Hybrid Metaheuristics for Endogenous Stochastic Constraint Programming*

S. D. Prestwich[1], S. A. Tarim[2], R. Rossi[3], and B. Hnich[4]

[1]Cork Constraint Computation Centre, University College Cork, Ireland
[2]Department of Management, Hacettepe University, Ankara, Turkey
[3]University of Edinburgh Business School, Edinburgh, UK
[4]Department of Computer Engineering, Izmir University of Economics, Turkey
s.prestwich@cs.ucc.ie, armtar@yahoo.com,
roberto.rossi@ed.ac.uk, brahim.hnich@ieu.edu.tr

Abstract. Stochastic Constraint Programming (SCP) is an extension of Constraint Programming for modelling and solving combinatorial problems involving uncertainty. This paper makes three contributions to the field. Firstly we propose a metaheuristic approach to SCP that can scale up to large problems better than state-of-the-art complete methods. Secondly we show how to use standard filtering algorithms to handle hard constraints more efficiently during search. Thirdly we extend our approach to problems with endogenous uncertainty, in which probability distributions are affected by decisions. This extension enables SCP to model and solve a wider class of problems.

1 Introduction

Many real-world problems contain elements of uncertainty, and these are often modelled and solved by Stochastic Programming (SP) methods [5]. Stochastic Constraint Programming (SCP) is an extension of Constraint Programming (CP) designed to model and solve similar problems, a research direction first proposed in [3, 36]. A motivation for SCP is that it should be able to exploit the more complex variable types and constraints used in CP, leading to more compact models and the use of powerful filtering algorithms. But Stochastic Constraint Satisfaction Problems (SCSPs) are in a higher complexity class than Constraint Satisfaction Problems (CSPs) and can be much harder to solve.

An m-stage SCSP is defined as a tuple $(V, S, D, P, C, \theta, L)$ where V is a set of decision variables, S a set of stochastic variables, D a function mapping each element of $V \cup S$ to a domain of values, P a function mapping each variable in S to a probability distribution, C a set of constraints on $V \cup S$, θ a function mapping each constraint $h \in C$ to a threshold value $\theta \in (0, 1]$, and $L = [\langle V_1, S_1 \rangle, \ldots, \langle V_m, S_m \rangle]$ a list of *decision stages* such that the V_i partition V and the S_i partition S. Each constraint must contain at least one V variable, a constraint h with threshold $\theta(h) = 1$ is a *hard constraint*, and

* This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 05/IN/I886. S. A. Tarim is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. MAG-110 K500.

Constraints:
$c_1 : \Pr\{s_1x_1 + s_2x_2 \geq 30\} \geq 0.75$
$c_2 : \Pr\{s_2x_1 = 12\} \geq 0.5$
Decision variables:
$x_1 \in \{1, 2, 3, 4\}$
$x_2 \in \{3, 4, 5, 6\}$
Stochastic variables:
$s_1 \in \{4(0.5), 5(0.5)\}$
$s_2 \in \{3(0.5), 4(0.5)\}$
Stage structure:
$V_1 = \{x_1\} \quad S_1 = \{s_1\}$
$V_2 = \{x_2\} \quad S_2 = \{s_2\}$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle]$

Fig. 1. A simple SCSP example.

one with $\theta(h) \leq 1$ is a *chance constraint*. To solve an m-stage SCSP an assignment to the variables in V_1 must be found such that, given random values for S_1, assignments can be found for V_2 such that, given random values for $S_2, \ldots$ assignments can be found for V_m such that, given random values for S_m, the hard constraints are each satisfied, and the chance constraints (containing both decision and stochastic variables) are satisfied in the specified fraction of all possible *scenarios* (set of values for the stochastic variables).

An SCSP solution is a *policy tree* of decisions, in which each node represents a value chosen for a decision variable, and each arc from a node represents the value assigned to a stochastic variable. Each path in the tree represents a different possible scenario and the values assigned to decision variables in that scenario. A *satisfying policy tree* is a policy tree in which each chance constraint is satisfied with respect to the tree. A chance constraint $h \in C$ is satisfied with respect to a policy tree if it is satisfied under some fraction $\phi \geq \theta(h)$ of all possible paths in the tree.

As an example, consider the 2-stage SCSP in Figure 1 where the numbers in brackets indicate that the stochastic variable values each have probability 0.5. Decision variable x_1 must be set to a fixed value while the value of x_2 depends on that of s_1. A policy tree for this problem is shown in Figure 2. The four scenarios A, B, C and D each have probability 0.25. Constraint c_1 is satisfied in A, B, C and D hence with probability 1.0. Constraint c_2 is satisfied in A and C hence with probability 0.5. These probabilities satisfy the thresholds $\theta(c_1)$ and $\theta(c_2)$ so this is a satisfying policy.

The above framework is a CP analogue of the subfield of SP known as *Chance-Constrained* (or *Probabilistic Constrained*) *Programming*. The form in which it is stated, involving both decision and stochastic variables, is sometimes called the *implicit form* in the SP literature [5], which has both decision and stochastic variables (though the SP notation is different). The alternative is the *extensive form* in which scenarios are expanded to form a *deterministic equivalent* problem containing only decision variables. The extensive form is typically much larger than the implicit form. We shall use only the implicit form in this paper and in our SCP solver.

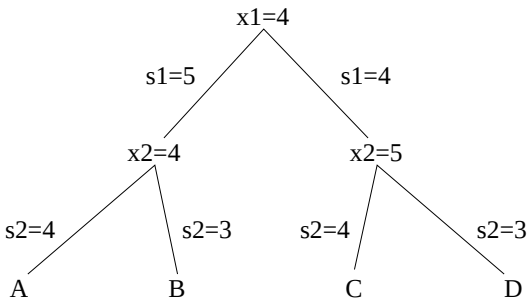

Fig. 2. A satisfying policy tree for the SCSP in Figure 1.

We extend this version of SCP in two ways. Firstly, we allow constraints on statistical parameters of the problem variables, such as expectation or variance. Note that a chance constraint h with threshold $\theta(h)$ can be written as a constraint on an expectation $\mathbb{E}\{\text{reify}(h)\} \geq \theta(h)$ where reify(h) is 1 if h is satisfied and 0 otherwise. Secondly, we may wish to solve an SCSP while minimising (or maximising) an objective function, in which case we have a Stochastic Constrained Optimisation Problem (SCOP). The objective function may be defined in terms of statistical parameters, for example we may wish to maximise the expectation or minimise the variance of a decision variable, or maximise a chance constraint threshold. In all these cases an SCOP can be solved as a series of SCSPs with an increasingly tight constraint.

Several complete methods for solving multi-stage SCP problems have been proposed but none seems practicable for large problems. Incomplete methods such as local search and genetic algorithms are often more scalable but very little work has been done on applying them to SCP. We propose a new approach: transform a multi-stage problem into an unconstrained optimisation problem which can be solved by metaheuristics. The paper is organised as follows. Section 2 describes the basic method for solving problems in SCP. Section 3 adds standard filtering algorithms to handle hard constraints more effectively. Section 4 extends SCP and our solver to a form of *endogenous uncertainty* in which probability distributions may depend on earlier decisions. Section 5 discusses related work. Section 6 concludes the paper.

This work is based on [28, 29] but contains new results: only random problems were used in [28], Quantified Boolean Formulae were used as an example in [29], both papers used artificial neural networks whereas this paper does not, and this paper extends SCP to endogenous uncertainty and supplies missing proofs. Some of the material was briefly described in [17, 19] but this paper contains the definitive descriptions.

2 Metaheuristics for SCP

In this section we describe a metaheuristic approach to solving SCP problems.

2.1 SCP as unconstrained optimisation

We transform the problem of finding a satisfying policy tree to an unconstrained optimisation problem. We do this via *penalty functions* which are commonly used when applying local search or genetic algorithms to CSPs [10]. Define a variable at each policy tree node, whose values are the domain values for the decision variable at that node. (Our solver does this automatically by traversing the tree once before search begins.) Then a vector of values for these variables represents a policy tree.

For each hard or chance constraint $h \in C$ we define a penalty x_h in each scenario, which is 0 if h is satisfied and 1 if it is violated in that scenario. Then the objective function for a vector **v** is:

$$f(\mathbf{v}) = \sum_{h \in C} \max\{\mathbb{E}\{x_h\} - \theta(h), 0\}$$

We compute $\mathbb{E}\{x_h\}$ by traversing the policy tree, and at each leaf checking whether constraint h is satisfied: if it is then that scenario contributes its probability to $\mathbb{E}\{x_h\}$. If $f(\mathbf{v}) = 0$ then each constraint $h \in C$ is satisfied with probability at least that of its satisfaction threshold $\theta(h)$. When solving an SCOP we impose a constraint on the objective function, and on finding a new solution we tighten this constraint; so $f(\mathbf{v})$ includes a penalty for the SCOP objective function. We can now apply metaheuristics to the following unconstrained optimisation problem: find a vector **v** that minimises $f(\mathbf{v})$ to 0. We shall refer to this method as *Evolved Policies* (EP).

2.2 A genetic algorithm

A solution to an SCP problem is a vector **v** with one value per policy tree node. In principle any metaheuristic algorithm can be used to explore the space of vectors, but in this paper we shall use a genetic algorithm, implemented in the Eclipse constraint logic programming system [1]. We assume a basic knowledge of metaheuristics; a survey can be found in [7], and for further details on evolutionary algorithms in particular we refer the reader to books such as [11].

We represent a vector of values by a chromosome in the obvious way, with one gene per value. The fitness is simply $f(\mathbf{v})$ and is to be minimised. The metaheuristic we use is a version of the Microbial Genetic Algorithm (MGA) [13] shown in Figure 3. Other evolutionary algorithms and a simple local search algorithm also performed well in experiments, but we found the MGA to be robust and efficient and we shall use it throughout the paper. In the MGA chromosomes are notionally arranged in a ring and breeding is restricted to neighbouring chromosomes. This restriction makes the MGA an example of the class of *cellular genetic algorithms* whose localised breeding often leads to greater genetic diversity. The recombination operator is *uniform crossover* in which each gene in the new chromosome takes a value chosen randomly from the corresponding values in the two parent chromosomes. Because the single offspring replaces the weaker of the two parents, this can be interpreted as the weaker parent receiving genetic material from the stronger parent. This mimics bacterial reproduction, hence the name of the algorithm.

initialise the population randomly
repeat until finding desired solution or timeout
    randomly choose 2 neighbouring chromosomes a, b
    recombine a and b to form new chromosome c
    mutate c to form new chromosome d
    replace the less-fit of a and b by d

Fig. 3. The Microbial Genetic Algorithm.

Constraints:

$$\begin{aligned}
\Pr\{x_1s_1 + x_2s_3 + x_3s_5 + x_4s_7 = 80\} &\geq \alpha \\
\Pr\{x_1s_2 + x_2s_4 + x_3s_6 + x_4s_8 \leq 100\} &\geq \beta \\
\Pr\{x_1s_2 + x_2s_4 + x_3s_6 + x_4s_8 \geq 60\} &\geq \beta \\
\Pr\{x_1s_2 + x_3s_6 \geq 30\} &\geq 0.7 \\
\Pr\{x_2s_4 + x_4s_8 = 20\} &\geq 0.05
\end{aligned}$$

Decision variables:

$x_1 \in \{5, \ldots, 10\}$ $x_2 \in \{4, \ldots, 10\}$
$x_3 \in \{3, \ldots, 10\}$ $x_4 \in \{6, \ldots, 10\}$

Stochastic variables:

$s_i \in D_i \quad \forall i \in \{1, \ldots, 8\}$

Stage structure:

$V_1 = \{x_1\}$ $S_1 = \{s_1, s_5\}$
$V_2 = \{x_2\}$ $S_2 = \{s_2, s_6\}$
$V_3 = \{x_3\}$ $S_3 = \{s_3, s_7\}$
$V_4 = \{x_4\}$ $S_4 = \{s_4, s_8\}$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle, \langle V_3, S_3\rangle, \langle V_4, S_4\rangle]$

Fig. 4. Random SCSPs.

The mutation operator we use changes each gene's value with a random value with probability $(1 - 0.5^{n^{-1}})^{-1}$ where n is the number of genes per chromosome. This value was chosen so that the probability of each chromosome being mutated is 0.5. The population size we shall use is 50 unless stated otherwise.

2.3 Experiments

We now show that EP can scale to large SCP problems better than complete methods. We use a benchmark set of random 4-stage SCSPs with 5 chance constraints over 4 decision variables $x_1 \ldots x_4$ and 8 stochastic variables $s_1 \ldots s_8$, shown in Figure 4 where D_i denotes a random domain chosen uniformly from $\{1, 2, 3, 4, 5\}$ with 2 values for $i \in \{1, 3, 5, 7\}$ and 3 values for $i \in \{2, 4, 6, 8\}$. We generate instances for $\alpha \in \{0.05, 0.1, 0.12, 0.15, 0.17, 0.2\}$ and $\beta \in \{0.6, 0.7, 0.8\}$, with 5 different sets of

stochastic variable domains, giving 90 instances in total.[1] None of these instances is known to be unsatisfiable but some might be.

Table 1 compares the global chance-constraint (GCC) method of [14], the scenario-based method (SBA) of [34] and EP. GCC is the state-of-the-art complete method for these problems while SBA was the previous best. All figures are in seconds and "—" denotes that the time is greater than 200 seconds. All times were obtained on a 2.8 GHz Pentium (R) 4 with 512 MB RAM, or on another machine with times normalised to this one; the same machine was used throughout this paper. EP figures are medians over 30 runs.

		problem set 1			problem set 2			problem set 3			problem set 4			problem set 5		
α	β	SBA	GCC	EP	SBA	GCC	EP	SBA	GCC	EP	SBA	GCC	EP	SBA	GCC	EP
0.05	0.6	—	7	10	—	86	96	1	8	13	—	—	16	—	6	7
0.10	0.6	—	8	15	—	—	132	1	7	35	—	13	32	—	6	10
0.12	0.6	—	8	17	—	—	138	1	7	49	—	23	33	—	6	12
0.15	0.6	—	7	21	—	—	165	—	—	63	—	—	47	—	—	17
0.17	0.6	—	8	29	—	—	181	—	—	76	—	—	68	—	—	21
0.20	0.6	—	8	30	—	—	185	—	—	121	—	—	82	—	—	25
0.05	0.7	—	7	17	—	9	116	1	7	21	—	11	27	0	6	12
0.10	0.7	—	—	24	—	—	139	1	8	37	—	—	41	—	—	14
0.12	0.7	—	—	18	—	—	154	1	7	47	—	—	41	—	—	15
0.15	0.7	—	—	31	—	—	189	—	—	63	—	—	54	—	—	19
0.17	0.7	—	—	35	—	—	—	—	—	77	—	—	66	—	—	22
0.20	0.7	—	—	42	—	—	—	—	—	117	—	—	85	—	—	30
0.05	0.8	—	7	40	—	—	—	1	8	30	—	10	56	—	7	19
0.10	0.8	—	8	64	—	—	—	1	7	47	—	12	63	—	—	21
0.12	0.8	—	—	89	—	—	—	1	7	62	—	14	68	—	—	25
0.15	0.8	—	—	92	—	—	—	—	—	73	—	—	81	—	—	28
0.17	0.8	—	—	—	—	—	—	—	—	89	—	—	95	—	—	28
0.20	0.8	—	—	—	—	—	—	—	—	105	—	—	191	—	—	33

Table 1. Results in seconds for random SCSPs.

SBA transforms these SCSPs into (deterministic) CSPs with 6739 variables and 6485 constraints. GCC transforms them into CSPs with 259 variables and 5 constraints. EP transforms them into unconstrained optimisation problems with 185 variables. The SCSPs turn out to be hard for SBA and somewhat easier for GCC. However, EP solves all problems that could be solved by SBA or GCC, as well as many that were solved by neither. Moreover, EP can potentially be improved by using more sophisticated meta-heuristics.

[1] These problems were used in [28] but there was a typographical error in their definition which we correct here.

3 Filtering hard constraints

EP treats hard constraints as a special case of chance constraints. This is not incorrect but it does not exploit CP filtering methods. Moreover, the use of penalty functions is not always the best way of handling constraints in a local search or evolutionary algorithm. For example when solving permutation problems much better results are obtained by designing mutation and recombination operators specifically for deriving new permutations from old ones. We could design genetic operators for specific problems, but here we aim instead for a single solver that will do well on a variety of problem types.

Inspired by hybrid search algorithms for CP, we now describe a filtering technique for hard constraints on finite domain variables. This technique was first reported in [29]. It allows us to use standard CP filtering algorithms by treating stochastic and decision variables uniformly, without the need for specialised filtering algorithms. This greatly simplifies the implementation of our solver. We shall also show that it can give much better results on permutation problems, though it is a generic approach that is not aimed specifically at permutations.

3.1 The method

Evolutionary algorithms (EAs) and local search are often applied to constrained problems, and there are at least three distinct ways of handling constraints in an EA; for an overview see (for example) [10]. The simplest method is the use of *penalty functions* which we used in EP. Another method is *repair* where a chromosome representing an infeasible solution is transformed to one representing a feasible solution. A third method is a *decoder*, which is a function that maps chromosomes to solutions. In the decoder approach a chromosome is no longer a solution, but a set of instructions on how to construct a solution. This can be highly effective on tightly-constrained problems, but unfortunately a decoder is problem-specific.

We shall use filtering algorithms to *partially* decode chromosomes. The stronger the filtering method, the more likely we are to obtain a feasible solution. In the limit as consistency increases we would have a function that could map any chromosome into a feasible solution, but this is an NP-complete task as it amounts to solving a CSP. Nevertheless, we shall show that filtering algorithms can profitably be used as partial decoders for EAs applied to SCP problems. We call the modified method *Filtered Evolved Policies* (FEP). Whereas EP uses a penalty function approach on all constraints, FEP uses a decoder-like approach on the hard constraints and penalty functions on chance constraints.

Before starting the search we traverse the policy tree once without filtering, to count the number of nodes N. During search, as we traverse the policy tree guided by a chromosome $\mathbf{v}$ we apply constraint filtering algorithms using the hard constraints, treating both stochastic and decision variables as in a standard CSP. This may remove values from both decision and stochastic variable domains and cause backtracking, so the complete policy tree might not be traversed. We detect this by counting the number $M_{\mathbf{v}}$ of nodes visited and computing a penalty $t(\mathbf{v}) = (N - M_{\mathbf{v}})/(M_{\mathbf{v}} + 1)$. If all nodes are visited then $M_{\mathbf{v}} = N$ and $t(\mathbf{v}) = 0$. We then modify our objective function to include

Constraints:
$c_1 : o = r$
$c_2 : o \leq c$
Decision variables:
$c, o \in \{0, 1\}$
Stochastic variables:
$r \in \{0(0.5), 1(0.5)\}$
Stage structure:
$V_1 = \{c\} \quad S_1 = \{r\}$
$V_2 = \{o\} \quad S_1 = \emptyset$
$L = [\langle V_1, S_1 \rangle, \langle V_2, S_2 \rangle]$

Fig. 5. The umbrella problem.

the new penalty: $f(\mathbf{v}) + t(\mathbf{v})$. This is also a penalty function, but the penalty is now different *and it depends on the filtering algorithms used*. Though it might appear unsafe to apply filtering to stochastic variables, the additional penalty $t(\mathbf{v})$ makes FEP correct (see Section 3.4). We further modify FEP so that alleles are not used directly as decision variable values, but are instead used as *recommended values*. This will be explained in Section 3.2.

3.2 Illustrative example

We shall illustrate the filtering method with a simple 2-stage SCSP. Suppose we are trying to decide whether to carry an umbrella, given two equally probable scenarios: that it rains or not. We use a first-stage binary decision variable c to denote carrying an umbrella ($c = 1$) or not ($c = 0$). A first-stage binary stochastic variable r indicates whether it rains ($r = 1$) or not ($r = 0$), with equal probabilities 0.5. A second-stage binary decision variable o represents the decision to open the umbrella ($o = 1$) or not ($o = 0$). In SP terminology the latter decision is a *recourse action* as it depends on the scenario. There are three hard constraints. Firstly, we can only put up the umbrella if we carry it: $o \leq c$. Secondly, we do not wish to get wet so if it rains then put up the umbrella: $r \leq o$. Thirdly, we would like to enjoy the sun if it is out, so we only put up the umbrella if it rains: $o \leq r$. The last two constraints can be combined into one: $o = r$. An SCSP model for this problem is shown in Figure 5.

This SCSP has 8 possible plans corresponding to the 2 choices for c combined with the 2 choices for o in each of the 2 scenarios. There is only one satisfying policy tree: carry the umbrella, and put it up if and only if it rains, as shown in Figure 6(i). In this figure satisfied constraints are indicated by ticks and violated constraints by crosses. EP can be used to search for this policy tree which we shall represent by a chromosome **101**: the first digit is the value of c, the second is the value of o in the scenario $r = 0$, and the third is the value of o in the scenario $r = 1$. In all scenarios both constraints are satisfied so the penalty is $f(\mathbf{101}) = 0$. FEP handles this chromosome by counting the number of nodes $M_{\mathbf{101}}$ visited by using this chromosome, and the number of nodes N in a satisfying policy tree; both are 3 so the penalty under FEP is

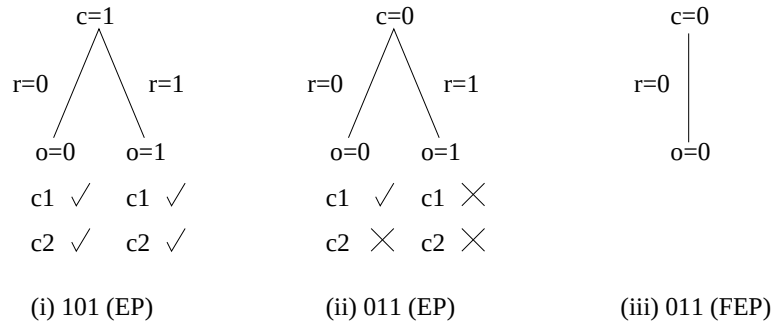

Fig. 6. Three policy trees for the umbrella problem.

$t(\mathbf{101}) = (3-3)/(3+1) = 0$. Hence for this chromosome EP and FEP both return a penalty of 0 indicating satisfaction, as would be expected.

Now consider how EP handles a chromosome corresponding to a *non-satisfying* policy tree such as **011**, in which we do not carry the umbrella but we always open it. The tree is shown in Figure 6(ii). In scenario $r = 0$ constraint c_1 is satisfied but c_2 is violated, while in scenario $r = 1$ both constraints are violated, so the penalty is $f(\mathbf{011}) = 0.5 \times 1 + 0.5 \times 2 = 1.5$. FEP handles this chromosome differently. It uses the first gene to assign $c = 0$, then uses constraint c_2 to filter the value 1 from dom(o) (the domain of o): an example of filtering on a stochastic variable domain. This removal triggers the further removal of value 1 from dom(s) via constraint c_1, so now dom(r) = dom$(o) = \{0\}$. The recommended values for o in each scenario are now irrelevant, as 0 must be chosen in both cases. The policy tree corresponding to this chromosome using FEP is shown in Figure 6(iii). Because dom$(r) = \{0\}$ the policy tree has only 2 nodes instead of 3, so the penalty is $t(\mathbf{011}) = (3-2)/(2+1) = 1/3$. FEP's filtering has pruned the domain dom(r) of a stochastic variable r, leading to an incomplete policy tree. This may appear to be incorrect but it only occurred because the policy tree is non-satisfying: FEP did not prune dom(r) in the case of the satisfying policy tree. EP and FEP simply use different functions to penalise non-satisfying trees.

This simple example also illustrates another feature of FEP. Under EP there is exactly one chromosome **101** corresponding to a solution, but under FEP there are *four* such chromosomes: **100**, **101**, **110** and **111**. As long as $c = 1$ the values of o in both scenarios are fixed by filtering on constraint c_1, so the recommended values are irrelevant. We shall show that this can make it easier to find a satisfying policy tree using FEP.

An additional advantage of FEP is that it can be used to reduce chromosome length. If we know that a decision variable is functionally dependent on earlier assignments then we need not represent it in the chromosome, as its recommended values are irrelevant. In this example constraint c_1 makes o functionally dependent on r so the chromosome needs only one gene, for c. Our FEP implementation allows this information to be provided by the user.

3.3 Experiments

In the well-known *Travelling Salesman Problem* (TSP) we must find a permutation of a set of cities such that visiting the cities in that order, and returning to the first city, minimises the total distance travelled. In the *Probabilistic TSP* (PTSP) [16] the probability of city i being in the tour is p_i; if it is not in the tour then it is ignored, and the distance between its predecessor and successor cities is used. The objective is to choose a tour that minimises the expected total distance travelled. We shall use the PTSP to test two hypotheses empirically: (i) can filtering improve performance, and (ii) can stronger filtering further improve performance?

We model the PTSP as an SCOP as follows. To model a permutation of N cities we use decision variables $v_i \in \{1, \ldots, N\}$ ($i = 1 \ldots N$) which must all take different values to form a permutation. We assume additional cities 0 and $N + 1$ are the start and end points of each tour, and that the distances between cities 0 and 1, and between N and $N + 1$, are 0. Use binary stochastic variables b_i ($i = 1 \ldots N$) where $b_i = 1$ means that city i is present in the tour.[2] An SCP model is shown in Figure 7. `tourlength` is a *global chance constraint* [32] that computes the length λ of a tour in a scenario. To do this it needs the values of the b_i and the $N \times N$ distance matrix M_{ij} of the graph. `alldifferent` is a standard global constraint that forces its parameters to take different values.[3]

We compare four implementations of `alldifferent`, two using EP:

- a single `alldifferent` chance constraint with threshold 1
- N^2 pairwise disequality chance constraints

and two using FEP:

- N^2 pairwise disequality hard constraints with arc consistency (AC)
- a single `alldifferent` hard constraint with generalised arc consistency (GAC) [30]

A drawback with this problem for our purposes is that FEP maps any chromosome to a valid permutation, whether using either GAC, pairwise AC or even pairwise forward checking. One way of making GAC and pairwise AC filter differently is to restrict some variable domains, and we somewhat arbitrarily choose to restrict the domains of $v_{N/2} \ldots v_N$ to $\{1, \ldots, N/2\}$. We compare the performance of our solver using the four `alldifferent` implementations with various values of N.

For simplicity we use complete graphs with $M_{ij} \equiv 1$ and $p_i \equiv 0.5$. Then the tour length in each scenario is independent of the permutation: $\lambda \equiv \sum_{i=1}^{N} b_i$ so all permutations have the same expected length $\mathbb{E}\{\lambda\} = N/2$. We are left with an SCSP in which we must simply find a permutation. We apply the same evolutionary algorithm as in Section 2 and take the median of 30 runs for $N \in \{5, 10, 15, 20, 25\}$. The results are shown in Table 2, where "?" indicates that the median runtime is greater than 1 minute and "chr" denotes the number of chromosomes used.

[2] There is an $O(N^3)$ expression for expected tour length but this is not relevant here.
[3] This is a hard constraint: the *stochastic* `alldifferent` constraint has a different meaning.

Minimise:
$\mathbb{E}\{\lambda\}$
Constraints:
$\texttt{alldifferent}(\{v_i\})$
$\texttt{tourlength}(\{b_i\}, M_{ij}, \lambda)$
Decision variables:
$v_i \in \{1, \ldots N\}$ $(\forall i = 1 \ldots N)$
Stochastic variables:
$b_i \in \{0(1-p_i), 1(p_i)\}$ $(\forall i = 1 \ldots N)$
Stage structure:
$V_1 = \{v_i\} \quad S_1 = \{b_i\}$
$L = [\langle V_1, S_1 \rangle]$

Fig. 7. An SCP model for the PTSP.

	EP				FEP			
	(1)		(2)		(3)		(4)	
N	chr	sec	chr	sec	chr	sec	chr	sec
5	26	0	41	0.01	1	0.00	1	0.00
10	22428	2	772	2	54	0.00	1	0.00
15	?	?	3054	19	206	0.02	1	0.00
20	?	?	?	?	328	0.04	1	0.00
25	?	?	?	?	476	0.08	1	0.00

Table 2. Comparing 4 implementations of `alldifferent`.

Method (1) is extremely poor: in fact it is worse than randomly choosing chromosomes, which would take on average 2^N iterations. Method (2) does better because it distinguishes between chromosomes that are different distances from being permutations: for example when $N = 5$ chromosome **11245** more nearly corresponds to a permutation of $\{1, 2, 3, 4, 5\}$ than **11111** does, but method (1) does not distinguish between them. Method (3) is much better, as its use of pairwise AC filtering enables it to partially decode chromosomes into near-permutations. Method (4) is the best: GAC enables it to fully decode all chromosomes into permutations.

These results empirically support both of our theoretically-motivated hypotheses. Firstly, both FEP methods beat both EP methods, showing that applying standard filtering algorithms to hard constraints can give much better results than treating hard constraints as chance constraints. Secondly, method (4) beats method (3), showing that using stronger filtering algorithms can boost performance further. Of course, as in (non-stochastic) CP, these results are unlikely to apply to all problems as runtime overheads might outweigh the advantages of filtering.

3.4 Properties

We now prove some useful FEP properties. Firstly it is important to show correctness:

Proposition 1. *FEP is correct.*

Proof. If a chromosome **v** represents a policy in which hard constraints are never violated then its recommended decision variable assignments must be consistent with the stochastic variable assignments in every scenario, so no filtering will occur on a stochastic variable domain in any scenario, every node will be visited, $M_{\mathbf{v}} = N$ and $t(\mathbf{v}) = 0$. Conversely, if $t(\mathbf{v}) = 0$ then $M_{\mathbf{v}} = N$, which is only possible if hard constraints are never violated, so **v** represents a policy in which hard constraints are never violated. Hence $t(\mathbf{v}) = 0$ if and only if the policy represented by **v** satisfies the hard constraints, which shows the correctness of FEP. □

As shown in Section 3.3, FEP can be more efficient than EP. A likely explanation is that FEP solves an optimisation problem with more optimal solutions (chromosomes representing satisfying policy trees for the SCSP) than EP:

Proposition 2. *The optimisation problem representing an SCSP has more optimal solutions under FEP than under EP.*

Proof. Firstly, we show that any EP solution is also a FEP solution. In the EP solution, the recommended value for each variable solves the problem, therefore it satisfies all hard constraints, so filtering cannot remove those values.

Secondly, we show by example that there exists a FEP solution for an SCSP, using a certain filtering algorithm, that is not an EP solution. We cite the illustrative example of Section 3.2 in which chromosomes 100, 110 and 111 are solutions under FEP but not EP. □

Section 3.3 also showed that using stronger filtering can further boost performance. Again a likely explanation is that it further increases the number of optimal solutions:

Proposition 3. *The optimisation problem representing an SCSP has more optimal solutions under FEP if the level of consistency is increased.*

Proof. Firstly, we show that any FEP solution with a given filtering algorithm $\mathcal{A}$ is also a solution under a stronger filtering algorithm $\mathcal{B}$. In the FEP solution under $\mathcal{A}$, taking the nearest remaining value in any decision variable domain under $\mathcal{A}$ solves the problem. Stronger filtering can only remove more values from the domain, but the value chosen under $\mathcal{A}$ will not be pruned because it was correct, so it will also be chosen under $\mathcal{B}$.

Secondly, we show by example that there exists an SCSP and filtering algorithms $\mathcal{A}$ and $\mathcal{B}$, with $\mathcal{B}$ stronger than $\mathcal{A}$, and a solution under $\mathcal{B}$ that is not a solution under $\mathcal{A}$. Take the SCSP in Figure 8, let $\mathcal{A}$ enforce arc consistency on each of the disequality constraints representing the `alldifferent` constraint, let $\mathcal{B}$ enforce GAC on the constraint, and let the chromosome contain genes $(x = 0, y_0 = 0, y_1 = 0)$ where y_i denotes y in the scenario with $s = i$. This is a solution under $\mathcal{B}$ because GAC applied before any assignments are made removes 0 and 1 from dom(x), so x will be set to the nearest remaining value which is 2. Then s is assigned $\omega(s) \in \{0, 1\}$ and GAC removes $\omega(s)$ from dom(y), so y is assigned $1 - \omega(s)$. However, it is not a solution under $\mathcal{A}$ because pairwise arc consistency does not remove 0 and 1 from dom(x), so x will follow the recommendation and be assigned value 0. This cannot lead to a solution. □

Constraints:
$\Pr\{\texttt{alldifferent}(x, y, s)\} = 1$
Decision variables:
$x \in \{0, 1, 2, 3\}$
$y \in \{0, 1\}$
Stochastic variables:
$s \in \{0(0.5), 1(0.5)\}$
Stage structure:
$V_1 = \{x\} \quad S_1 = \{s\}$
$V_2 = \{y\} \quad S_2 = \emptyset$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle]$

Fig. 8. SCSP used in the proof of Proposition 3.

4 Endogenous uncertainty

The form of uncertainty addressed so far is sometimes called *exogenous*: the stochastic variable probability distributions are fixed and known at the start. Exogenous uncertainty is the only type handled in most SCP and SP research. However, some problems have *endogenous uncertainty* which make them much harder to solve.

Endogenous uncertainty may be of two types. Firstly, the probability distribution of a stochastic variable may depend upon the values of decision variables from the same or an earlier stage. Secondly, the time at which the uncertainty is observed may depend upon the values of earlier decision variables, though the probability distributions are fixed and known at the start. The latter are sometimes called *STOXUNO* problems (STochastic Optimisation problems with eXogenous Uncertainty and eNdogenous Observations) [24]. We shall not tackle STOXUNO problems in this paper but we extend SCP to include the first type of endogenous uncertainty.

For a survey of work on endogenous uncertainty see [12], which mentions applications including network design and interdiction, server selection, facility location, and gas reservoir development. Other examples include clinical trial planning [9] and portfolio optimisation [33].

4.1 Handling decision-dependent probabilities

To model endogenous uncertainty we simply allow decision variables to specify probability distributions. That is, the probability associated with a value for stochastic variable $v \in S_i$ is allowed to be specified by a decision variable $x \in V_j$ where $j \leq i$. (We assume that the stochastic variable domains are the same in each case: if not then we can take their union as the domain and set some probabilities to 0.) To handle this extension we allow real-valued decision variables, which must be constrained to be functionally dependent upon the values of stochastic or other decision variables that have already been assigned. These variables need not be represented in the chromosome because they are functionally determined. The same is true of other functionally-determined decision variables, and the user can specify which these are.

These are the only modifications we need to model decision-dependent probabilities and the solver needs no other changes. The first application of this technique was described in [17] on a production planning problem and led to much faster performance than an SP approach.

4.2 A disaster planning application

As an example we take the stochastic network problem of Peeta *et al.* [25]. Consider a transportation network, each of whose links (bridges) may fail with some probability. The failure probability of a link can be reduced by investing money in it, and we have a budget limiting the total investment. We would like to minimise the expected shortest path between a specified source and sink node in the network. More generally, we might minimise a weighted sum of expected shortest paths for selected sources and sinks, chosen to represent (for example) high population areas and hospitals; for simplicity we shall consider only a single source and sink. This type of problem arises in pre-disaster planning, where a decision maker aims to maximise the robustness of a transportation network with respect to possible disasters, to facilitate rescue operations.

Endogenous uncertainty arises in this problem because the decisions (which links to invest in) affects the probabilities of the random events (the link failures). This is a 2-stage problem. In the first stage we must decide which links to invest in, then link failures occur randomly. In the second stage we must choose a shortest path between the source and sink (the recourse action), given the surviving links. If the source and sink are no longer connected then a fixed penalty is imposed. Peeta *et al.* point out that, though a natural approach is to strengthen the weakest links, this does not necessarily lead to the best results.

We can model the problem in SCP as follows. For each link $e \in E$ (where E is the set of links in the network) we define a binary decision variable y_e which is 1 if we invest in that link and 0 otherwise. We define a binary stochastic variable r_e which is 1 if link e survives and 0 if it fails. We define a single second-stage decision variable z to be computed by a shortest-path algorithm. Following Peeta *et al.* denote the survival (non-failure) probability of link e by p_e without investment and q_e with, the investment required for link e by c_e, the length of link e by t_e, the budget by B, and the penalty for no path from source to sink by M. Our 2-stage SCP is shown in Figure 9 where `shortest_path_cost`(M, t_e, r_e, z) is a global chance constraint that constructs a representation of the graph from the r_e values, uses Dijkstra's algorithm to find a shortest path between source and sink, and computes its length z; if source and sink are unconnected then $z = M$. We implemented this constraint via an Eclipse *suspended goal* whose execution is delayed until the second stage. To model probabilities we define real auxiliary decision variables f_e (failure) and s_e (survival). The f_e are constrained to be $1 - p_e$ if link e is invested in ($y_e = 1$) and $1 - q_e$ otherwise. The probabilities must sum to 1 so $s_e = 1 - f_e$. All constraints are hard and are handled by the filtering method described in Section 3.

Only the y_e are independent decision variables: the f_e, s_e, z variables are all functionally dependent on the y_e. Our solver allows the user to specify functionally-dependent variables so that no genes need be used for them; identifying them leaves only 12 genes per chromosome, one for each y_e.

Minimise:
$\mathbb{E}\{z\}$
Constraints:
$c_1 : \sum_{e \in E} c_e y_e \leq B$
$c_2 : f_e = y_e(1 - q_e) + (1 - y_e)(1 - p_e) \quad (\forall e \in E)$
$c_3 : s_e = 1 - f_e \quad (\forall e \in E)$
$c_4 : \texttt{shortest\_path\_cost}(M, t_e, r_e, z)$
Decision variables:
$y_e \in \{0, 1\} \quad (\forall e \in E)$
$f_e, s_e, z \in \mathbb{R} \quad (\forall e \in E)$
Stochastic variables:
$r_e \in \{0(f_e), 1(s_e)\} \quad (\forall e \in E)$
Stage structure:
$V_1 = \{y_e, f_e, s_e \mid e \in E\} \quad S_1 = \{r_e \mid e \in E\}$
$V_2 = \{z\} \quad S_2 = \emptyset$
$L = [\langle V_1, S_1 \rangle, \langle V_2, S_2 \rangle]$

Fig. 9. An SCP model for the disaster planning problem.

4.3 Experiments

Peeta *et al.* tackle a real road network with 30 directed links, giving a billion scenarios which are then sampled. We will address scenario reduction methods in future work so we do not use this network, which has too many scenarios for FEP to enumerate. They also tackle an 8-node 9-link network, but we do not use this either because not quite enough information is provided to recreate the problem exactly, and no execution times are provided for comparison. Instead we design our own (slightly larger) network with 8 nodes and 12 undirected links, which is small enough for us to compute optimal solutions by brute force in a reasonable time. A further drawback with the disaster planning problem is that we have no other algorithms to compare FEP with: current complete SCP solvers cannot solve this model of the problem because of its endogenous uncertainty. However, we can analyse the usefulness of FEP's metaheuristics.

Our network is shown in Figure 10 with links labelled 1–12, and the problem is to minimise the expected shortest distance between nodes A and B. The problem instances have the parameters shown in Table 3. We set all investments $c_e \equiv 1$ and allow a budget $B = 6$ so that up to 6 links may be invested in. There are $2^{12} = 4096$ scenarios and the same number of possible plans. By enumerating and evaluating all possible plans we can completely solve each instance by brute force in a few minutes. The (unique) optimal investment plans and their costs are shown in Table 4.

Note that some plans may be infeasible, for example the plan in which all links are invested in ($y_e \equiv 1$) violates the budget constraint. But for this problem FEP's filtering always yields a feasible plan. For instances 1–4 exactly 1 chromosome corresponds to the optimal solution, but because of FEP filtering 2 different chromosomes yield the same optimal plan for instance 5: **110011000110** and **110011000111**, which differ only in the final gene. The explanation is that after assigning $y_1 \ldots y_{11}$ to the values shown,

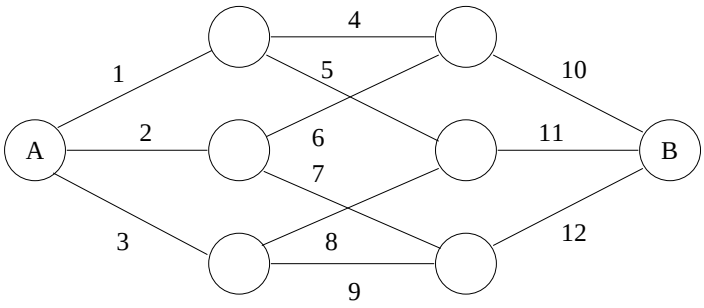

Fig. 10. A transportation network.

$p_e \equiv 0.7$, $q_e \equiv 0.8$, $c_e \equiv 1$, $B \equiv 6$, $M \equiv 100$

No	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
1	10	10	10	10	10	10	10	10	10	10	10	10
2	10	20	10	20	10	20	10	20	10	20	10	20
3	20	10	20	10	20	10	20	10	20	10	20	10
4	20	20	10	10	20	20	10	10	20	20	10	10
5	10	20	30	30	20	10	30	20	10	10	20	30

Table 3. Disaster planning instance parameters.

No	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	cost
1	1	1	1	0	0	0	0	0	0	1	1	1	33.603
2	1	0	1	0	1	0	0	0	1	0	1	1	39.900
3	1	1	1	0	0	1	0	0	0	1	0	1	40.050
4	0	1	1	0	0	0	1	1	0	0	1	1	39.943
5	1	1	0	0	1	1	0	0	0	1	1	0	50.705

Table 4. Disaster planning optimal investment plans.

there are already 6 variables assigned to 1, so 1 is filtered from dom(y_{12}) and y_{12} is assigned to 0 whatever value the gene recommends.

The computational results are shown in Table 5: the mean number of chromosomes c, and the mean execution time "sec" in seconds. If metaheuristics were of no use we would expect them to perform no better than random selection with replacement, which would take a mean of $4096/(s+1)$ chromosomes to find an optimal solution, where s is the number of chromosomes corresponding to an optimal solution. The table also shows the number of chromosomes s corresponding to an optimal solution, and the speedup relative to random selection which is computed as $4096/c(s+1)$. We use a smaller population size of 10 for this problem. Mean results are reported over 10 runs. FEP uses 2.8–8.4 times fewer chromosomes than random selection, showing the benefit of metaheuristics for this problem.

	instance no.				
	1	2	3	4	5
c	333	489	245	379	480
sec.	78	144	48	75	95
s	1	1	1	1	2
speedup	6.2	4.2	8.4	5.4	2.8

Table 5. Disaster planning mean results.

Note that it is easy to change the objective function in EP and FEP. As mentioned above, we can generalise the single source and sink to several, and minimise a weighted sum of shortest paths. We can use Dynamic Programming to compute all shortest path distances from a given node, and take the sum of the lengths. For problems where there is no well-defined source or sink node, we can use the Floyd-Warshall algorithm to compute all shortest path lengths in the network and sum them, though this would incur higher overhead. SP researchers have recently explored *risk-averse* disaster planning including transportation networks [21]. FEP can use risk-averse objective functions such as conditional value-at-risk, as was done in [17]. Or we can use a chance constraint to ensure a minimum service-level (the probability that there exists a path from A to B). All these variations can easily be implemented in FEP.

5 Related work

Several SCSP solution methods have been proposed in the literature. [36] presented two complete algorithms based on backtracking and forward checking and suggested some approximation procedures, while [2] described an arc-consistency algorithm. In the method of [34] an SCSP is transformed into a *deterministic equivalent* CSP and solved by standard CP methods. It is also extended to handle multiple chance constraints and multiple objective functions. This method gives much better performance on the book production planning problem of [36] compared to the tree search methods. To

reduce the size of the CSP *scenario reduction* methods are proposed, as used in SP. These choose a small but representative set of scenarios. However, it might not always be possible to find a small representative set of scenarios, and in some cases choosing an inappropriate set of scenarios can yield an unsolvable CSP. Moreover, using even a modest number of scenarios leads to a CSP that is several times larger than the original SCSP. [8] modify a standard backtracking algorithm to one that can handle multiple chance constraints and uses polynomial space, but is inefficient in time. For the special case of SCP with linear recourse, [35] propose a Bender's decomposition algorithm. [15, 31] proposed a cost-based filtering technique for SCP, and [14] generalised this problem-specific approach to global constraints. The design of local search algorithms for SCP was suggested in order to improve scalability [36], but this idea does not seem to have been pursued further.

Stochastic Boolean Satisfiability (SSAT) is related to SCP. A recent survey of the SSAT field is given in [23], on which we base this discussion. An SSAT problem can be regarded as an SCSP in which all variable domains are Boolean, all constraints are extensional and may be non-binary, and all constraints are treated as a single chance constraint (there are also restricted and extended versions). Our method therefore applies immediately to SSAT problems. SSAT algorithms fall into three classes: systematic, approximation, and non-systematic. Systematic algorithms are based on the standard SAT backtracking algorithm and correspond roughly to some current SCP algorithms. Approximation algorithms work well on restricted forms of SSAT but less well on general SSAT problems. For example the APPSSAT algorithm [22] considers scenarios in decreasing order of probability to construct a partial tree, but does not work well when all scenarios have similar probability. A non-systematic algorithm for SSAT is randevalssat [20], which applies local search to the decision (existential) variables in a random set of scenarios. This algorithm also suffers from memory problems because it must build a partial tree.

FEP uses *hybrid* metaheuristics, in which metaheuristics are hybridised with other techniques (such as constraint filtering, dynamic programming or other metaheuristics) in order to improve performance on some class of problems. A survey of hybrid metaheuristics (for non-stochastic problems) can be found in [6], including a section on metaheuristic/constraint programming hybrids. Regarding this survey, FEP is most closely related to hybrids such as [18, 26, 27] in which the search space consists of consistent partial variable assignments. In the special case of an SCP problem with no stochastic variables the problem reduces to a CP problem, and FEP performs a hybrid search that uses filtering while building partial assignments whose values are guided by a metaheuristic; in this sense FEP generalises some existing hybrid CP methods to SCP by generalising a partial assignment to a partial policy tree.

Metaheuristics have been applied to stochastic problems many times. The field is too large to cover here but a recent survey is given in [4]. EP can be seen as an adaptation of such methods to SCP. A major aspect of this work is the efficient computation or approximation of the objective function. We have not addressed this issue in EP or FEP, instead computing the objective exhaustively by traversing the policy tree, but in future work we will use these important techniques.

6 Conclusion

We propose EP, a method for solving Stochastic Constraint Programming problems by metaheuristics. EP is the first incomplete algorithm that is applicable to SCP models without scenario expansion, though as noted in Section 1 one can also apply incomplete search *indirectly* via a deterministic equivalent model. Experiments show that on some problems EP is faster than current complete methods. We also proposed FEP, a hybrid of EP and constraint filtering that can greatly outperform EP on problems with hard constraints. Finally, we extended SCP and our method to include a form of endogenous uncertainty, which allows it to model and solve more problems in a direct and natural way. We suggest that this extended language be called *Endogenous Stochastic Constraint Programming*.

All these features contribute to the power of SCP as a framework for modelling and solving problems involving uncertainty. The use of metaheuristics makes the method more scalable than complete methods to problems with many decision variables. However, for problems with many stochastic variables the cost of traversing the policy tree becomes very high, and our method alone is too weak. For such problems we must use other techniques such as scenario sampling, and this will be the subject of future work.

References

1. K. R. Apt, M. Wallace. Constraint Logic Programming Using Eclipse. Cambridge University Press, 2007.
2. T. Balafoutis, K. Stergiou. Algorithms for Stochastic CSPs. *12th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 4204, Springer, 2006, pp. 44–58.
3. T. Benoist, E. Bourreau, Y. Caseau, B. Rottembourg. Towards Stochastic Constraint Programming: A Study of On-Line Multi-Choice Knapsack with Deadlines. *7th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* 2239, Springer, 2001, pp. 61–76.
4. L. Bianchi, M. Dorigo, L. M. Gambardella, W. J. Gutjahr. A Survey on Metaheuristics for Stochastic Combinatorial Optimization. *Natural Computing* **8**(2):239–287, 2009.
5. J. Birge, F. Louveaux. Introduction to Stochastic Programming. Springer Series in Operations Research, 1997.
6. C. Blum, J. Puchinger, G.R. Raidl, A. Roli. Hybrid Metaheuristics in Combinatorial Optimization: A Survey. *Applied Soft Computing* **11**, 2011.
7. C. Blum, A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys* **35**(3), 2003.
8. L. Bordeaux, H. Samulowitz. On the Stochastic Constraint Satisfaction Framework. *ACM Symposium on Applied Computing*, 2007, pp. 316–320.
9. M. Colvin, C. T. Maravelias. A Stochastic Programming Approach for Clinical Trial Planning in New Drug Development. *Computers and Chemical Engineering* **32**(11):2626–2642, 2008.
10. B. Craenen, A.E. Eiben, E. Marchiori. How to Handle Constraints with Evolutionary Algorithms. Practical Handbook of Genetic Algorithms. L. Chambers (ed.), 2001, pp. 341–361.
11. A. E. Eiben, J. E. Smith. Introduction to Evolutionary Computing. Springer, 2003.
12. V. Goel, I. E. Grossmann. A Class of Stochastic Programs with Decision Dependent Uncertainty. *Mathematical Programming* **108**(2):355–394, 2006.

13. I. Harvey. The Microbial Genetic Algorithm. *10th European Conference on Advances in Artificial Life, Lecture Notes in Computer Science* vol. 5778, 2011, pp. 126–133.
14. B. Hnich, R. Rossi, S. A. Tarim, S. D. Prestwich. Synthesizing Filtering Algorithms for Global Chance-Constraints. *15th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 5732, 2009, pp. 439–453.
15. B. Hnich, R. Rossi, S. A. Tarim, S. D. Prestwich. Filtering Algorithms for Global Chance Constraints. *Artificial Intelligence* **189**:69–94, 2012.
16. P. Jaillet. Probabilistic Traveling Salesman Problems. PhD dissertation, Massachusetts Institute of Technology, MA, USA, 1985.
17. B. Kawas, M. Laumanns, E. Pratsini, S. Prestwich. Risk Averse Production Planning. *Proceedings of the 2nd International Conference on Algorithmic Decision Theory, Lecture Notes in Computer Science* vol. 6992, Springer 2011, pp. 108–120.
18. M. Khichane, P. Albert, C. Solnon. Integration of ACO in a Constraint Programming Language. *Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science* vol. 5217, 2008, pp. 84–95.
19. M. Laumanns, E. Pratsini, S. Prestwich, C.-S. Tiseanu. Production Planning under Non-Compliance Risk. *International Conference on Operations Research: Mastering Complexity, Operations Research Proceedings*, Munich, Germany, 2010, pp. 545–550.
20. M. L. Littman, S. M. Majercik, T. Pitassi. Stochastic Boolean Satisfiability. *Journal of Automated Reasoning* 27(3):251–296, 2001.
21. C. Liu, Y. Fan, F. Ordóñez. A Two-Stage Stochastic Programming Model for Transportation Network Protection. *Computers & Operations Research* **36**:1582–1590, 2009.
22. S. M. Majercik. APPSSAT: Approximate Probabilistic Planning Using Stochastic Satisfiability. *International Journal of Approximate Reasoning* 45(2):402–419, 2007.
23. S. M. Majercik. Stochastic Boolean Satisfiability. Handbook of Satisfiability, Chapter 27, IOS Press, 2009, pp. 887–925.
24. L. Mercier, P. V. Hentenryck. An Anytime Multistep Anticipatory Algorithm for Online Stochastic Combinatorial Optimization. *Annals of Operations Research* **184**(1):233–271, 2011.
25. S. Peeta, F. S. Salman, D. Gunnec, K. Viswanath. Pre-Disaster Investment Decisions for Strengthening a Highway Network. *Computers & Operations Research* **37**:1708–1719, 2010.
26. S. D. Prestwich. Generalized Graph Colouring by a Hybrid of Local Search and Constraint Programming. *Discrete Applied Mathematics* **156**:148–158, 2008.
27. S. D. Prestwich. Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search. *Annals of Operations Research* **115**:51–72, 2002.
28. S. D. Prestwich, S. A. Tarim, R. Rossi, B. Hnich. Evolving Parameterised Policies for Stochastic Constraint Programming. *15th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 5732, 2009, pp. 684–691.
29. S. D. Prestwich, S. A. Tarim, R. Rossi, B. Hnich. Stochastic Constraint Programming by Neuroevolution With Filtering. *7th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, Lecture Notes in Computer Science* vol. 6140, Springer, 2010, pp. 282–286.
30. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. *12th National Conference on Artificial Intelligence*, AAAI Press, 1994, pp. 362–367.
31. R. Rossi, S. A. Tarim, B. Hnich, S. D. Prestwich. Cost-Based Domain Filtering for Stochastic Constraint Programming. *14th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* 5202, Springer, 2008, pp. 235–250.
32. R. Rossi, S. A. Tarim, B. Hnich and S. D. Prestwich. A Global Chance-Constraint for Stochastic Inventory Systems under Service Level Constraints. *Constraints* **13**(4):490–517, 2008.

33. S. Solak, J.-P. B. Clarke, E. L. Johnson, E. R. Barnes. Optimization of R&D Project Portfolios Under Endogenous Uncertainty. *European Journal of Operational Research* **207**:420–433, 2010.
34. S. A. Tarim, S. Manandhar, T. Walsh. Stochastic Constraint Programming: A Scenario-Based Approach. *Constraints* 11(1):1383–7133, 2006.
35. S. A. Tarim, I. Miguel. A Hybrid Bender's Decomposition Method for Solving Stochastic Constraint Programs with Linear Recourse. *Lecture Notes in Computer Science* vol. 3978, Springer, 2006, pp. 133–148.
36. T. Walsh. Stochastic Constraint Programming. *15th European Conference on Artificial Intelligence*, 2002, pp. 111–115.