Stochastic Constraint Programming by Neuroevolution With Filtering

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

[1]Cork Constraint Computation Centre
[2]Hacettepe University
[3]Wageningen UR
[4]Izmir University of Economics

background: SCP

SCP is a hybrid of SP & CP for modelling & solving combinatorial problems involving uncertainty, as in many real-world problems

traditionally tackled by SP, but SCP should be able to exploit the more complex constraints used in CP, leading to more compact models & the use of powerful filtering algorithms

an m-stage SCSP is a tuple $(V, S, D, P, C, \theta, L)$ where V are decision variables, S stochastic variables, D a function mapping $V \cup S$ to a domain of values, P a function mapping S to a probability distribution, C a set of constraints on $V \cup S$, θ a function mapping C to a threshold value $\theta \in (0, 1]$, & *stage structure* $L = [\langle V_1, S_1 \rangle, \ldots, \langle V_m, S_m \rangle]$

an SCSP solution is a *policy tree* of decisions: node = decision variable assignment, arc = stochastic variable assignment

each path in the tree is a *scenario* plus decision variable assignments in that scenario

a *satisfying policy tree* is one in which each chance constraint is satisfied with respect to the tree (ie it is satisfied under some fraction $\phi \geq \theta(h)$ of all possible paths in the tree)

a constraint with threshold $\theta(h) = 1$ is a *hard constraint*, one with $\theta(h) < 1$ is a *chance constraint*

SCP inherits CP's rich variable types & constraints

brief history:

- [Benoist, Bourreau, Caseau & Rottembourg 2001] proposed SCP

- [Walsh 2002] concretised SCP, describing BT & FC complete algorithms & approximation procedures

- [Balafoutis & Stergiou 2006] described an AC algorithm

- [Tarim, Manandhar & Walsh 2006] transformed an SCSP into a *deterministic equivalent* CSP & solved it by standard CP methods; also used *scenario reduction* methods

• [Tarim & Miguel 2006] proposed a Bender's decomposition algorithm for special case of SCP with linear recourse

• [Bordeaux & Samulowitz 2007] modified a BT algorithm to handle multiple chance constraints & uses polynomial space, but it was inefficient in time

• [Rossi, Tarim, Hnich & Prestwich 2008] proposed a *cost-based filtering* technique for SCP that beat previous complete methods on random SCSPs

• [Prestwich, Tarim, Rossi & Hnich 2009] used *neuroevolution*: *Evolved Parameterised Policies* (EPP) — 1st incomplete method for SCP, beat all previous methods on random SCSPs note SSAT is related to SCP, & SSAT algorithms may be:

- *systematic* (based on DPLL) or
- *approximation* or
- *non-systematic*

but SSAT does not allow multiple chance constraints

EPP

EPP uses an EA to find an ANN (hence "neuroevolution") whose input is a representation of a policy tree node, & 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, & can be used to represent or recreate a policy tree

the EA fitness function penalises chance constraint violations, & is optimal for ANNs representing satisfying policy trees

on random SCSPs, EPP was orders of magnitude faster than state-of-the-art complete algorithms

this work

EPP treats hard constraints in the same way as chance constraints: not incorrect, but a problem with many hard constraints may need a complex ANN & longer run times

in FEPP the ANN output computes a *recommended value* for a decision variable, not necessarily the value assigned:

• as we assign values to decision & stochastic variables under some scenario ω, we use hard constraints to filter decision *& stochastic* variable domains

• if domain wipe-out occurs on *any* variable then we stop assigning variables under ω, & every constraint is artificially considered to be violated in ω; else continue

• to assign a stochastic variable s we choose $\omega(s)$, but if it has been removed from dom(s) then we stop assigning variables under ω & every constraint h is artificially considered to be violated in ω; else continue

• on assigning a decision variable x we compute the recommended value by ANN, then choose the first remaining domain value after it in cyclic order

clarifications

• if we filter a *stochastic* variable domain, might this force the variable to depart from the scenario?

no if the scenario value has been pruned then it is not assigned & a penalty is imposed

• if we avoid choosing a value for a decision variable because it wipes out the domain of a *later* stochastic variable via filtering, doesn't this violate the stage structure?

no filtering makes no assumptions on the values of unassigned variables, it can only tell us that assigning a value to a decision variable will *inevitably* violate a hard constraint

• we consider all constraints to be violated on domain wipe-out or if a selected value for a stochastic variable has been filtered: does this make the fitness function incorrect?

no both cases correspond to hard constraint violations, & considering constraints to be violated in this way is similar to using a penalty function in a GA or LS algorithm: it only affects fitness for non-solutions

so FEPP is similar to EPP but it uses filtering on hard constraints, & if the ANN recommends a filtered value then FEPP looks for another value

we now state some properties of FEPP

property 1

FEPP can learn more policies than EPP with a given ANN

Proof sketch any policy that can be learned by EPP with a given ANN can also be learned by FEPP with the same ANN: the EPP-evolved ANN always makes correct decisions, which FEPP tries first & they succeed

conversely, $\exists$ an SCSP that can be solved by FEPP but not by EPP with a given ANN:

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

let the ANN be a perceptron whose inputs are the s & t values & whose output is used to select a domain value for x, let FEPP enforce AC

a perceptron can't learn the $\oplus$ (XOR) function [Minsky & Papert 1962] so EPP can't solve the SCSP

but AC removes the incorrect value from dom(x) so FEPP succeeds

property 2

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

Proof sketch any policy that can be learned by FEPP with a given ANN & filtering algorithm $\mathcal{A}$ can also be learned by FEPP with a stronger filtering algorithm $\mathcal{B}$: easy to show

conversely, $\exists$ an SCSP, an ANN, & filtering algorithms $\mathcal{A}$ & $\mathcal{B}$, st the SCSP can be solved with $\mathcal{B}$ but not $\mathcal{A}$:

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:
$V_1 = \emptyset \quad S_1 = \{s, t\}$
$V_2 = \{x\} \quad S_2 = \{u\}$
$V_3 = \{y\} \quad S_3 = \emptyset$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle, \langle V_3, S_3\rangle]$

let $\mathcal{A}$ enforce pairwise AC on the $\neq$-constraints for c_2, $\mathcal{B}$ enforce GAC on c_2, both enforce AC on c_1, & use a perceptron

$\mathcal{B}$ reduces dom(x) to $\{0, 1\}$ before search, so $\oplus$ is immediately enforced

$\mathcal{A}$ can't do this so FEPP must learn $\oplus$: impossible for a perceptron

property 3

the optimisation problem representing an SCSP has more solutions under FEPP than EPP, with a given ANN

(a *solution* here is a set of ANN weights representing a satisfying policy tree for the SCSP)

Proof sketch any EPP solution is also a FEPP solution

conversely, $\exists$ an SCSP with an FEPP solution that is not an EPP solution, using a given ANN & filtering algorithm:

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

let the ANN be the perceptron

$$x = \begin{cases} 1 \text{ if } w_1 s + w_2 > 0 \\ 0 \text{ otherwise} \end{cases}$$

with weights $w_1 = 2$ & $w_2 = 0$, & let FEPP use AC

this is not an EPP solution, because if $s = 1$ then $x = 1$ which violates the disequality constraint

but it is an FEPP solution, because AC removes the assigned s value from dom(x) (for any weights)

property 4

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

Proof sketch any FEPP solution with a given ANN & filtering algorithm $\mathcal{A}$ is also a solution under a stronger filtering algorithm $\mathcal{B}$

conversely, $\exists$ an SCSP, ANN & filtering algorithms $\mathcal{A}, \mathcal{B}$ with $\mathcal{B}$ stronger than $\mathcal{A}$, with a solution under $\mathcal{B}$ that is not a solution under $\mathcal{A}$:

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

let $\mathcal{A}$ enforce pairwise-AC on alldifferent, $\mathcal{B}$ enforce GAC on it, & use the same perceptron as above; assume x is assigned before y

the ANN is a solution under $\mathcal{B}$ because GAC removes 0 & 1 from dom(x), so x will be set to the cyclically next value 2

then s is assigned $\omega(s) \in \{0, 1\}$ & GAC removes $\omega(s)$ from dom(y)

so y is assigned $1 - \omega(s)$

but it is not a solution under $\mathcal{A}$ because pairwise-AC does not remove 0 & 1 from dom(x), so x will follow the perceptron recommendation & be assigned value 0, which can't lead to a solution

summary

these properties show that:

• unlike EPP, FEPP can exploit advanced CP techniques (global constraints, stronger filtering, etc) to correct *some* poor ANN decisions, & use a simpler ANN to solve a given SCSP

• FEPP may be more efficient than EPP because it solves an optimisation problem with more solutions — not guaranteed to reduce runtime but it may do

from the EA point of view, filtering is used as a *partial decoder*: the stronger the filtering the more complete the decoder

experiments: QBF

We test 2 hypotheses:

• does filtering *really* enable an ANN to learn more complex policies?

• where a policy can be learned without filtering, does filtering *really* speed up learning?

we use QBF instances: closely related to SCP (there is a simple mapping from QBF to SSAT, which is a special case of SCSP) & has only hard constraints

we implemented FEPP with a weak form of filtering (*backchecking*) & a *periodic perceptron* (as in EPP)

but recently we got better results with a hash function with H entries instead of an ANN (hence H genes) so I'll present these results (the above 4 properties also apply to hash functions or other approximation functions)

instance	N	EPP sec	EPP H^*	FEPP sec	FEPP H^*
cnt01	18	0.07	232	0.03	25
impl02	17	0.0	1	0.01	1
impl04	77	0.01	1	0.12	1
impl06	317	0.03	1	0.96	1
impl08	1277	0.17	1	6.2	1
TOILET2.1.iv.4	60	—	—	3.3	60
toilet_a_02_01.4	92	—	—	2.4	106
tree-exa10-10	10240	—	—	4.0	1

times are medians of 30 runs

"—" means "unsolved"

N is #policy tree nodes

H^* is smallest H able to solve problem results support both hypotheses:

- some problems can be solved by FEPP with smaller H than with EPP

- where both can solve a problem EPP beats FEPP (perhaps due to filtering overhead) but harder problems were solved by FEPP & not EPP

interesting result some QBF problems can be solved with very small H, eg 1!

experiments: random SCSPs

we experimented further with the random 4-stage SCSPs from our EPP paper, with 259 policy tree nodes

(no hard constraints so FEPP=EPP here)

using a hash function instead of an ANN we can now solve *all* instances for the first time (no complete method managed this)

we consistently found that less-constrained problems can be compressed further, some of the least-constrained to $H = 1$

experiments: deterministic JSP

can deterministic problem solutions also be compressed?

we implemented FEPP in Eclipse (a constraint programming system) & expressed JSP benchmarks as SCOPs

we used a standard Eclipse model with cumulative constraints, which have a stronger & a weaker filtering algorithm

using strong filtering FEPP solves `ft06` with $H = 1$, `abz6` with $H = 3$, & `ft10` with $H = 20$

using weak filtering: `ft06` with $H = 1$ (again) & `abz6` with $H = 5$ (no result for `ft10` yet)

discussion

we conjecture that underconstrained problems (CSP, SCSP, QBF, etc) have many solutions, including *some* with simple structures that can be compressed

filtering allows further compression, & compression can speed up search

a solution strategy: try small H first

but as $H \longrightarrow H^*$ runtime can increase, perhaps because there are fewer satisfying policies with simple structure

conclusion

EPP was the first incomplete SCP algorithm, & beat complete algorithms on some problems

FEPP adds filtering on hard constraints & beats EPP

it can be applied to stochastic, quantified & deterministic problems

results so far are preliminary but promising: much work to do!

solution compression, along with scenario sampling, may be a key to finding good solutions to very large problems

FEPP is being applied to *risk management* in a project with IBM: a new application of SCP