Stochastic Constraint Programming

Toby Walsh
Department of Computer Science
University of York
York
England
tw@cs.york.ac.uk

Abstract

To model decision problems involving uncertainty and probability, we propose stochastic constraint programming. Stochastic constraint programs contain both decision variables (which we can set) and stochastic variables (which follow some probability distribution), and combine together the best features of traditional constraint satisfaction, stochastic integer programming, and stochastic satisfiability. We give a semantics for stochastic constraint programs, and propose a number of complete algorithms and approximation procedures. Using these algorithms, we observe phase transition behavior in stochastic constraint programs. Interestingly, the cost of both optimization and satisfaction peaks in the satisfaction phase boundary. Finally, we discuss a number of extensions of stochastic constraint programming to relax various assumptions like the independence between stochastic variables.

Introduction

Many real world 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 propose an extension of constraint programming called stochastic constraint programming in which we distinguish between decision variables, which we are free to set, and stochastic (or observed) variables, which follow some probability distribution.

Stochastic constraint programs

We define 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 are given values. This can model situations in which we must act now and observe later. For example, we may have to decide now which nurses to have on duty and only later discover the actual workload. We can easily invert the instantiation order if the application demands, with the stochastic variables given values before the decision variables are set.

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. The stochastic variables independently take values with probabilities given by a fixed probability distribution. We discuss later how to relax these assumptions, and how this model compares with related frameworks like mixed constraint satisfaction. 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 probability that all the constraints are satisfied equals or exceeds some threshold probability, \( \theta \). The probabilistic satisfaction of constraints allows us to ignore worlds (values for the stochastic variables) which are too rare to require consideration.

In a two stage stochastic CSP, there are two sets of decision variables, \( V_d^1 \) and \( V_d^2 \), and two sets of stochastic variables, \( V_s^1 \) and \( V_s^2 \). The aim is to find values for the variables in \( V_d^1 \), so that given random values for \( V_d^1 \), we can find values for \( V_d^2 \), so that given random values for \( V_d^2 \), the probability that all the constraints are satisfied equals or exceeds \( \theta \). Note that the values chosen for the second set of decision variables \( V_d^2 \) are conditioned on both the values chosen for the first set of decision variables \( V_d^1 \) and on the random values given to the first set of stochastic variables \( V_s^1 \). 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.
**Production planning example**

The following $m$ stage stochastic constraint program models a simple $m$ quarter production planning problem. In each quarter, there is an equal chance that we will sell anywhere between 100 and 105 copies of a book. To keep customers happy, we want to satisfy demand in all $m$ quarters with 80% probability. At the start of each quarter, we must decide how many books to print for that quarter. This problem can be modelling by a $m$ stage stochastic CSP. There are $m$ decision variables, $x_i$ representing production in each of the $i$th quarter. There are also $m$ stochastic variables, $y_i$ representing demand in the $i$th quarter. These takes values between 100 and 105 with equal probability. There is a constraint to ensure first quarter production meets first quarter demand:

$$x_1 \geq y_1$$

There is also a constraint to ensure second quarter production meets second quarter demand either plus any unsatisfied demand from the first quarter or less any stock carried forward from the first quarter:

$$x_2 \geq y_2 + (y_1 - x_1)$$

And there is a constraint to ensure $j$th quarter production ($j \geq 2$) meets $j$th quarter demand either plus any unsatisfied demand from earlier quarters or less any stock carried forward from earlier quarters:

$$x_j \geq y_j + \sum_{i=1}^{j-1} (y_i - x_i)$$

We must satisfy these $m$ constraints with a threshold probability $\theta = 0.8$. This stochastic CSP has a number of solutions including $x_i = 105$ for each $i$ (i.e. always produce as many books as the maximum demand). However, this solution will tend to produce books surplus to demand which is undesirable.

Suppose storing surplus book costs $1 per quarter. We can define a $m$ stage stochastic COP based on this stochastic CSP in which we additionally minimize the expected cost of storing surplus books. As the number of surplus books in the $j$th quarter is $\min(\sum_{i=1}^j x_i - y_i, 0)$, we have a cost function over all quarters of:

$$\sum_{j=1}^m \min(\sum_{i=1}^j x_i - y_i, 0)$$

**Semantics**

A stochastic constraint satisfaction problem is a 6-tuple $\langle V, S, D, P, C, \theta \rangle$ where $V$ is a list of variables, $S$ is the subset of $V$ which are stochastic variables, $D$ is a mapping from $V$ to domains, $P$ is a mapping from $S$ to probability distributions for the domains, $C$ is a set of constraints over $V$, and $\theta$ is a threshold probability in the interval $[0, 1]$. Constraints are defined (as in traditional constraint satisfaction) by a set of variables and a relation giving the allowed tuples of values for these variables. Variables are set in the order in which they appear in $V$. Thus, in an one stage stochastic CSP, $V$ contains first all the decision variables and then all the stochastic variables. In a two stage stochastic CSP, $V$ contains the first set of decision variables, the first set of stochastic variables, then second set of decision variables, and finally the second set of stochastic variables.

A policy is a tree with nodes labelled with variables, starting with the first variable in $V$ labelling the root, and ending with the last variable in $V$ labelling the nodes directly above the leaves. Nodes labelled with decision variables have a single child, whilst nodes labelled with stochastic variables have one child for every possible value. Edges in the tree are labelled with values assigned to the variable labelling the node above. Finally, each leaf node is labelled with 1 if the assignment of values to variables along the path to the root satisfies all the constraints, and 0 otherwise. Each leaf node corresponds to a possible world and has an associated probability; if $x_i$ is the $i$th stochastic variable on a path to the root, $d_i$ is the value given to $x_i$ on this path (i.e. the label of the following edge), and $prob(s_i = d_i)$ is the probability that $s_i = d_i$, then the probability of this world is simply $\prod_i prob(s_i = d_i)$. We define the satisfaction of a policy as the sum of the leaf values weighted by their probabilities. A policy satisfies the constraints iff its satisfaction is at least $\theta$. A stochastic CSP is satisfiable iff there exists a policy which satisfies the constraints. When $S = \emptyset$ and $\theta = 1$, this reduces to the traditional definition of constraint satisfaction. The optimal satisfaction of a stochastic CSP is the maximum satisfaction of all policies. For a stochastic COP, the expected value of a policy as sum of the objective valuations of each of the leaf nodes weighted by their probabilities. A policy is optimal if it satisfies the constraints and maximizes (or, if desired, minimizes) the expected value.

Consider again the production planning problem and a two-quarter policy that sets $x_1 = 104$ and if $y_1 > 100$ then $x_2 = y_1 + 1$ else $y_1 = 100$ and $x_2 = 100$. We can represent this policy by the following (partial) tree:

![Production planning example tree](image)

By definition, each of the leaf nodes in this tree is equally probable. There are 6² leaf nodes, of which only 7 are labelled 0. Hence, this policy's satisfaction is $(36 - 7)/36$, and the policy satisfies the constraints as this just exceeds $\theta = 0.8$.

**Complexity**

Constraint satisfaction is NP-complete in general. Not surprisingly, stochastic constraint satisfaction moves us up the complexity hierarchy. It may therefore be useful for modelling problems like reasoning under uncertainty which lie in these higher complexity classes. We show how a number...
of satisfiability problems in these higher complexity classes reduce to stochastic constraint satisfaction. In each case, the reduction is very immediate. Note that each reduction can be restricted to stochastic CSPs on binary constraints using a hidden variable encoding to map non-binary constraints to binary constraints. The hidden variables are added to the last stage of the stochastic CSP.

PP, or probabilistic polynomial time is characterized by the PP-complete problem, Majisat which decides if at least half the assignments to a set of Boolean variables satisfy a given clausal formula. This can be reduced to a one stage stochastic CSP in which there are no decision variables, the stochastic variables are Boolean, the constraints are the clauses, the two truth values for each stochastic variable are equally likely and the threshold probability \( \theta = 0.5 \). A number of other reasoning problems like plan evaluation in probabilistic domains are PP-complete.

NP<sub>PP</sub> is the class of problems that can be solved by non-deterministic guessing a solution in polynomial time (NP) and then verifying this in probabilistic polynomial time (PP). Given a clausal formula, E-Majisat is the problem of deciding if there exists an assignment for a set of Boolean variables so that, given randomized choices of values for the other variables, the formula is satisfiable with probability at least equal to some threshold \( \theta \). This can be reduced very immediately to an one stage stochastic CSP. A number of other reasoning problems like finding optimal size-bounded plans in uncertain domains are NP<sub>PP</sub>-complete.

PSPACE is the class of problems that can be solved in polynomial space. Note that NP \( \subseteq \text{PP} \subseteq \text{NP}^{\text{PP}} \subseteq \text{PSPACE} \). SSAT, or stochastic satisfiability is an example of a PSPACE-complete problem. In SSAT, we are given a clausal formula with \( m \) alternating decision and stochastic variables, and must decide if the formula is satisfiable with probability at least equal to some threshold \( \theta \). This can be immediately reduced to a \( m \) stage stochastic CSP. A number of other reasoning problems like propositional STRIPS planning are PSPACE-complete.

Procedure BT(\( i, \theta_1, \theta_h \))

\[
\begin{align*}
& \text{if } i > n \text{ then return 1} \\
& \theta := 0 \\
& q := 1 \\
& \text{for each } d_j \in D(x_i) \\
& \text{if } x_i \in S \text{ then} \\
& \quad p := \text{prob}(x_i \rightarrow d_j) \\
& \quad q := q - p \\
& \quad \text{if consistent}(x_i \rightarrow d_j) \text{ then} \\
& \quad \quad \theta := \theta + p \times BT(i + 1, \theta_1 + p \times (\theta_1 - \theta)), \theta_h - \theta) \\
& \quad \quad \text{if } \theta > \theta_h \text{ then return } \theta \\
& \quad \quad \text{if } \theta + q < \theta_1 \text{ then return } \theta \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{if consistent}(x_i \rightarrow d_j) \text{ then} \\
& \quad \quad \quad \quad \theta := \text{max}(\theta, BT(i + 1, \text{max}(\theta, \theta_1), \theta_h))) \\
& \quad \quad \quad \text{if } \theta > \theta_h \text{ then return } \theta \\
& \text{return } \theta
\end{align*}
\]

Figure 1: The backtracking (BT) algorithm for stochastic CSPs. The algorithm is called with the search depth, \( i \) and with upper and lower bounds, \( \theta_h \) and \( \theta_1 \). If the optimal satisfaction lies between these bounds, BT returns the exact satisfaction. If the optimal satisfaction is \( \theta_h \) or more, BT returns a value greater than or equal to \( \theta_h \). If the optimal satisfaction is \( \theta_1 \) or less, BT returns a value less than or equal to \( \theta_1 \). \( S \) is the set of stochastic variables.

As in the Davis-Putnam like algorithm for stochastic satisfiability (Littman, Majercik, & Pitassi 2000), upper and lower bounds, \( \theta_h \) and \( \theta_1 \) are used to prune search. By setting \( \theta_1 = \theta_h = \theta \), we can determine if the optimal satisfaction is at least \( \theta \). Alternatively, by setting \( \theta_1 = 0 \) and \( \theta_h = 1 \), we can determine the optimal satisfaction. The calculation of upper and lower bounds in recursive calls requires some explanation. Suppose that the current assignment to a stochastic variable returns a satisfaction of \( \theta_0 \). We can safely ignore other values for this stochastic variable if \( \theta + p \times \theta_0 \geq \theta_h \). That is, if \( \theta_0 \geq \frac{\theta_h - \theta}{p} \). This gives the upper bound in the recursive call to BT on a stochastic variable. Alternatively, we cannot hope to satisfy the constraints adequately if \( \theta + p \times \theta_0 + q < \theta_1 \) as \( q \) is the maximum that the remaining values can contribute to the satisfaction. That is, if \( \theta_0 \leq \frac{\theta_1 - \theta - q}{p} \). This gives the lower bound in the recursive call to BT on a stochastic variable. Finally, suppose that the current assignment to a decision variable returns a satisfaction of \( \theta \). If this is more that \( \theta_1 \), then any other values must better \( \theta \) to be part of a better policy. Hence, we can replace the lower bound in the recursive call to BT on a decision variable by \( \text{max}(\theta, \theta_1) \). Note that value ordering heuristics may reduce search. For decision variables, we should choose values that are likely to return the optimal satisfaction. For stochastic variables, we should choose values that are more likely.

Forward checking

The Forward Checking (FC) procedure is based on the BT algorithm. On instantiating a decision or stochastic variable,
the FC algorithm checks forward and prunes values from the domains of future decision and stochastic variables which break constraints. Checking forwards fails if a stochastic or decision variable has a domain wipeout, or if a stochastic variable has so many values removed that we cannot hope to satisfy the constraints. As in the regular forward checking algorithm, we use a 2-dimensional array, \( \text{prune}(i,j) \), to record the depth at which the value \( d_j \) for the variable \( x_i \) is removed by forward checking. This is used to restore values on backtracking. In addition, each stochastic variable, \( z_i \) has an upper bound, \( q_i \) on the probability that the values left in its domain can contribute to a solution. When forward checking removes some value, \( d_j \) from \( x_l \), we reduce \( q_i \) by \( \text{prob}(x_i \rightarrow d_j) \), the probability that \( x_i \) takes the value \( d_j \). This reduction of \( q_i \) is undone on backtracking. If forward checking ever reduces \( q_i \) to less than \( \theta_i \), we can immediately backtrack as it is now impossible to set \( x_i \) and satisfy the constraints adequately.

**Phase transition behavior**

As in other search problems (Cheeseman, Kanefsky, & Taylor 1991), stochastic constraint programs display phase transition behavior. To investigate such behavior, we have developed a model of random stochastic constraint programs with four parameters: the number of variables \( n \), the domain size \( m \), the constraint density \( p_1 \), and the constraint tightness \( p_2 \). For simplicity, we assume that decision and stochastic variables strictly alternate, and that each of the \( m \) values for a stochastic variable are equally likely. It would be interesting to relax these assumptions.

Binary constraints are generated between two decision variables or one decision and one stochastic variable. As with the usual models of random CSPs, \( p_1 \) and \( p_2 \) can be either probabilities or fractions. For example, we can construct a fraction \( p_1 \) of the possible binary constraints. Alternatively, we can add each possible binary constraint with probability \( p_1 \). Similarly, when generating conflict matrices, either we generate a fixed fraction of \( p_2 m^2 \) nogoods or we add each of the \( m^2 \) possible nogoods with probability \( p_2 \). To prevent trivially insoluble problems, we recommend “flawless” conflict matrices are generated so that each value has at least one support (MacIntyre et al. 1998).

In Figure 3, we plot the average optimal satisfaction as the constraint density and tightness is varied. As in traditional constraint satisfaction, we observe a ridge along which there is a rapid change in the satisfaction of problems. As might be expected, there is a complexity peak in the cost to solve problems along this ridge. In Figure 4, we plot the search cost to find the optimal satisfaction. Although this is an optimization and not a decision problem as often studied in phase transition experiments, we observe a complexity peak along the ridge. The search cost for the decision problem of determining if a policy exists to meet a given fixed threshold \( \theta \) displays a similar (but slightly lower) complexity peak. The runtime (as well as nodes visited) for FC dominates BT at least a factor of 2 on all but the easiest problems.

---

**Figure 2:** The forward checking (FC) algorithm for stochastic CSPs. The algorithm is called with the search depth, \( i \) and with upper and lower bounds, \( \theta_h \) and \( \theta_l \). \( S \) is the set of stochastic variables. The array \( q_i \) is an upper bound on the probability that the stochastic variable \( x_i \) satisfies the constraints and is initially set to 1, whilst \( \text{prune}(i,d) \) is the depth at which the value \( d \) is pruned from \( x_i \) and is initially set to 0 which indicates that the value is not yet pruned.
Approximation procedures

There are a number of methods for approximating the answer to a stochastic constraint program. For example, we can replace the stochastic variables in a stochastic CSP by their most probable values (or in ordered domains like integers by their median or integer mean values), and then solve (or approximate the answer to) the resulting traditional constraint satisfaction problem. Similarly, we can estimate the optimal solution for a stochastic COP by replacing the stochastic variables by their most probable values and then finding (or approximating the answer to) the resulting traditional constraint optimization problem. We can also use Monte Carlo sampling to test a subset of the possible worlds. For example, we can randomly generate values for the stochastic variables according to their probability distribution. It would also be interesting to develop local search procedures like GSAT and WalkSAT which explore the "policy space" of stochastic constraint programs.

Related work

Stochastic constraint programming is inspired by both stochastic integer programming and stochastic satisfiability (Littman, Majercik, & Pitassi 2000). It shares the advantages that constraint programming has over integer programming (e.g. global constraints, non-linear constraints, and constraint propagation). It also shares the advantages that constraint programming has over satisfiability (e.g. global constraints, and arithmetic constraints). Optimization is likely to play a fundamental role in stochastic constraint programming. This adds to the distinction with stochastic satisfiability as the latter purely concerns decision problems.

Mixed constraint satisfaction (Fargier, Lang, & Schiex 1996) is closely related to one stage stochastic constraint satisfaction. In a mixed CSP, the decision variables are set after the stochastic variables are given random values. In addition, the random values are chosen uniformly. In the case of full observability, the aim is to find conditional values for the decision variables so that we satisfy all possible worlds. In the case of no observability, the aim is to find values for the decision variables so that we satisfy as many possible worlds. An earlier constraint satisfaction model for decision making under uncertainty (Fargier et al. 1995) also included a probability distribution over the space of possible worlds.

Constraint satisfaction has been extended to include prob-
probabilistic preferences on the values assigned to variables (Shazeer, Littman, & Keim 1999). Associated with the values for each variable is a probability distribution. A “best” solution to the constraint satisfaction problem is then found. This may be the maximum probability solution (which satisfies the constraints and is most probable), or the maximum expected overlap solution (which is most like the true solution). The latter can be viewed as the solution which has the maximum expected overlap with one generated at random using the probability distribution. The maximum expected overlap solution could be found by solving a suitable one stage stochastic constraint optimization problem.

Branching constraint satisfaction (Fowler & Brown 2000) models problems in which there is uncertainty in the number of variables. For example, we can model a nurse rostering problem by assigning shifts to nurses. Branching constraint satisfaction then allows us to deal with the uncertainty in which nurses are available for duty. We can represent such problems with a stochastic CSP with a stochastic 0/1 variable for each nurse representing their availability.

A number of extensions of the traditional constraint satisfaction problem model constraints that are uncertain, probabilistic or not necessarily satisfied. For example, in partial constraint satisfaction we maximize the number of constraints satisfied (Freuder & Wallace 1992). As a second example, in probabilistic constraint satisfaction each constraint has a certain probability independent of all other probabilities of being part of the problem (Fargier & Lang 1993). As a third example, both valued and semi-ring based constraint satisfaction (Bistarelli et al. 1996) generalizes probabilistic constraint satisfaction as well as a number of other frameworks. In semi-ring based constraint satisfaction, a value is associated with each tuple in a constraint, whilst in valued constraint satisfaction, a value is associated with each constraint. However, none of these extensions deal with variables that may have uncertain or probabilistic values. Indeed, stochastic constraint programming can easily be combined with most of these techniques. For example, we can define stochastic partial constraint satisfaction in which we maximize the number of satisfied constraints, or stochastic probabilistic constraint satisfaction in which each constraint has an associated probability of being in the problem.

Stochastic constraint programs are closely related to Markov decision problems (MDPs). These have been very influential in AI of late for dealing with situations involving reasoning under uncertainty (Kaelbling, Littman, & Cassandra 1998). Stochastic constraint programming could be used to model problems which lack the Markov property that the next state and reward depend only on the previous state and action taken. Stochastic constraint optimization could also be used to model more complex reward functions than the (discounted) sum of individual rewards. On the other hand, MDPs can, at least in theory, be solved efficiently using linear programming (Littman, Dean, & Kaelbling 1995) and can have infinite state spaces.

**Conclusions**

We have proposed stochastic constraint programming, an extension of constraint programming to deal with both decision variables (which we can set) and stochastic variables (which follow some probability distribution). This framework is designed to take advantage of the best features of traditional constraint satisfaction, stochastic integer programming, and stochastic satisfiability. It can be used to model a wide variety of decision problems involving uncertainty and probability. We have given a semantics for stochastic constraint programs based upon policies. These determine how decision variables are set depending on earlier decision and stochastic variables. We have proposed a number of complete algorithms and approximation procedures for stochastic constraint programming. Using these algorithms, we have observed phase transition behavior in stochastic constraint programs. Interestingly, the cost of both optimization and satisfaction peaks along the satisfaction phase boundary. Finally, we have discussed a number of extensions of stochastic constraint programming to relax assumptions like the independence between stochastic variables.

**Acknowledgements**

The author is an EPSRC advanced research fellow. He thanks the other members of the APES research group (http://apes.cs.strath.ac.uk/), especially Ian Gent for his helpful discussions.

**References**


135