a scenario based view of stochastic constraint programs. A scenario is any possible set of values for the stochastic variables. Thus, a scenario is associated with each path in the policy. Within each scenario, we have a conventional (non-stochastic) constraint program to solve. We simply replace the stochastic variables by the values taken in the scenario, and ensure that the values found for the decision variables are consistent across scenarios.

4 Stochastic OPL

To implement stochastic constraint programming, we propose extending OPL with stochastic variables, chance constraints and optimized expectations. The extension is in the spirit of ESRA [2] since the extended OPL models can be compiled down into (regular) OPL models with relative ease.

An OPL model consists of two parts: a set of declarations, followed by an instruction. Declarations define the data types, (input) data and the variables. An OPL instruction is either to satisfy a set of constraints or to maximize/minimize an expression subject to a set of constraints. We will extend OPL's declarations to include stochastic variables, and a timeline giving the order in which stochastic and decision variables are set. In addition, we will extend

4.1 Stochastic variables

Unlike decision variables, which the user sets, stochastic variables are set by nature according to some probability distribution. A stochastic constraint program can have multiple stages, with the user allowed to set later decision variables according to the values taken by earlier stochastic variables. Stochastic variables are defined using a command of the form:

```
stochastic var <Type> <Id> <Dist>;
```

Where `<Type>` is (as with decision variables) a data type (e.g. a range of values, or an enumerated list of values), `<Id>` is (as with decision variables) the variable name with an (optional) array range, and `<Dist>` defines the probability distribution of the stochastic variable(s). Probability distributions include `uniform`, `poisson(lambda)`, and user defined via a list of (not necessarily normalized) values. Other types of distribution can be supported as needed. Here are some examples:

```
stochastic var 0..1 market[Years] uniform;
stochastic var int incomingCalls poisson(100);
stochastic var 100..105 demand {1,2,3,3,2,1};
```

In the first, we have a 0/1 variable which takes either value with equal probability. In the last, we have a demand variable, which takes the value 100 in 1 out of 12 cases, 101 in 2 out 12 cases, ... For simplicity, we will assume that stochastic variables take values independently of each other. However, it would not be hard to extend the language of stochastic OPL to drop this assumption. A timeline is then defined over the stochastic and decision variables with the command:

```
<Timeline> = start <SeqIds> end;
```

Where

```
<SeqIds> = <Id> |
           forall(<Var> in <Range) <Timeline> |
           <Id> <SeqIds> |
           forall(<Var> in <Range) <Timeline> <SeqIds>
```

For example, in the financial planning example, we might have:

```
start
  forall(i in Years) start stocks[i] bonds[i] market[i] end;
end;
```

This ensures that the decision variable `stocks[i]` comes earlier in the timeline than the decision variable `bonds[i]`, and that this comes before the stochastic variable `market[i]`. In addition, the timeline ensures that the stochastic variable `market[i]` comes before the decision variable `stocks[i+1]` (and hence also before the stochastic variable `market[i+1]`).

4.2 Chance constraints

One way in which stochastic variables are used is in "chance constraints". These are constraints that hold in some of the possible worlds. Chance constraints are defined using a command of the form:

```
prob(<Constraint>) <ArithOp> <Expr>;
```

Where `<Constraint>` is any OPL expression, `<ArithOp>` is any of the arithmetically comparison operations (=,<,>, <=, or >=) and `<Expr>` is any arithmetic expression (it may contain decision variables or may just be a rational or a float in the range 0 to 1). For example, the following command specifies the chance constraint that in each quarter the stochastic variable, demand does not exceed the decision variable, production plus the stock carried forward in each quarter with 96% probability:

```
forall(i in 0..n)
   prob(demand[i] <= production[i]+stock[i]) >= 0.96;
```

Constraints which are not chance constraints are hard and have to hold in possible worlds. For example, the stock carried forwards (which is modelled as a decision variable even though it is not set by the user) is computed via the hard constraint:

```
forall(i in 1..n)
   stock[i] = stock[i-1] + production[i] - demand[i];
```

4.3 Optimized expectations

If the objective function being maximized or minimized includes stochastic variables or decision variables which are dependent on stochastic variables, we now interpret this as maximizing or minimizing the expectation of the function. For example, in the book production example of [3], we want to minize the expected cost of storing surplus books. Each book costs £1 per quarter to store. This can be specified by the following model:

```
minimize
  expected cost
subject to
  cost = sum(i in 0..n) max(stock[i],0);
  forall(i in 1..n)
    stock[i] = stock[i-1] + production[i] - demand[i];
```

In addition to expectations, we can consider other extensions (e.g. we may try to minimize the maximum cost, or maximize the minimum profit).

5 An example

Let's put all this together. We will consider a financial planning example from the stochastic programming literature. Briefly, we have to divide our investment portfolio each of 3 years between stocks and bonds. The market goes up or down with equal probability. If the market goes up, bonds return 100% profit but stocks return 300%. However, if the market goes down, bonds return no profit (though we keep the principal) and stocks loose all their value. Our goal is to turn £5 into £25. The utility of returning more than £25 is the surplus times a constant, α. The utility of returning less than £25 is the deficit times a negative constant, β. This problem is defined by the following model:

```
int n = 3;
int Alpha = ..;
int Beta = ..;
int StockUp = 3;
int StockDown = -1;
int BondUp = 1;
int BondDown = 0;
range Years 0..n-1;

var int bonds[Years];
var int stocks[Years];
var int wealth[0..n];
stochastic var 0..1 market[Years] uniform;
start
  forall(i in Years)
    start wealth[i] bonds[i] stocks[i] market[i] end;
  wealth[n]
end;

maximize
```

```
  expected Alpha*max(0,wealth(n)-25) + Beta*min(0,wealth(n)-25)
subject to
  wealth(0) = 5;
  forall(i in Years)
    wealth(i) = bonds(i)+stocks(i);
  forall(i in 1..n)
    wealth(i) = (1+StockUp)*stocks(i-1)*market(i-1) +
                (1+BondUp)*bonds(i-1)*market(i-1) +
                (1+StockDown)*stocks(i-1)*(1-market(i-1)) +
                (1+BondDown)*bonds(i-1)*(1-market(i-1));
```

6 Compiling stochastic OPL

Stochastic OPL models can be compiled down to (regular) OPL models without much difficulty by exploiting the scenario based interpretation of stochastic constraint programs.

6.1 Stochastic variables

From the declarations of the stochastic variables and the timeline, we can compute the possible scenarios and their probabilities. We then introduce new decision variables for each of the possible scenarios. To do this we need to introduce ragged arrays into OPL. Again, these can easily be compiled out, perhaps using records. As an example, in the financial planning example, the `bonds` array can be compiled down into the following declarations:

```
int Scenarios[0..2] = [1..1,1..2,1..4]]
var int bonds[Years,Scenarios];
```

Note that `bonds` is now a ragged array, with elements: `bonds[0,1]`, `bonds[1,1]`, `bonds[1,2]`, `bonds[2,1]`, `bonds[2,2]`, `bonds[2,3]`, and `bonds[2,4]`.

6.2 Hard constraints

Hard constraints need to be expanded out in terms of the different scenarios. For example, consider the hard constraints:

```
forall(i in Years)
 wealth(i) = bonds(i)+stocks(i);
```

This is compiled down into:

```
 forall(i in Years)
   forall(j in Scenarios[i])
     wealth(i,j) = bonds(i,j)+stocks(i,j);
```

6.3 Chance constraints

We can rewrite chance constraints as reified sum constraints. For example, consider the chance constraint from the production planning example:

```
  forall(i in 0..n)
    prob(demand[i] <= production[i]+stock[i]) >= 0.96;
```

This becomes:

```
  forall(i in 0..n)
    sum(j in Scenarios[i])
      (Demand[i,j] <= production[i,j]+stock[i,j])*P[i,j] >= 0.96
```

Where `P[i,j]` is the probability of the `j`th scenario from `Scenarios[i]`. And `Demand[i,j]` is no longer a variable but an array of data, giving the value taken by the stochastic variable `Demand[i]` in the `j`th scenario. Thus,

```
 int Demand[Years,Scenarios] = [[100,101,102,103,104,105],
                                [100,101,102,103,104,105],..];
```

Note that stochastic variables are completely eliminated by this means. They are eliminated in a similar fashion from hard constraints.

6.4 Optimized expectations

Objective functions over stochastic variables or decision variables which are dependent on stochastic variables are replaced by their expectation in a similar way. For example, in the financially planning example, we have the objective:

```
 maximize
   expected Alpha*max(0,wealth(n)-25) +
            Beta*min(0,wealth(n)-25)
 subject to
   ..
```

This is replaced by:

```
 maximize
   sum(j in Scenarios[n])
     P[n,j] * (Alpha*max(0,wealth(n,j)-25) +
               Beta*min(0,wealth(n,j)-25))
 subject to
   ..
```

Where `P[n,j]` is again the probability of the `j`th scenario from `Scenarios[n]`.

7 Conclusions

To model combinatorial decision problems involving uncertainty and probability, we have proposed "stochastic constraint programming" [3]. This extends constraint programming with stochastic variables, chance constraints and optimized expectations. We have proposed extending the OPL modelling language [1] with these features. We have shown how they can be compiled away using some simple rules. We are currently implementing stochastic OPL as a compiler to OPL. Future work includes other language extensions (e.g. optimization functions like maximizing the minimum of some objective function, and dependencies between stochastic variables), and compilation optimizations (e.g. efficiencies in the size of the compiled OPL model).

Acknowledgements

The author is currently supported by Science Foundation Ireland. He thanks the other members of the 4C lab, and members of the APES research group especially Armagan Tarim. Support for this research also comes from EPSRC under GR/R30792.

References

[1] P. Van Hentenryck, L. Michel, L. Perron, and J-C. Regin., 'Constraint programming in OPL', in *Principles and Practice of Declarative Programming*, ed., G. Nadathur, pp. 97–116. Springer-Verlag, (1999). Lecture Notes in Computer Science 1702.

[2] B. Hnich and P. Flener, 'High-level reformulation of constraint programs', in *Proceedings of 10th International French Speaking Conference on Logic and Constraint Programming (JFPLC'01)*, ed., P. Codognet, pp. 75–89, (2001).

[3] Toby Walsh, 'Stochastic constraint programming', in *Proceedings of the 15th ECAI*. European Conference on Artificial Intelligence, IOS Press, (2002).