Evolving Parameterised Policies for Stochastic Constraint Programming

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

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

overview

Stochastic Constraint Programming (SCP): an extension of CP for modelling and solving combinatorial problems involving uncertainty [Benoist, Bourreau, Caseau & Rottembourg 2001] [Walsh 2002]

a solution is a *policy tree* specifying decision variable assignments in each scenario

several solution methods have been proposed, none practical for *large, multi-stage* problems

our incomplete approach: specify a policy tree *indirectly* via a parameterised function, whose parameter values are found by evolutionary search

will show that this can be orders of magnitude faster than a state-of-the-art scenario-based approach

it also provides a very compact representation of policy trees

introduction

m-stage SCSP: a tuple $(V, S, D, P, C, \ominus, L)$ where

- V are decision variables
- S are stochastic variables
- D maps each $x \in V \cup S$ to a domain
- P maps each $s \in S$ to a prob. dist.
- C is constraints on $V \cup S$
- $\ominus$ maps each $h \in C$ to a threshold value $\theta \in (0, 1]$
- $L = (\langle V_1, S_1 \rangle, \ldots, \langle V_m, S_m \rangle)$ are *decision stages*

each constraint must contain at least one V variable

an $h \in C$ containing only V variables is a *hard constraint* with $\Theta(h) = 1$

an $h \in C$ containing at least one S variable is a *chance constraint*

to solve an m-stage SCSP an assignment to the V_1 must be found s.t. given random values for S_1 assignments can be found for V_2 s.t. given random values for $S_2, \ldots$ assignments can be found for V_m s.t. given random values for S_m, the hard constraints are each satisfied and the chance constraints are satisfied in the specified fraction of all possible *scenarios* (set of values for the stochastic variables)

policy trees

a useful concept: a *policy tree* of decisions

each node represents a value chosen for a decision variable

each arc 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 one in which each chance constraint is satisfied with respect to the tree

$h \in C$ 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

example

consider a 2-stage SCSP with $V_1 = \{x_1\}$, $S_1 = \{s_1\}$, $V_2 = \{x_2\}$, $S_2 = \{s_2\}$, dom$(x_1) = [1, 4]$, dom$(x_2) = [3, 6]$, dom$(s_1) = [4, 5]$, dom$(s_2) = [3, 4]$, all stochastic domain values with probability 0.5, and 2 chance constraints

$$(c_1)\text{: } s_1 x_1 + s_2 x_2 \geq 30$$

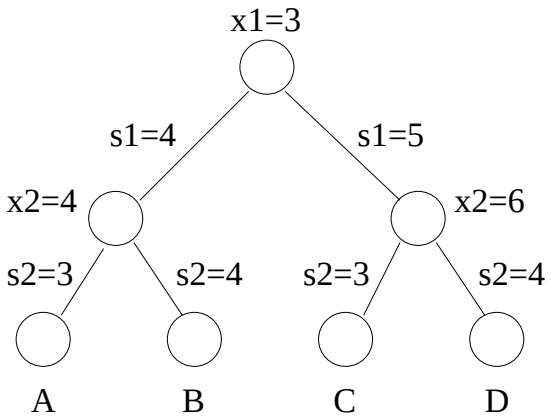
and

$$(c_2)\text{: } s_2 x_1 = 12$$

with $\theta_{c_1} = 0.75$ and $\theta_{c_2} = 0.5$

x_1 must be set to a unique value, but the value of x_2 depends on that of s_1

A policy for this problem is:



notice that it is in the form of a tree

the 4 scenarios A, B, C and D each have probability 0.25

c_1 is satisfied in A, C and D so with probability 0.75; c_2 is satisfied in A and C so with probability 0.5; these satisfy the thresholds $\theta_{c_1}, \theta_{c_2}$ so this is a satisfying policy

no practical way of solving large multi-stage SCSPs has yet been proposed

the design of local search algorithms for SCP was suggested by [Walsh 2002] to improve scalability; but it does not seem to have been pursued, and it does not address the problem of representing large policy trees

we propose a new approach called Evolved Parameterised Policies (EPP): use an evolutionary algorithm (EA) to choose parameter values for a parameterised function that indirectly specifies a policy tree

EPP is an incomplete SCP algorithm that should scale well in 2 ways: a simple parameterised function can be used to represent a large policy tree, and an EA can handle problems with many decision variables

EPP

instead of explicitly representing a policy tree we use a parameterised function $\tau_{\underline{w}}$, whose input is the current stochastic variable assignments and a decision variable, and whose output is a domain value for that variable

its parameters $\underline{w} = (w_1, w_2 \ldots)$ are real-valued numbers which we call *weights*

$\tau_{\underline{w}}$ completely defines the policy tree

if $\tau_{\underline{w}}$ does not require an exponential number of weights then it avoids the memory problem associated with large trees

this is a form of compression, and some trees may be incompressible, but we hope that relatively simple functions will suffice for most problems of interest

the method is as follows

we'll consider only SCSPs whose decision and stochastic variable domains are intervals $[L, U]$ of integers (easily generalises to other domains)

assume a fixed ordering of the problem variables (any ordering that conforms to the stage structure will do)

first compute an affine function

$$\alpha_{\underline{w}}(S, x_j) = w_j + \sum_{i \in \sigma_j} w_i s_i$$

where σ_j is the set of indices of the stochastic variables S that precede decision variable x_j

this is the simplest possible function that involves all relevant stochastic variables; may not suffice for all SCSPs, but it needs only a linear number of weights and works well in experiments so far

w_j is necessary because in the case of a decision variable x_j that is not preceded by any stochastic variable (so that $\sigma_j = \emptyset$) we require a default value

in the special cases of a (deterministic) CSP or a 1-stage SCSP, no decision variable is preceded by a stochastic variable, so the policy is simply a weight w_j for each decision variable x_j

note that the stochastic variables S_m (those in the final stage) do not precede any decision variables, and therefore do not appear in $\underline{w}$

but the value of $\alpha_{\underline{w}}(S, x_j)$ is a real number and not a domain value

to obtain an integer in $[L, U]$ it is discretised by truncation, then modular arithmetic is used to obtain an integer in the required range:

$$\tau_{\underline{w}}(S, x_j) = L + (\lfloor \alpha_{\underline{w}}(S, x_j) \rfloor \bmod [U - L + 1])$$

this gave better results than the more obvious

$$\tau_{\underline{w}}(\underline{x}) = \min(U, \max(L, \lfloor \underline{x} \cdot \underline{w} \rfloor))$$

now $\underline{w}$ defines a policy: for each decision variable x_j we choose its value to be $\tau_{\underline{w}}(S, x_j)$

example

consider a 3-stage problem: $V_1 = \{x_1, x_2\}$, $S_1 = \{s_1, s_2\}$, $V_2 = \{x_3, x_4\}$, $S_2 = \{s_3, s_4\}$, $V_3 = \{x_5, x_6\}$, $S_3 = \{s_5, s_6\}$

suppose we wish to find the value of x_3 given that $s_1 = 5$, $s_2 = 7$, dom$(x_3) = [5, 10]$ and our policy is specified by a weight vector

$$\underline{w} = (0.1, 5.3, 7.1, 9.9, 8.7, 4.1, -0.6, 5.5, -5.2, 2.9)$$

($\underline{w}$ has 10 components though there are 12 variables: the S_3 do not precede any decision variables)

then

$$\alpha_{\underline{w}}(S, x_3) = 8.7 + 7.1s_1 + 9.9s_2 = 113.5$$

and

$$\tau_{\underline{w}}(S, x_3) = 5 + (113 \bmod 6) = 5 + 5 = 10$$

so under the policy defined by $\underline{w}$, variable x_3 is set to 10 under any scenario in which $s_1 = 5$ and $s_2 = 7$

search space & objective function

we have defined the form of our policies, but how should we search for them and solve an SCSP?

the state space to be explored is $\mathbf{R}^k$ representing the space of real-valued weight vectors $\underline{w}$, where k is the total number of SCP variables not counting those in S_m

to handle the constraints we use *penalty functions* to obtain an unconstrained optimisation problem: a standard technique that penalises constraint violations, commonly used when applying genetic algorithms or local search to CSPs

the objective function to be minimised is

$$\Phi(\underline{w}) = \sum_{h \in C} \phi(h, \underline{w})$$

where the penalty functions are

$$\phi(h, \underline{w}) = \begin{cases} 0 & \text{if } \pi_h(\underline{w}) \geq \Theta(h) \\ \Theta(h) - \pi_h(\underline{w}) & \text{if } \pi_h(\underline{w}) < \Theta(h) \end{cases}$$

and $\pi_h(\underline{w})$ is the probability that that h is satisfied under the policy defined by $\underline{w}$

any policy defined by $\underline{w}$ such that $\Phi(\underline{w}) = 0$ is clearly a satisfying policy

search algorithm

we now have a search space and an objective function, so we can apply an EA (or local search) to solve the SCSP

we describe the EA only briefly because our emphasis is on showing the feasibility of EPP

it's a cellular evolution strategy with Cauchy mutation, plus some additional mutation heuristics designed for this application

each chromosome is a weight vector $\underline{w}$

for each chromosome we compute its fitness $\Phi(\underline{w})$ (fitness is conventionally maximised but we minimise Φ)

to compute the $\pi_h(\underline{w})$ we check every leaf node in the implied policy tree via *resampling*: the more promising a chromosome is, the more leaves are checked using it

the probability associated with a leaf ℓ is the product of the probabilities associated with the stochastic variable assignments in the arcs of the path leading to ℓ

at each leaf a chance constraint $h \in C$ is either satisfied or violated

by summing the probabilities of the leaves at which h is satisfied we obtain the probability $\pi_h(\underline{w})$ that h is satisfied under the policy defined by $\underline{w}$

(for problems with many stages, the $\pi_h(\underline{w})$ can also be estimated by sampling the leaves using any of the scenario reduction techniques used in [Tarim, Manandhar, Walsh 2006]; we can sample many leaves because we do not use them to derive a CSP: we use over 1000 scenarios below)

EPP transforms a multi-stage SCSP into a noisy numerical optimisation problem ("noisy" : the objective function must be averaged over many scenarios)

experiments

we now show empirically that it is possible to find a satisfying policy using EPP

we use a benchmark set (from the previous talk) of random SCSPs with 5 chance constraints over 4 decision variables $x_1 \ldots x_4$ and 8 stochastic variables $s_1 \ldots s_8$

decision variable domains are $\mathrm{dom}(x_1) = [5, 10]$, $\mathrm{dom}(x_2) = [4, 10]$, $\mathrm{dom}(x_3) = [3, 10]$, $\mathrm{dom}(x_4) = [6, 10]$

stochastic variable domains of s_1, s_3, s_5, s_7 contain 2 values while those of s_2, s_4, s_6, s_8 contain 3 values (in both cases the values are chosen randomly from the discrete interval $[1, 5]$ and have equal probabilities)

chance constraints:

$$\begin{array}{rcl} x_1s_1 + x_2s_2 + x_3s_3 + x_4s_4 & = & 80 \ (\theta = \alpha) \\ x_1s_5 + x_2s_6 + x_3s_7 + x_4s_8 & \leq & 100 \ (\theta = \beta) \\ x_1s_5 + x_2s_6 + x_3s_7 + x_4s_8 & \geq & 60 \ (\theta = \beta) \\ x_1s_2 + x_3s_6 & \geq & 30 \ (\theta = 0.7) \\ x_2s_4 + x_4s_8 & = & 20 \ (\theta = 0.05) \end{array}$$

where $\alpha \in \{0.005, 0.01, 0.03, 0.05, 0.07, 0.1\}$ and $\beta \in \{0.6, 0.7, 0.8\}$

the problems are 4-stage: $V_1 = \{x_1\}$, $S_1 = \{s_1, s_5\}$, $V_2 = \{x_2\}$, $S_2 = \{s_2, s_6\}$, $V_3 = \{x_3\}$, $S_3 = \{s_3, s_7\}$, $V_4 = \{x_4\}$ and $S_4 = \{s_4, s_8\}$

in total we have 6 α-values and 3 β-values, and we randomly generate 5 different sets of stochastic variable domains, giving 90 instances in total

we compare the scenario-based approach (SBA) of [Tarim, Manandhar, Walsh 2006] with EPP ("—" is $>$ 200 sec, EPP figures are medians over 30 runs)

SBA transforms them into deterministic CSPs with 6739 variables and 6485 constraints: EPP transforms them into unconstrained noisy optimisation problems with 10 real-valued variables

these small-looking SCSPs are non-trivial!

problem set 1			
α	β	SBA	EPP
0.05	0.6	—	0.5
0.10	0.6	—	1.0
0.12	0.6	—	0.9
0.15	0.6	—	1.4
0.17	0.6	—	1.7
0.20	0.6	—	1.6
0.05	0.7	—	1.3
0.10	0.7	—	1.2
0.12	0.7	—	1.3
0.15	0.7	—	1.9
0.17	0.7	—	2.7
0.20	0.7	—	2.8
0.05	0.8	—	12
0.10	0.8	—	9.4
0.12	0.8	—	11
0.15	0.8	—	12
0.17	0.8	—	15
0.20	0.8	—	13

problem set 2			
α	β	SBA	EPP
0.05	0.6	—	1.6
0.10	0.6	—	4.8
0.12	0.6	—	14
0.15	0.6	—	15
0.17	0.6	—	118
0.20	0.6	—	—
0.05	0.7	—	1.7
0.10	0.7	—	4.8
0.12	0.7	—	16
0.15	0.7	—	16
0.17	0.7	—	144
0.20	0.7	—	—
0.05	0.8	—	2.7
0.10	0.8	—	7.1
0.12	0.8	—	20
0.15	0.8	—	13
0.17	0.8	—	—
0.20	0.8	—	—

problem set 3			
α	β	SBA	EPP
0.05	0.6	0.7	0.4
0.10	0.6	0.5	3.1
0.12	0.6	0.5	3.1
0.15	0.6	—	15
0.17	0.6	—	14
0.20	0.6	—	49
0.05	0.7	0.6	2.5
0.10	0.7	0.7	9.1
0.12	0.7	0.6	12
0.15	0.7	—	27
0.17	0.7	—	46
0.20	0.7	—	159
0.05	0.8	0.8	9.7
0.10	0.8	0.6	17
0.12	0.8	0.6	29
0.15	0.8	—	58
0.17	0.8	—	109
0.20	0.8	—	—

problem set 4			
α	β	SBA	EPP
0.05	0.6	—	4.2
0.10	0.6	—	—
0.12	0.6	—	—
0.15	0.6	—	—
0.17	0.6	—	—
0.20	0.6	—	—
0.05	0.7	—	4.9
0.10	0.7	—	—
0.12	0.7	—	—
0.15	0.7	—	—
0.17	0.7	—	—
0.20	0.7	—	—
0.05	0.8	—	7.5
0.10	0.8	—	—
0.12	0.8	—	—
0.15	0.8	—	—
0.17	0.8	—	—
0.20	0.8	—	—

problem set 5			
α	β	SBA	EPP
0.05	0.6	—	0.1
0.10	0.6	—	0.5
0.12	0.6	—	0.7
0.15	0.6	—	0.8
0.17	0.6	—	2.2
0.20	0.6	—	1.9
0.05	0.7	0.2	0.1
0.10	0.7	—	0.4
0.12	0.7	—	0.7
0.15	0.7	—	0.8
0.17	0.7	—	1.8
0.20	0.7	—	3.3
0.05	0.8	—	0.2
0.10	0.8	—	0.9
0.12	0.8	—	0.8
0.15	0.8	—	1.2
0.17	0.8	—	1.6
0.20	0.8	—	3.3

clear pattern:

- EPP solved every problem that SBA solved plus many more
- where SBA solved a problem it was up to 48 times faster than EPP
- but in some cases EPP was at least 2000 times faster
- EPP is on average much faster than SBA

where SBA and EPP both failed to solve an instance, the instance might be infeasible: we don't know in most cases

but in a few cases both SBA and EPP failed to solve a feasible instance (verified by GCC, previous talk) so there is room for improvement

perhaps our parameterised policy space does not contain satisfying policies for these problems, and that a more complex parameterised function is required

new results we have experimented with nonlinear functions and solved **all but one** of these instances quite easily

related work

several SCSP solution methods have been proposed

[Walsh 2002] presented 2 complete algorithms based on backtracking and forward checking, and suggested some approximation procedures

[Balafoutis, Stergiou 2006] describe an arc-consistency algorithm

[Bordeaux, Samulowitz 2007] modify a standard backtracking algorithm to one that can handle multiple chance constraints and uses polynomial space, but is inefficient in time

[Rossi, Tarim, Hnich, Prestwich 2008] propose a cost-based filtering technique

for the special case of SCP with linear recourse [Tarim, Miguel 2006] propose a Bender's decomposition algorithm

[Tarim, Manandhar, Walsh 2006] transform an SCSP into a *deterministic equivalent* CSP and solve it by standard CP methods

extended to handle multiple chance constraints and multiple objective functions

gives much better performance on some problems than tree search methods

to reduce the size of the CSP *scenario reduction* methods are proposed, as used in Stochastic Programming, which choose a small but representative set of scenarios, but:

- it might not always be possible to find a small representative set of scenarios

- in some cases choosing an inappropriate set of scenarios can yield an unsolvable CSP

- using even a modest number of scenarios leads to a CSP that is several times larger than the original SCSP

related work: SSAT

Stochastic Boolean Satisfiability (SSAT) is related to SCP: a recent survey is given in [Majercik 2009]

an SSAT problem: as an SCSP with Boolean variables, extensional non-binary constraints, and all constraints are treated as 1 chance constraint

(so EPP applies immediately to SSAT)

SSAT algorithms fall into three classes:

systematic algorithms are based on the standard SAT backtracking algorithm (DPLL) and correspond roughly to some current SCP algorithms

approximation algorithms work well on restricted forms of SSAT but less well on general SSAT problems

eg APPSSAT considers scenarios in decreasing order of probability to construct a partial tree

does not work well when all scenarios have similar probability

non-systematic algorithms such as `randevalssat` which applies local search to the decision (existential) variables in a random set of scenarios

also suffers from memory problems because it must build a partial tree

related work: machine learning

EPP is closely related to a ML method that has been used for many optimisation problems involving uncertainty: *neuroevolution* (NE)

in NE the parameterised function is an ANN whose parameters are the network weights, which are found by an EA

unlike our simple function, ANNs are *universal function approximators* which can in principle approximate any policy

NE has been applied to very challenging control problems with good results

also used for learning to play games such as Backgammon, Go, Checkers & Chess

these successes indicate that EPP might work well on real-world SCP problems that are too large to solve by complete methods

conclusion

EPP does not suffer from the memory problems of most methods and does not introduce a large number of new variables

it is the first incomplete algorithm for SCP

in experiments EPP was several orders of magnitude faster than the current best (complete) method

it does not exploit constraint filtering but this could perhaps be used to handle hard constraints

EPP will also require slight modification for handling variable domains that contain arbitrary integers or real numbers, and for handling problems with objective functions

we will explore these issues and test EPP on more interesting SCP problems

stop press

Quantified Boolean Formulae (QBF) can be modelled as SCSPs: all thresholds are 1, all value probabilities are nonzero, and all scenarios must be sampled

we tested EPP on a few QBF benchmarks

on Rintanen's `impl` problems EPP beats the worst reported QBF backtrackers, though it is much worse than the best