Stochastic Constraint Programming

Toby Walsh[1]

Abstract. To model combinatorial decision problems involving uncertainty and probability, we introduce stochastic constraint programming. Stochastic constraint programs contain both decision variables (which we can set) and stochastic variables (which follow a probability distribution). They 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. Finally, we discuss a number of extensions of stochastic constraint programming to relax various assumptions like the independence between stochastic variables, and compare with other approaches for decision making under uncertainty.

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 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.

2 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. 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 workload. 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. The stochastic variables independently take values with probabilities given by a probability distribution. We discuss later how to relax these assumptions, and how this compares with other frameworks. 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 a threshold θ. The probabilistic satisfaction of constraints allows us to ignore worlds (values for the stochastic variables) which are rare. Note that the definition reduces to that of a traditional constraint satisfaction problem if we have no stochastic variables and $\theta = 1$.

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.

3 Production planning example

The following stochastic constraint program models a simple m quarter production planning problem. In each quarter, we will sell 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 decide how many books to print for that quarter. This problem is modelled by a m stage stochastic CSP. There are m decision variables, x_i representing production in each quarter. There are also m stochastic variables, y_i representing demand in each quarter. These take values between 100 and 105 with equal probability. There is a constraint to ensure 1st quarter production meets 1st quarter demand:

$$x_1 \geq y_1$$

There is also a constraint to ensure 2nd quarter production meets 2nd quarter demand plus any unsatisfied demand or less any stock:

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

And there is a constraint to ensure jth quarter production ($j \geq 2$) meets jth quarter demand plus any unsatisfied demand or less any

[1] Cork Constraint Computation Centre, University College Cork, Ireland. email: tw@4c.ucc.ie. The author thanks the other members of the APES research group, especially Ian Gent for his helpful discussions. This research was supported by an EPSRC advanced research fellowship.

stock:

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

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:

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

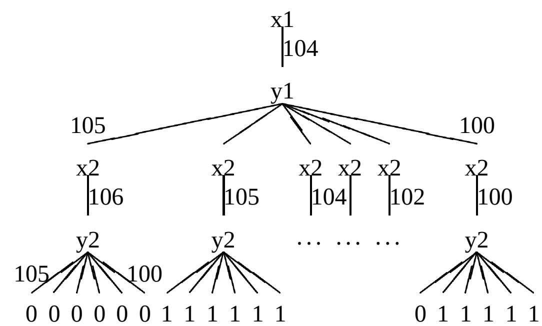
Note that a solution to a stochastic CSP or COP defines how to set later decision variables given the values for earlier stochastic and decision variables.

4 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 by a set of variables and a relation giving the allowed tuples of values. Variables are set in the order in which they appear in V. Thus, in an one stage stochastic CSP, V contains the decision variables and then the stochastic variables. In a two stage stochastic CSP, V contains the first set of decision variables, the first set of stochastic variables, then the 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. Leaf nodes are 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 is a policy which satisfies the constraints. The **optimal satisfaction** of a stochastic CSP is the maximum satisfaction of all policies. For a stochastic COP, the **expected value** of a policy is the sum of the objective valuations of each leaf node 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 $\theta = 0.8$.

5 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 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 the stochastic variables 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 non-deterministic 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 [LGM98]. 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 have 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.

6 Complete algorithms

We present a backtracking algorithm for solving stochastic CSPs, which is then extended to a forward checking procedure.

6.1 Backtracking

We assume that variables are instantiated in order. However, if decision variables occur together, they can be instantiated in any order. A branching heuristic like fail first may therefore be used 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 is returned to the previous recursive call. On meeting a stochastic variable, we try 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.

```
procedure BT(i,θ_l,θ_h)
  if i > n then return 1
  θ := 0
  q := 1
  for each d_j ∈ D(x_i)
    if x_i ∈ S then
      p := prob(x_i → d_j)
      q := q − p
      if consistent(x_i → d_j) then
        θ := θ + p× BT(i + 1, (θ_l−θ−q)/p, (θ_h−θ)/p)
        if θ > θ_h then return θ
      if θ + q < θ_l then return θ
    else
      if consistent(x_i → d_j) then
        θ := max(θ,BT(i + 1,max(θ, θ_l),θ_h))
        if θ > θ_h then return θ
  return θ
```

Figure 1. The backtracking (BT) algorithm is called with the search depth, i and with 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.

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 exceed 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 $\max(\theta, \theta_l)$. Because of these bounds, value ordering heuristics can 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.

6.2 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 (dwo), 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 can 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 on q_j is undone on backtracking. If forward checking ever reduces q_i to less than θ_l, we backtrack as it is impossible to set x_i and satisfy the constraints adequately.

```
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 is called with the search depth, i and bounds, θ_h and θ_l. If the maximum satisfaction of all policies lies between these bounds, FC returns the exact maximum satisfaction. If the maximum satisfaction of all policies is θ_h or more, FC returns a value greater than or equal to θ_h. If the minimum satisfaction of all policies is θ_l or less, FC returns a value less than or equal to 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.

7 Experimental evaluation

We implemented the BT and FC algorithms in Common Lisp and ran them on the production planning problem given in Section 3, as well as on a range of randomly generated problems. For the production plannng problem, we use a simple heuristic which orders values for the decision variables by their size. This will tend to keep stock levels low. It will also ensure that the worlds in which we fail to satisfy the demand constraints are those where demand is much higher than average. Results are given in Table 1 with the threshold for satisfiability θ set to 0.8. Similar results are obtained for other non-zero θ. Surprisingly, performance was relatively insensitive to the precise value of θ used.

Number of quarters	BT nodes	BT CPU/sec	FC nodes	FC CPU/sec
1	28	0.01	10	0.01
2	650	0.09	148	0.03
3	17,190	2.72	3,604	0.76
4	510,346	83.81	95,570	19.07
5	15,994,856	3,245.99	2,616,858	509.95

Table 1. Backtracking (BT) and forward checking (FC) algorithms on the production planning problem from Section 3. CPU times are for a Common Lisp implementation running under Linux on an ancient 133MHz Pentium, whilst "nodes" are the number of nodes visited in the and/or search tree.

The performance advantage of the FC algorithm over the BT algorithm increases as the stochastic CSP increases in size. On the largest problem in Table 1, the FC algorithm visits approximately 1/6th the search nodes in roughly 1/6th the CPU time. This is in line with our results on random problems, where the FC algorithm is often an order of magnitude faster than the BT algorithm. Even larger gains can be expected on problems in which constraints apply to variables which are set far apart in the search tree. On such problems, forward checking will prune domains far down the search tree, thereby avoiding deep backtracks.

Our results show that the FC algorithm clearly dominates the BT algorithm. Consistency testing and domain pruning ensures that it only visits a small fraction of the possible worlds. Further performance gains could be obtained by more powerful constraint propagation, more intelligent backtracking, and more sophisticated branching heuristics. These are all areas for future work.

8 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. If the fraction of the resulting constraint satisfaction problems that are satisfiable is at least equal to the threshold θ, then the original stochastic constraint satisfaction problem is likely to be satisfiable. It would also be interesting to develop local search procedures like GSAT and WalkSAT [SLM92, SKC94] which explore the "policy space" of stochastic constraint programs.

9 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, and 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 the probability distribution is known in advance. It would be interesting to explore methods for estimating it based on observation.

Finally, we have 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. Interval reasoning techniques could be extended to deal with such variables.

10 Related work in decision making under uncertainty

Stochastic constraint programs are closely related to Markov decision problems (MDPs). An MDP model consists of a set of states, a set of actions, a state transition function which gives the probability of moving between two states as a result of a given action, and a reward function. A solution to an MDP is a policy, which specifies the best action to take in each possible state. MDPs These have been very influential in AI of late for dealing with situations involving reasoning under uncertainty [Put94]. Stochastic constraint programs can model problems which lack the Markov property that the next state and reward depend only on the previous state and action taken. To represent a stochastic constraint program in which the current decision depends on all earlier decisions would require an MDP with an exponential number of states. Stochastic constraint optimization can also be used to model more complex reward functions than the (discounted) sum of individual rewards.

Stochastic constraint programs are also closely related to influence diagrams. Influence diagrams are Bayesian networks in which the chance nodes are augmented with decision and utility nodes [OS90]. The usual aim is to maximize the sum of the expected utilities. Chance nodes in an influence diagram correspond to stochastic variables in a stochastic constraint program, whilst decision nodes correspond to decision variables. The utility nodes correspond to the cost function in a stochastic constraint optimization problem. It would therefore be relatively straightforward to map stochastic constraint programs into influence diagrams. However, reasoning about stochastic constraint programs is likely to be easier than about influence diagrams. First, the probabilistic aspect of a stochastic constraint program is simple and decomposable as there are only unary marginal probabilities. Second, the dependencies between decision variables and stochastic variables are represented by declarative constraints. We can therefore borrow from traditional constraint satisfaction and optimization powerful algorithmic techniques like branch and bound, constraint propagation and nogood recording. As a result, if a problem can be modelled within the more restricted format of a stochastic constraint program, we hope to be able to reason about it more efficiently.

11 Related work in constraints

Stochastic constraint programming is inspired by both stochastic integer programming and stochastic satisfiability [LMP00]. It shares the advantages that constraint programming has over integer programming (e.g. non-linear constraints, and constraint propagation). It also shares the advantages that constraint programming has over satisfiability (e.g. global and arithmetic constraints, and more compact models).

Mixed constraint satisfaction [FLS96] 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 in a mixed CSP so that we satisfy all possible worlds. In the case of no observability, the aim is to find values for the decision variables in a mixed CSP so that we satisfy as many possible worlds. An earlier constraint satisfaction model for decision making under uncertainty [FLMCS95] also included a probability distribution over the space of possible worlds.

Constraint satisfaction has been extended to include probabilistic preferences on the values assigned to variables [SLK99]. 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 [FB00] 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 [FW92]. 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 [FL93]. As a third example, both valued and semi-ring based constraint satisfaction [BFM+96] 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.

12 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. Finally, we have discussed a number of extensions of stochastic constraint programming to relax assumptions like the independence between stochastic variables, and compared it with other approaches for decision making under uncertainty like Markov decision problems and influence diagrams.

REFERENCES

[BFM+96] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semi-ring based CSPs and valued CSPs: Basic properties and comparison. *Over-Constrained Systems*, pages 111–150. Springer-Verlag, 1996. LNCS 1106.

[FB00] D. Fowler and K. Brown. Branching constraint satisfaction problems for solutions robust under likely changes. In *Proc. of 6th Int. Conf. on Principles and Practices of Constraint Programming*. Springer-Verlag, 2000.

[FL93] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approac h. In *Proc. of ECSQARU*. Springer-Verlag, 1993. LNCS 747.

[FLMCS95] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. A constraint satisfaction framework for decision under uncertainty. In *Proc. of the 11th Int. Conf. on Uncertainty in AI*, 1995.

[FLS96] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: a framework for decision problems under in complete information. In *Proc. of the 13th Nat. Conf. on AI*. 1996.

[FW92] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[LGM98] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic plan existence and evaluation. *J. of AI Research*, 9:1–36, 1998.

[LMP00] M.L. Littman, S.M. Majercik, and T. Pitassi. Stochastic Boolean satisfiability. *J. of Automated Reasoning*, 2000.

[OS90] R.M. Oliver and J.Q. Smith. *Influence Diagrams, Belief Nets and Decision Analysis*. John Wiley and Sons, 1990.

[Put94] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley and Sons, 1994.

[SKC94] B. Selman, H. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proc. of the 12th Nat. Conf. on AI*, pages 337–343. 1994.

[SLK99] N. Shazeer, M. Littman, and G. Keim. Constraint satisfaction with probabilistic preferences on variable values. In *Proc. of the 16th Nat. Conf. on AI*. 1999.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. of the 10th Nat. Conf. on AI*, pages 440–446. 1992.