Stochastic Constraint Programming by Neuroevolution with Filtering*

Steve D. Prestwich[1], S. Armagan Tarim[2], Roberto Rossi[3], and Brahim 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.

A. Lodi, M. Milano, and P. Toth (Eds.): CPAIOR 2010, LNCS 6140, pp. 282–286, 2010.

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 $\mathrm{dom}(s)$ 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 $\mathrm{dom}(x) = \{1, 2, 3, 4, 5\}$ but this has been reduced to $\{2, 4\}$, and the recommended value is 5. This value is no longer in $\mathrm{dom}(x)$ 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