Stochastic Constraint Programming by Neuroevolution With Filtering*

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]Logistics, Decision and Information Sciences Group, Wageningen UR, the Netherlands
[4]Faculty of Computer Science, Izmir University of Economics, Turkey
s.prestwich@cs.ucc.ie, armtar@yahoo.com,
roberto.rossi@wur.nl, brahim.hnich@ieu.edu.tr

Abstract. Stochastic Constraint Programming is an extension of Constraint Programming for modelling and solving combinatorial problems involving uncertainty. A solution to such a problem is a policy tree that specifies decision variable assignments in each scenario. Several complete solution methods have been proposed, but the authors recently showed that an incomplete approach based on neuroevolution is more scalable. In this paper we hybridise neuroevolution with constraint filtering on hard constraints, and show both theoretically and empirically that the hybrid can learn more complex policies more quickly.

1 Introduction

Stochastic Constraint Programming (SCP) is an extension of Constraint Programming (CP) designed to model and solve complex problems involving uncertainty and probability [7]. 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 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 $h \in C$ containing only V variables is a *hard constraint* with threshold $\theta(h) = 1$, and one containing at least one S variable is a *chance constraint*.

To solve an SCSP we must find 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*

* S. A. Tarim and B. Hnich are supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. SOBAG-108K027. S. A. Tarim is also supported by Hacettepe University (BAB). A version of this algorithm will used to further research in risk management as part of a collaboration with IBM Research, with partial support from the Irish Development Association and IRCSET.

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.

Most current SCP approaches are complete and do not seem practicable for large multi-stage problems, but the authors recently proposed a more scalable method called *Evolved Parameterised Policies* (EPP) [3]. In this paper we hybridise EPP with constraint filtering, and show theoretically and empirically that this improves learning. An upcoming technical report will contain details omitted from this short paper.

2 Filtered Evolved Parameterised Policies

EPP [3] uses an evolutionary algorithm to find an artificial neural network (ANN) whose input is a representation of a policy tree node, and whose output is a domain value for the decision variable to be assigned at that node. The ANN describes a *policy function*: it is applied whenever a decision variable is to be assigned, and can be used to represent or recreate a policy tree. The evolutionary fitness function penalises chance constraint violations, and is designed to be optimal for ANNs representing satisfying policy trees. In experiments on random SCSPs, EPP was orders of magnitude faster than state-of-the-art complete algorithms [3]. Because it evolves an ANN it is classed as a *neuroevolutionary* method (see for example [6]).

A drawback with EPP is that it treats hard constraints in the same way as chance constraints. This is not incorrect, but a problem containing many hard constraints may require a complex ANN with more parameters to tune, leading to longer run times. We now describe a constraint-based technique for the special case of finite domain SCSPs that allows more complex policies to be learned by simpler ANNs.

We modify EPP so that the ANN output is not used to compute a decision variable value directly, but instead to compute a *recommended value*. As we assign values to the decision and stochastic variables under some scenario ω, we apply constraint filtering algorithms using only the hard constraints, which may remove values from both decision and stochastic variable domains. If domain wipe-out occurs on any decision or stochastic variable then we stop assigning variables under ω and every constraint is artificially considered to be violated in ω; otherwise we continue. On assigning a stochastic variable s we choose $\omega(s)$, but if $\omega(s)$ has been removed from dom(s) then we stop assigning variables under ω and every constraint h is artificially considered to be violated in ω; otherwise we continue. On assigning a decision variable x we compute the recommended value then choose the first remaining domain value after it in cyclic order. For example suppose that initially dom$(x) = \{1, 2, 3, 4, 5\}$ but this has been reduced to $\{2, 4\}$, and the recommended value is 5. This value is no longer in dom(x) so we choose the cyclically next remaining value 2. If all variables are successfully assigned in ω then we check by inspection whether each constraint is violated or satisfied.

Some points should be clarified here. Firstly, it might be suspected that filtering a stochastic variable domain violates the principle that these variables are randomly assigned. But stochastic variables are assigned values from their *unfiltered* domains. Secondly, the value assigned to a decision variable must depend only upon the values assigned to stochastic variables occurring *earlier* in the stage structure. Does filtering

Constraints:
$c_1 : \Pr\{x = s \oplus t\} = 1$
Decision variables:
$x \in \{0, 1\}$
Stochastic variables:
$s, t \in \{0, 1\}$
Stage structure:
$V_1 = \emptyset \quad S_1 = \{s, t\}$
$V_2 = \{x\} \quad S_2 = \emptyset$
$L = [\langle V_1, S_1 \rangle, \langle V_2, S_2 \rangle]$

Fig. 1. SCSP used in Proposition 1.

the domains of stochastic variables that occur *later* violate this principle? No: constraint filtering makes no assumptions on the values of unassigned variables, it only tells us that assigning a value to a decision variable will inevitably lead to a hard constraint violation. Thirdly, we consider all constraints to be violated if either domain wipe-out occurs, or if the selected value for a stochastic variable has been removed earlier by filtering. This might appear to make the evolutionary fitness function incorrect. But both these cases correspond to hard constraint violations, and considering constraints to be violated in this way is similar to using a penalty function in a genetic or local search algorithm: it only affects the objective function value for non-solutions.

We call the modified method *Filtered Evolved Parameterised Policies* (FEPP) and now state two useful properties.

Proposition 1. *FEPP can learn more policies than EPP with a given ANN.*

Proof sketch. We can show that any policy that can be learned by EPP can also be learned by FEPP. Conversely, we show by example that there exists an SCSP that can be solved by FEPP but not by EPP using a given ANN. Suppose that the ANN is a single *perceptron* [2] whose inputs are the s and t values and whose output is used to select a domain value for x, the SCSP is as shown in Figure 1, and FEPP enforces arc consistency. A single perceptron cannot learn the $\oplus$ (exclusive-OR) function [2] so EPP cannot solve the SCSP. But arc consistency removes the incorrect value from dom(x) so FEPP makes the correct assignment irrespective of the ANN. $\square$

Proposition 2. *Increasing the level of consistency increases the set of policies that can be learned by FEPP with a given ANN.*

Proof sketch. We can show that any policy that can be learned by FEPP with a given ANN and filtering algorithm $\mathcal{A}$ can also be learned with a stronger filtering algorithm $\mathcal{B}$. Conversely, we show by example that there exists an SCSP, an ANN, and filtering algorithms $\mathcal{A}$ and $\mathcal{B}$, such that the SCSP can be solved by FEPP with $\mathcal{B}$ but not $\mathcal{A}$. Let the SCSP be as shown in Figure 2, $\mathcal{A}$ enforce pairwise arc consistency on the disequality constraints comprising c_2, $\mathcal{B}$ enforce hyper-arc consistency on c_2 using the algorithm of [5], and both $\mathcal{A}$ and $\mathcal{B}$ enforce arc consistency on c_1. In any satisfying policy $x = s \oplus t$.

Constraints:
$c_1 : \Pr\{ x < 2 \rightarrow x = s \oplus t\} = 1$
$c_2 : \Pr\{\text{alldifferent}(x, y, u)\} = 1$
Decision variables:
$x \in \{0, 1, 2, 3\}$
$y \in \{2, 3\}$
Stochastic variables:
$s, t \in \{0, 1\}$
$u \in \{2, 3\}$
Stage structure:
$V_1 = \emptyset \quad S_1 = \{s, t\}$
$V_2 = \{x\} \quad S_2 = \{u\}$
$V_3 = \{y\} \quad S_3 = \emptyset$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle, \langle V_3, S_3\rangle]$

Fig. 2. SCSP used in Proposition 2.

The proof rests on the fact that $\mathcal{B}$ reduces dom(x) to $\{0, 1\}$ before search begins so $\oplus$ can immediately be enforced. FEPP under $\mathcal{A}$ cannot do this so it is forced to learn $\oplus$, which is impossible for a perceptron. $\square$

Thus FEPP can potentially exploit advanced CP techniques such as global constraints. We state without proof two further propositions.

Proposition 3. *The optimisation problem representing an SCSP has more solutions under FEPP than under EPP, with a given ANN.*

A *solution* here is a set of parameter values for the ANN that represents a satisfying policy tree for the SCSP.

Proposition 4. *The optimisation problem representing an SCSP has more solutions under FEPP if the level of consistency is increased, with a given ANN.*

So even where FEPP has the same learning ability as EPP, it may be more efficient because it solves an optimisation problem with more solutions. Increasing the number of solutions is not guaranteed to make the problem easier to solve, especially as filtering incurs a runtime overhead, but it may do so.

3 Experiments

We now test two hypotheses: does filtering enable an ANN to learn more complex policies in practice as well as in theory (proposition 1)? And where a policy can be learned without filtering, does filtering speed up learning (as we hope is implied by proposition 3)? For our experiments we use Quantified Boolean Formula (QBF) instances. QBF and SCSP are closely related as there is a simple mapping from QBF to Stochastic Boolean Satisfiability, which is a special case of SCSP [1]. QBF-as-SCSP is an interesting test for FEPP because *all* its constraints are hard.

instance	EPP	FEPP
cnt01	0.9	0.03
impl02	2.9	0.02
impl04	—	8.8
TOILET2.1.iv.4	—	31
toilet_a_02_01.4	—	9.5
tree-exa10-10	—	4.0

Table 1. Results on QBF instances transformed to SCSPs

We have implemented a prototype FEPP using a weak form of constraint filtering called *backchecking*. We use the same ANN as in [3]: a *periodic perceptron* [4], which has been shown to learn faster and require fewer weights than a standard perceptron. Results for EPP and FEPP are shown in Table 1, both tuned roughly optimally to each instance. All times were obtained on a 2.8 GHz Pentium (R) 4 with 512 MB RAM and are medians of 30 runs. "—" indicates that the problem was never solved despite multiple runs with different EPP parameter settings. These preliminary results support both our hypotheses: there are problems that can be solved by FEPP but not (as far as we can tell) by EPP; and where both can solve a problem FEPP is faster. So far we have found no QBF instance on which EPP beats FEPP.

4 Conclusion

FEPP is a true hybrid of neuroevolution and constraint programming, able to benefit from improvements to its evolutionary algorithm, its neural network and its filtering algorithms. In future work we will work on all three of these aspects and test FEPP on real-world optimisation problems involving uncertainty.

References

1. S. M. Majercik. Stochastic Boolean Satisfiability. Handbook of Satisfiability, Chapter 27, IOS Press, 2009, pp. 887–925.
2. M. Minsky, S. Papert. Perceptrons: An Introduction to Computational Geometry. The MIT Press, Cambridge MA, 1972.
3. 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.
4. R. Racca. Can Periodic Perceptrons Replace Multi-Layer Perceptrons? *Pattern Recognition Letters* 21:1019–1025, 2000.
5. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. *12th National Conference on Artificial Intelligence*, AAAI Press, 1994, pp. 362–367.
6. K. O. Stanley, R. Miikkulainen. A Taxonomy for Artificial Embryogeny. *Artificial Life* 9(2):93–130, MIT Press, 2003.
7. T. Walsh. Stochastic Constraint Programming. *15th European Conference on Artificial Intelligence*, 2002.