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, θ. 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_{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 probability that all the constraints are satisfied equals or exceeds θ. 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.

Production planning example

The following m stage stochastic constraint program models a simple m quarter production planning problem. In each quarter, there is a 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 ith quarter. There are also m stochastic variables, y_i representing demand in the ith 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 a also 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 jth quarter production ($j \geq 2$) meets jth 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 miminize the expected cost of storing surplus books. As the number of surplus books in the jth quarter is $\min(\sum_{i=1}^{j} x_i - y_i, 0)$, we have a cost function over all quarters of:

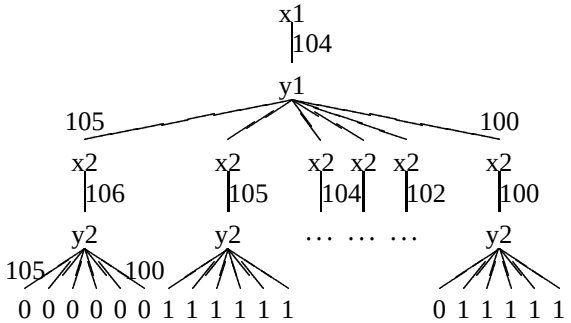
$$\sum_{j=1}^{m} \min(\sum_{i=1}^{j} x_i - y_i, 0)$$

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 varibles, 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 θ 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 s_i is the ith stochastic variable on a path to the root, d_i is the value given to s_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 θ. A stochastic CSP is **satisfiable** iff there exists a policy which satisfies the constraints. When $S = \{\}$ 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:



By definition, each of the leaf nodes in this tree is equally probable. There are 6^2 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, MAJSAT 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^{PP} is the class of problems that can be solved by nondeterministic guessing a solution in polynomial time (NP) and then verifying this in probabilistic polynomial time (PP). Given a clausal formula, E-MAJSAT 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 θ. 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^{PP}-complete.

PSPACE is the class of problems that can be solved in polynomial space. Note that NP $\subseteq$ PP $\subseteq$ NP^{PP} $\subseteq$ 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 θ. This can be immediately reduced to a m stage stochastic CSP. A number of other reasoning problems like propositional STRIPS planning are PSPACE-complete.

Complete algorithms

We present a backtracking algorithm, which is then extended to a forward checking procedure.

Backtracking

We assume that variables are instantiated in order. However, if decision variables occur together, we can choose in which order to instantiate them. A branching heuristic like fail first may therefore be useful to order decision variables which occur together. On meeting a decision variable, the backtracking (BT) algorithm tries each value in its domain in turn. The maximum value found is returned to the previous recursive call. On meeting a stochastic variable, the BT algorithm tries each value in turn, and returns the sum of the all answers to the subproblems weighted by the probabilities of their occurrence. At any time, if instantiating a decision or stochastic variable breaks a constraint, we return 0. If we manage to instantiate all the variables without breaking any constraint, we return 1. The algorithm can be trivially adapted to record the optimal policy.

procedure BT(i,θ_l,θ_h)
  if $i > n$ **then** return 1
  $\theta := 0$
  $q := 1$
  for each $d_j \in D(x_i)$
    if $x_i \in S$ **then**
      $p :=$ prob($x_i \rightarrow d_j$)
      $q := q - p$
      if $consistent(x_i \rightarrow d_j)$ **then**
        $\theta := \theta + p \times$ BT($i+1$,$\frac{\theta_l - \theta - q}{p}$,$\frac{\theta_h - \theta}{p}$)
        if $\theta > \theta_h$ **then** return θ
      if $\theta + q < \theta_l$ **then** return θ
    else
      if $consistent(x_i \rightarrow d_j)$ **then**
        $\theta :=$ max(θ,BT($i+1$,max(θ, θ_l),θ_h))
        if $\theta > \theta_h$ **then** return θ
  return θ

Figure 1: The backtracking (BT) algorithm for stochastic CSPs. The algorithm is called with the search depth, i and with upper and lower bounds, θ_h and θ_l. If the optimal satisfaction lies between these bounds, BT returns the exact satisfaction. If the optimal satisfaction is θ_h or more, BT returns a value greater than or equal to θ_h. If the optimal satisfaction is θ_l or less, BT returns a value less than or equal to θ_l. 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, θ_h and θ_l are used to prune search. By setting $\theta_l = \theta_h = \theta$, we can determine if the optimal satisfaction is at least θ. Alternatively, by setting $\theta_l = 0$ 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 θ_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 $\theta + p \times \theta_0 + q \leq \theta_l$ as q is the maximum that the remaining values can contribute to the satisfaction. That is, if $\theta_0 \leq \frac{\theta_l - \theta - q}{p}$. 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 θ. If this is more that θ_l, then any other values must better θ 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 $\max(\theta, \theta_l)$. 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 an 2-dimensional array, $prune(i, j)$ 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, x_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_i, we reduce q_i by $prob(x_i \rightarrow d_j)$, the probability that x_i takes the value d_j. This reduction of q_j is undone on backtracking. If forward checking ever reduces q_i to less than θ_l, 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 θ 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.

```
procedure FC(i,θ_l,θ_h)
  if i > n then return 1
  θ := 0
  for each d_j ∈ D(x_i)
    if prune(i,j) = 0 then
      if check(x_i → d_j,θ_l) then
        if x_i ∈ S then
          p := prob(x_i → d_j)
          q_i := q_i − p
          θ := θ + p× FC(i + 1,(θ_l − θ − q_i)/p,(θ_h − θ)/p)
          restore(i)
          if θ + q_i < θ_l then return θ
          if θ > θ_h then return θ
        else
          θ := max(θ,FC(i + 1,max(θ,θ_l),θ_h))
          restore(i)
          if θ > θ_h then return θ
      else restore(i)
  return θ

procedure check(x_i → d_j,θ_l)
  for k = i + 1 to n
    dwo := true
    for d_l ∈ D(x_k)
      if prune(k,l) = 0 then
        if inconsistent(x_i → d_j,x_k → d_l) then
          prune(k,l) := i
          if x_k ∈ S then
            q_k := q_k− prob(x_k → d_l)
            if q_k < θ_l then return false
        else dwo := false
    if dwo then return false
  return true

procedure restore(i)
  for j = i + 1 to n
    for d_k ∈ D(x_j)
      if prune(j,k) = i then
        prune(j,k) = 0
        if x_j ∈ S then q_j := q_j+ prob(x_j → d_k)
```

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, θ_h and θ_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 $prune(i, d)$ 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.

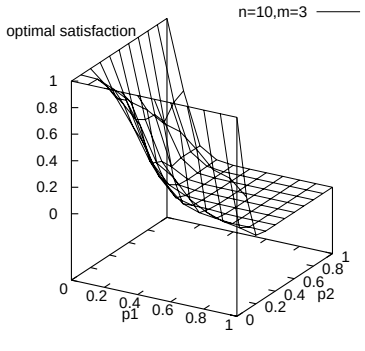


Figure 3: Average optimal satisfaction (z-axis) against constraint density (x-axis) and tightness (y-axis). Problems have 10 variables, each with 3 values. 100 problems are generated at each value of p_1 and p_2 from 0 to 1 in steps of $\frac{1}{10}$.

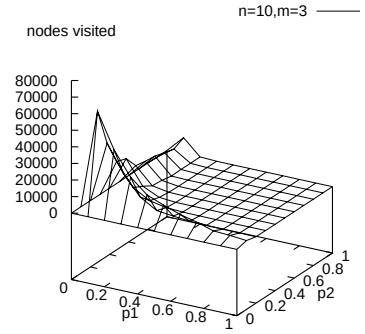


Figure 4: Mean nodes visited by FC computing the optimal satisfaction (z-axis) against constraint density (x-axis) and tightness (y-axis).

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.

Extensions

We have assumed that stochastic variables are independent. There are problems which may require us to relax this restriction. For example, a stochastic variable representing electricity demand may depend on a stochastic variable representing temperature. It may therefore be useful to combine stochastic programming with techniques like Bayes networks which allow for conditional dependencies to be efficiently and effectively represented. An alternative solution is to replace the dependent stochastic variables by a single stochastic variable whose domain is the product space of the dependent variables. This is only feasible when there are a small number of dependent variables with small domains.

We have also assumed that the probability distribution of stochastic variables is fixed. In particular, we have assumed that it does not depend on earlier decision variables. Again, there are problems which may require us to relax this restriction. For example, the decision variable representing price may influence a stochastic variable representing demand. A solution may again be to combine stochastic programming with techniques like Bayes networks.

We have also assumed that all variable domains are finite. There are problems which may require us to relax this restriction. For example, in scheduling power stations, we may use 0/1 decision variables to model whether a power station runs or not, but have continuous (observed) variables to model future electricity demands. A continuous probability density function could be associated with these variables. Similarly, a continuous decision variable could be useful to model the power output of the power stations that we decide to operate. Interval reasoning techniques could be extended to deal with such variables.

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-

abilistic 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 & Wallacs 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

Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; and Verfaillie, G. 1996. Semi-ring based CSPs and valued CSPs: Basic properties and comparison. In Jample, M.; Freuder, E.; and Maher, M., eds., *Over-Constrained Systems*, 111–150. Springer-Verlag. LNCS 1106.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. 1991. Where the really hard problems are. In *Proc. of the 12th IJCAI*, 331–337. Int. Joint Conference on Artificial Intelligence.

Fargier, H., and Lang, J. 1993. Uncertainty in constraint satisfaction problems: a probabilistic approac h. In *Proc. of ECSQARU*. Springer-Verlag. LNCS 747.

Fargier, H.; Lang, J.; Martin-Clouaire, R.; and Schiex, T. 1995. A constraint satisfaction framework for decision under uncertainty. In *Proc. of the 11th Int. Conference on Uncertainty in Artificial Intelligence*.

Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed constraint satisfaction: a framework for decision problems under in complete information. In *Proc. of the 13th National Conference on Artificial Intelligence*.

Fowler, D., and Brown, K. 2000. Branching constraint satisfaction problems for solutions robust under likely changes. In *Proc. of 6th Int. Conference on Principles and Practices of Constraint Programming*. Springer-Verlag.

Freuder, E., and Wallacs, R. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21–70.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1-2):99–134.

Littman, M.; Dean, T.; and Kaelbling, L. 1995. On the complexity of solving Markov decision problems. In *Proc. of the 8th Annual Conference on Uncertainty in AI*.

Littman, M.; Majercik, S.; and Pitassi, T. 2000. Stochastic Boolean satisfiability. *Journal of Automated Reasoning*.

MacIntyre, E.; Prosser, P.; Smith, B.; and Walsh, T. 1998. Random constraint satisfaction: Theory meets practice. In *4th Int. Conference on Principles and Practices of Constraint Programming (CP-98)*, 325–339. Springer.

Shazeer, N.; Littman, M.; and Keim, G. 1999. Constraint satisfaction with probabilistic preferences on variable values. In *Proc. of the 16th National Conference on Artificial Intelligence*.