On the Stochastic Constraint Satisfaction Framework

Lucas Bordeaux
Microsoft Research
Cambridge, UK
lucasb@microsoft.com

Horst Samulowitz
University of Toronto, Dpt of Computer Science
Toronto, Canada
horst@cs.toronto.edu

ABSTRACT

Stochastic constraint satisfaction is a framework that allows to make decisions taking into account possible futures. We study two challenging aspects of this framework: (1) variables in stochastic CSP are ordered sequentially, which is adequate for the representation of a number of problems, but is not a natural choice for the modeling of problems involving branching; (2) the framework was designed to allow multi-objective decision-making, yet this issue has been treated only superficially in the literature. We bring a number of clarifications to these two aspects. In particular, we show how minor modifications allow the framework to deal with non-sequential forms, we identify a number of technicalities related to the use of the sequential ordering of variables and of the use of multiple objectives, and in addition we propose the first search algorithm that solves multi-objective stochastic problems in polynomial space.

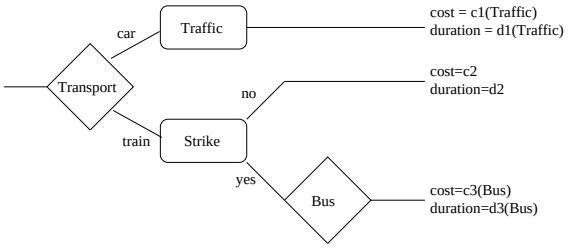
1. INTRODUCTION AND MOTIVATION

Context: Integrating the Future in Decisions

An important and challenging problem in optimisation is to make decisions *in prediction to a future* which, by definition, cannot be forecasted precisely. To make this type of decisions it is necessary to consider the whole range of futures that are possible, to estimate the likelihood of each of these future scenarios, to predict the quality of the decisions *w.r.t.* each future, and to favour the decisions whose quality is likely to be high. Examples of contexts involving this type of prospective reasoning are abundant: a decision whether to launch a new product will have completely different consequences depending on whether the competitor is secretly planning to propose a similar offer; when deciding the quantity of goods to produce we aim at satisfying a future demand which can be estimated only with limited confidence, but a storage cost will be incurred if there is some surplus, and the decision has to take into account this risk in addition to the production cost (*Book Example*, [7]).

SAC'07 March 11-15, 2007, Seoul, Korea

The framework we discuss in this paper is more specifically driven by a class of uncertain decision-making problems that are best understood by considering a preliminary example. In this example we want to make a trip from Paris to Amsterdam. Train and car are the two options, but we are in a period of strikes, and there is an estimated 40% risk that the train never leaves, should we consider this option. In case this happens, the train company would be forced to provide a replacement (*e.g.,* a bus service). This solution is however typically essentially slower. Now if we directly choose to take the car there is also some uncertainty , *i.e.,* the traffic might be low, medium or high. *What should we decide?* In general the train offers the best compromise between speed and price, but today if we choose this solution we have a chance to end-up with an inefficient replacement service (*e.g.,* bus). The example can be described pictorially as follows:



The diamonds represent *decision* variables: a decision variable represents a choice for which we do or will have to make the decision. On the contrary, the ovals represent *stochastic* variables: the values of these variables will not be decided by us but by some external agent, or by "the environment". We assume that we can estimate the probability according to which each value will be chosen. The diagram specifies that we initially have to choose the value for a variable *Transport* ranging over $\{car, train\}$. If (for instance) we choose *car*, then the environment will fix a value for the variable *Traffic* $\in \{low, med, high\}$. The duration and the cost in this case are both functions of the traffic (in the other cases in the Figure the costs are simply constants). While this example is simplistic, it exhibits the following features which are representative of a whole class of applications:

- In evaluating the quality of our decision we have used several criteria: the duration and the cost. This is often the most natural way to state preferences, in which case the problem is called *multi-objective*;
- We have adopted a "branching" (tree-shaped) repre-

sentation of the future, as opposed to a "linear" (sequential) one[1]. By this we mean, more specifically, that some decisions only apply in some branches: for instance variable *Strike* plays a role only if we choose *train*. *If on the contrary we choose car then variable Strike simply does not exist*;

- The leaves of the tree essentially specify values for each of the objective. These values are most likely functions of the variables accumulated along the corresponding branch of the tree (in our simple example the cost and duration depend on the traffic in the case of choosing the car in the first place).

Goal of the Paper

Our goal in this paper is to propose a framework that will allow us to conveniently model problems such as the one described above, *i.e.,* problems in which we make decisions against the future, in which the future involves alternating decision and stochastic variables, and in which we adopt a branching viewpoint on the future. We also investigate the employment of multi-objectives and reveal several technicalities caused by its usage.

We think that the ability to model branching time is important as this branching will typically arise whenever the environment is allowed to make a discrete choice that causes the remaining possibilities for the next steps to be completely different. For instance, suppose that a construction company is making decisions related to its interaction with a particular client. The future is modelled as follows:

> "if the customer accepts the current offer then we will start the construction; otherwise we will propose him an alternative".

The answer of the customer is clearly a stochastic variable, and at the time of the decision the only information we can have on this variable is the probability of the yes/no answers. Depending on this answer, we will have to make decisions on which team of builders should start working or we will have to schedule another round of negotiation with a member of the business team. Clearly these two tasks take place in completely different contexts and therefore do not involve the same decision variables.

Summary of the Contributions

The framework we start off with is *stochastic constraint satisfaction* as introduced by Walsh [7]. This appears like a natural choice, as the distinction between deterministic and stochastic variables is central to this framework. However, this framework is essentially sequential and was not primarily aimed at decisions involving a branching future. The additional core observations can be summarized as follows:

1. The original stochastic CSP framework is sequential, and does not allow any branching. At first this could be considered as a minor drawback: a stochastic CSP can be thought of as a formula in prenex form, while branching time would require a non-prenex form. If known results in the closely related field of quantified constraints were also applicable to stochastic CSP, we could always express branching time using a sequential framework. *We show that this is in general not the case.* In our newly introduced framework we are able to clearly define the non-prenex form and to deal with it in a proper fashion.

2. The original formulation of stochastic CSP allows to model different objectives (each of which is assigned a different threshold in the decision version). But surprisingly, all the complete algorithms that have been proposed for the original framework are only valid for the case of a single objective [2]. As an explanation to this fact *we exhibit a technical issue that makes it difficult to solve the multi-objective framework using search-based approaches.* We show that this issue can be fixed using a new enumeration mechanism. This algorithm provides us with the proof that the multi-objective version of the problem can be solved in polynomial space.

In the subsequent section we present our new stochastic constraint satisfaction framework. In this section we also position the original stochastic constraint satisfaction framework and state the main difference to it. Then we address the previously highlighted topics (1) and (2) in Sections 3 and 4, respectively. A brief discussion and a summary of the results concludes the paper.

2. THE FRAMEWORK

Example

The stochastic constraint satisfaction framework [4, 7] uses "quantifiers" of the form $\exists x$ and $\forall x$ to introduce decision and stochastic variables in sequence. We use this idea and propose to consider a class of *non-prenex* stochastic CSPs. In order to model tree-shaped problem structures we employ the *if-then-else*. For instance the example of Section 1 will be modelled by the following formula:

$$\exists \mathit{Trans}. \left(\begin{array}{l} \text{if } \mathit{Trans} = \mathit{car} \text{ then} \\ \quad \forall \mathit{Traffic}.\langle c_1(\mathit{Traffic}), d_1(\mathit{Traffic})\rangle \\ \text{else} \\ \quad \forall \mathit{Strike}. \left(\begin{array}{l} \text{if } \mathit{Strike}{=}\mathit{no} \text{ then} \\ \quad \langle c_2, d_2\rangle \\ \text{else} \\ \quad \exists \mathit{Bus}.\langle c_3(\mathit{Bus}), d_3(\mathit{Bus})\rangle \end{array} \right) \end{array} \right)$$

We note that all the branches of this expression end in a vector of the form $\langle a, b\rangle$; this is the notation used to express the quality of this branch *w.r.t.* each of the 2 criteria. The reader might be surprised to see an existential quantifier appearing in the *else* branch of a condition whose *then* branch is a vector. This is not a typing mistake that would make the example ill-defined: such existential formulae should not be

[1] The branching/linear terminology we are using is borrowed from the Temporal Logics literature, in which there have been well-known debates on the respective advantages and limitations of these two viewpoints on time, see, *e.g.,* [6].

[2] The algorithms we are mentioning are the search algorithms in [7, 1]. In both of these papers the possibility of using different objectives (or different thresholds) is initially mentioned, but the algorithms are then presented for the case where only one threshold is present. The approach proposed by [5], on the contrary, seems to be able to deal with multiple objectives. Our understanding is nevertheless that this approach requires exponential space in order to guarantee completeness.

understood as a Boolean condition, the branch starting by this internal decision can be evaluated to a vector of costs.

We also note that, although the framework is entirely based on the evaluation of *functions*, this is expressive enough to represent *constraints*, which are seen as functions returning values in $\{0, 1\}$. Depending on the threshold she associates to the constraint, the user is allowed to either impose it strictly or ask to simply maximise its satisfaction.

Syntax

Formulae are built over a vocabulary S of stochastic variables, and a vocabulary I of internal decision variables. Each variable x has a finite domain D_x. If a variable y is stochastic, then we are given, for each value $v \in D_y$, the probability $p_y(v)$ that y takes value v (note that $\sum_{v \in D_y} p_y(v) = 1$). A stochastic formula (*Form*) over m objectives is an expression written according to the following grammar:

The language allowed for terms can vary, for instance, a simple language is based on linear expressions of the following syntax:

A formula is *closed* if in every atom, every occurrence of a stochastic or decision variable falls under the scope of a quantifier. A formula is well-defined if it is closed and if no variable appears under the scope of two different quantifiers.

Semantics

Because this is rarely done in the literature and because this raises some technicalities we specify the semantics of the formulae of our framework. This semantic is based on the notion of *strategy* (*a.k.a.* policy). A strategy completely specifies the choices made for the decision variables as *functions* of the stochastic variables that are chronologically assigned before it. Because of the non-prenex form, strategies have to be defined in a slightly non-standard way, as follows.

The relation "*variable* x *precedes variable* y *in formula* Ψ", noted $x \prec_\Psi y$, is true if y falls under the scope of x. Formally, if $\Psi \equiv \exists z.\phi$ or $\Psi \equiv \forall z.\phi$, then $x \prec_\Psi y$ holds if $x = z$ and y has an occurrence in ϕ or if $x \prec_\phi y$. If $\Psi \equiv$ **if** *Cond* **then** A **else** B then $x \prec_\Psi y$ iff $x \prec_A y$ or $x \prec_B y$.

Let $Pred(x)$ denote the set of *stochastic* variables that precede x in the considered formula. Then a strategy s defines, for each decision variable $x \in I$, a function s_x of signature $(\prod_{y \in Pred(x)} D_y) \to D_x$ (where $\prod$ denotes Cartesian product).

We can determine the vector of *expected values* of a formula ϕ with respect to a strategy s. This evaluation function, $eval_s$, takes as parameters the vector $\langle I_1..I_p \rangle$ of values accumulated for the preceding *decision* variables during the exploration of the formula, and similarly the vector $\langle S_1..S_q \rangle$ of values accumulated for the preceding *stochastic* variables. The rules are the following:

- $eval_s(\langle I_1..I_p \rangle, \langle S_1..S_q \rangle, \exists x.\phi) \equiv$
  $eval_s(\langle I_1..I_p, I_{p+1} \rangle, \langle S_1..S_q \rangle, \phi)$ where $I_{p+1} = s\langle S_1..S_q \rangle$
- $eval_s(\langle I_1..I_p \rangle, \langle S_1..S_q \rangle, \forall y.\phi) \equiv$
  $\sum_{v \in D_y} p_y(v) \cdot eval_s(\langle I_1..I_p \rangle, \langle S_1..S_q, v \rangle, \phi)$

Note that for the case $\forall x.\phi$, the multiplication is between a real value (probability) and a vector returned by the evaluation; this corresponds to a weighted sum on each objective. In the base case (evaluation of an atom, *i.e.,* vector of terms), the evaluation simply returns the vector obtained by computing each term. The conditional is evaluated as one would expect.

Several computational problems can be considered:

satisfaction- the user provides a threshold for each of the objectives (for instance constraints, whose satisfaction will be imposed a lower bound); The problem is to determine whether a strategy exists whose evaluation on each vector satisfies the threshold;

optimisation- the user provides thresholds for each of the objectives but one, and the goal will be to compute the strategy that satisfies all the threshold and whose value is optimal *w.r.t.* to the last objective.

Features of the Framework

In summary the two key features of the framework are that it allows to express non-prenex formulas, and to deal with multiple thresholds. Prenex formulae are essentially equivalent to the classical (sequential) stochastic CSP framework; the special case of formulae involving only one objective function will also be of special interest, as we will show that is it better-behaved in some aspects. We call this special case *single-objective*, as opposed to the general SCSP framework defined by Walsh which is *multi-objective*.

3. DEALING WITH BRANCHING TIME

To allow to naturally encode the tree-shaped structure of applications such as the example of Section 1, we have proposed to use a prenex form and conditional expressions. Let us first note that, while the idea of non-prenex form appears to be natural, classical non-prenex form (in the sense ofnon-prenex quantified logical formulae) does not directly apply in our context: consider the following stochastic SAT [3] instance $\forall x.\ \forall y.\ (x \wedge y)$. An attempt to put this formula into non-prenex form (*e.g.,* $\forall x.x \wedge \forall\, y.y)$) would be incorrect, as stochastic formulas evaluate to real numbers, not Booleans.

Prenex Form

A legitimate question is whether systematic means exist to put a arbitrary formula into an equivalent formula in prenex form. Putting a formula into prenex form means that we extract quantifiers that appear as subterms of the formula and bring them outside of the formula. In our case the syntax allows to nest quantifiers only within one type of constructs, namely the *if-then-else*, for instance:

$$\textit{if cond then } \exists\, x.A \textit{ else } B$$

And similarly with a quantifier $\forall$. (We may also consider the case where the quantified formula appears in the *else* branch, but this case is clearly symmetric.) The question is therefore whether the following rewritings are correct:

$$\textit{if cond then } \exists\, x.A \textit{ else } B \quad \equiv \quad \exists x.\textit{if cond then } A \textit{ else } B$$

$$\textit{if cond then } \forall\, x.A \textit{ else } B \quad \equiv \quad \forall x.\textit{if cond then } A \textit{ else } B$$

As it turns out, this question is dependent on whether we consider a multi-objective framework or not.

The Single-Objective Case

In case we have a single-objective problem, we can benefit from an important simplification: because the quantity to

optimise (say, minimise) is a unique value instead of a vector, we will always obtain the best chance to obtain a winning strategy if we take the Min of all possible values whenever we meet a decision variable. For this reason it is easy to see that the equalities justifying the transition to prenex form are correct, *e.g.,* the minimum value of a formula of the form:

$$\textit{if } cond \textit{ then } \exists x.A \textit{ else } B$$

will be $\mathsf{Min}_{v \in D_x} eval(A[x := v])$ if *cond* is true and $eval(B)$ otherwise. In both cases we have the same value for:

$$\exists x.\textit{if } cond \textit{ then } A \textit{ else } B$$

(The other case is similar.)

The Multi-Objective Case

The multi-objective case is unfortunately much more complex, and indeed the transformation into prenex form has unpredictable effects. To see this, consider the formula:

$$\forall x. \left(\begin{array}{l} \textit{if } x = 0 \textit{ then} \\ \quad \exists a.\ \langle a, 1-a \rangle \\ \textit{else} \\ \quad \forall y.\ \langle 0.5, 0.5 \rangle \end{array} \right)$$

where all variables have domain $\{0, 1\}$, and the probabilities are $p_x(0) = p_x(1) = p_y(0) = p_y(1) = 1/2$. This formula has two objectives. The constraint is to find a strategy that would assign an expected value of at least $1/2$ to each of the components.

We first observe that the formula in its original form does not have a strategy that satisfies the given thresholds. A strategy for this formula is a function deciding the value assigned to a depending on x. Due to the condition, the evaluation of a is only applied under the setting $x = 0$, and under this fix setting of x there is only one choice for a. Consequently, we will always average the vector $\langle 0.5, 0.5 \rangle$ (*else* branch) with either $\langle 0, 1 \rangle$ or $\langle 1, 0 \rangle$ which in both cases obviously violates the given thresholds.

Now by putting the formula into prenex form, we can obtain the following:

$$\forall x.\forall y.\exists a.\ \left(\textit{ if } x = 0 \textit{ then } \langle a, 1-a \rangle \textit{ else } \langle 0.5, 0.5 \rangle \right)$$

It is easy to see that this formula has a number of satisfying strategies. For instance, with a strategy that systematically applies the value of y to a, we obtain the required expected values $\langle 0.5, 0.5 \rangle$. Note that in this example there actually exists an equivalence-preserving conversion to the prenex form. However, in the general case there exists no systematic procedure to gain an equivalence-preserving conversion to prenex form.

Conclusion

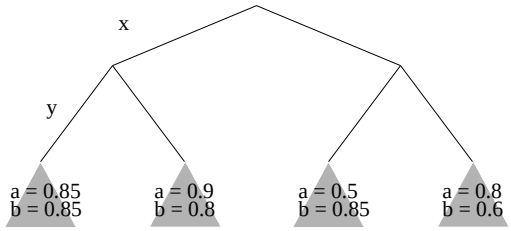
With the single-objective framework the transformation of an arbitrary formula into prenex form is equivalence-preserving. This indicates that the sequentially defined framework is in theory expressive enough in this particular case. We nevertheless believe that the non-prenex form is more natural and that makes the structure apparent and therefore easy to exploit. In contrast in the context of the multi-objective framework it is necessary to use a non-prenex form.

4. DEALING WITH MULTIPLE OBJECTIVES

The Problem

Solving a stochastic constraint satisfaction problem involving a unique objective can be done using a tree-search algorithm involving a number of technicalities that we shall not explain due to space limitations (we refer the reader to [7, 1]). With multiple objectives, an additional issue is that each leaf of the search tree is ranked by a vector of values, one value for each component of the objective.

To see whether existing search algorithms can be adapted to this context we consider an instance of the form $\forall x.\exists y.\ \phi$, also represented in the figure below. Each of the variables x and y have two values, say 0 and 1 (left and right branches). The probability of each branch of the stochastic variable x is 0.5. For the sake of simplicity we do not specify ϕ, which in the figure is abstracted by triangles. We have two objectives: a, on which we impose a threshold of 0.7, and b, on which we have a threshold of 0.8. Note that the evaluation of ϕ in the end of each branch gives us a pair of values for a and b. In general, this pair would not necessarily be unique: each subtree may very well have a number of strategies whose value vectors are incomparable, like $\langle 0.7, 0.6 \rangle$ and $\langle 0.6, 0.7 \rangle$. But for simplicity in the example each subtree does have a unique vector of costs, *e.g.,* $\langle 0.85, 0.85 \rangle$ for the leftmost subtree.



Essentially and ignoring the technicalities and optimisations not directly relevant to our current discussion, the search algorithms that have been proposed for the single-objective case [7, 1] recursively explore the search tree and do the following: once we have explored all branches of a stochastic node we return the weighted sum of the values of these nodes; once we have explored all branches of a decision node we return the value of one of the satisfactory nodes, in fact we can typically choose the one with the best value.

Now with multiple objectives we return vectors of values instead of values. Importantly, there exists no notion of "best vector of values" in general as some vectors may be incomparable. All we can do is therefore check whether one of the branches is satisfactory and return its vector of values. Now following the execution of the algorithm we will see that it becomes non-trivial to adapt the search algorithm to vectors of values. In our example the algorithm explores all branches of the variable x, starting (say) by the left. Then it explores a branch of y (*e.g.,* the left branch). *This branch looks completely satisfactory* (*e.g.,* $\langle 0.85, 0.85 \rangle$) - yet when the algorithm goes ahead and explores the two possibilities for the right branch of x, none of these possibilities allows us to satisfy the thresholds for a and b (*i.e.,* 0.7 and 0.8) at the same time. In one case the expected value of a will be $(0.85 + 0.5)/2 < 0.7$, in the other case the expected value of b will be $(0.85 + 0.6)/2 < 0.8$. As it turns out, *we should have considered the leaf with values* $\langle a = 0.9, b = 0.8 \rangle$ when

exploring the first value of x, as this leaf, together with the leaf $\langle a = 0.5, b = 0.85\rangle$ for the branch on x, satisfies the thresholds. But how could we guess this before having explored the right branch of x?

The Solution

To fix the previous problem the idea is that when we inspect a stochastic node, we have to make sure to consider *all the vectors of values that can be obtained for each subtree.* To ensure that, we enumerate all the possible combinations of vectors of thresholds for each branch. In fact, it is only necessary to enumerate the combinations of vectors whose weighted sum yields a vector satisfying the thresholds imposed at the current level. Once the combinations are fixed, we can recursively ask to each sub-branch whether a subtree can be found that satisfies the vector of thresholds we have fixed for this branch. Note that we shall potentially explore each of the branches multiple times[3].

```
1: ALGORITHM solve(Ψ, ⟨θ_1 ... θ_m⟩):
2:
3: if Ψ is of the form ∃x.φ then
4:    for all v ∈ D_x do
5:       if solve(φ[x := v], ⟨θ_1 ... θ_m⟩) then
6:          return true
7:    return false
8: else if Ψ is of the form ∀x.φ then
9:    for all ⟨λ^1_{d_1} .. λ^m_{d_1}⟩, ..., ⟨λ^1_{d_n} .. λ^m_{d_n}⟩
                 s.t. ⋀_j (Σ_i λ^j_i · p_x(i) ≥ θ_j) do
10:      ok ← true
11:      for all v ∈ D_x do
12:         if ¬ solve (φ[x := v], ⟨λ^1_v .. λ^m_v⟩) then
13:            ok ← false
14:      if ok then return true
15:   return false
16: else
17:   return the vector of values for each objective
```

The θs represent the thresholds imposed on each of the m objectives. The algorithm verifies if these given thresholds can be satisfied or not. The matrix $\langle\lambda^1_{d_1}..\lambda^m_{d_1}\rangle, \ldots, \langle\lambda^1_{d_n}..\lambda^m_{d_n}\rangle$ represents all combinations of thresholds; the sum checks that the currently tested combination is valid (averaging to the θs). The notation $\phi[x := v]$ indicates that we instantiate variable x by v in ϕ.

A careful investigation of the previous algorithm shows that there is room for many optimisations that we do not detail as our goal is to keep its presentation minimal. Nevertheless, the cost of dealing with multiple objectives seems to be high: the algorithm is extremely redundant and explores some branches multiple times that would be explored only once if we use a single objective.

Conclusion

We note that our algorithm provides a proof that stochastic constraint satisfaction with multiple chance constraints can be solved in polynomial space, and is therefore **PSPACE**-complete (the hardness part is trivial). The result was indeed stated in [5]: this paper considers a stochastic constraint satisfaction framework that allows to define multiple chance constraints, each of which has a different threshold (which is essentially equivalent to what we have considered). While the result announced is correct, the proof of *membership in **PSPACE** follows from the existence of a naive algorithm [... which] recurses through the variables in order, making an "and" branch for a stochastic variable and an "or" branch for a decision variable.* This clearly does not hold for multiple objectives, as we have seen, and indeed the search algorithm has to explore the same branches potentially an exponential number of times, asking every time for a different vector of values.

[3]A minor technicality regards the precision of the enumeration. The only real values that need to be considered are determined by the probabilities involved in the problem, but we will not discuss this in detail.

5. SUMMARY OF THE CONTRIBUTIONS

In this paper we have proposed a number of modifications to the stochastic CSP framework which keep the essence of the original definition but enable the framework to model optimisation problems with branching time. The paper involves a number of new technical results which we now summarise: (1) we have shown that the search algorithms proposed in the literature cannot directly be used to solve multi-objective stochastic problems; (2) we have proposed the first search algorithm for these problems; while this algorithm is naive but it provides the first proof of membership in **PSPACE** of multi-objective stochastic CSP; (3) we have shown that non-prenex stochastic CSP cannot, in general, be put in prenex form.

A number of conclusions and guidelines naturally follow. In particular, whenever possible we advise to prefer an approach in which multiple objectives are aggregated into a unique objective; this avoids having to deal with considerably more complex algorithms.

Uncertainty is a topic of increasing importance in constraint satisfaction [2]. We believe stochastic constraint satisfaction to be an appealing framework in which a particular type of problems with uncertainty can be formalised and solved. These problems are those in which the uncertainty arises from a forecast on future decisions, with an alternation between decision and stochastic variables. Our hope is that the results presented in this paper will help developing the applications of the framework.

6. REFERENCES

[1] T. Balafoutis and K. Stergiou. Algorithms for stochastic csp. In *Proc. of Int. Conf. on Principles and Practice of Constraint Programming (CP)*. Springer, 2006.

[2] K. M. Brown and I. Miguel. Uncertainty and change. In F. Rossi, P. Van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 21. Elsevier, 2006.

[3] M. L. Littman, S. M. Majercik, and T. Pitassi. Stochastic boolean satisfiability. *J. of Automated Reasoning*, 27(3):251–296, 2001.

[4] C. Papadimitriou. Games against nature. *J. of Computer and System Sciences*, 31(2):288–301, 1985.

[5] A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

[6] M. Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. of Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 1–22. Springer, 2001.

[7] T. Walsh. Stochastic constraint programming. In *Proc. of Euro. Conf. on Artificial Intelligence (ECAI)*, pages 111–115. John Wiley and Sons, 2002.