A Cultural Algorithm for POMDPs from Stochastic Inventory Control

S. Prestwich, S. A. Tarim, R. Rossi, B. Hnich

4C/UCC, Cork
Izmir University of Economics, Turkey
Hacettepe University, Turkey

(HM'08)

introduction

POMDPs occur in many fields of AI

Reinforcement Learning (RL) algorithms and Evolutionary Computation (EC) algorithms are competing approaches to solving them

a powerful form of EC that has not previously been applied to POMDPs is the *cultural algorithm* (CA): evolving agents share knowledge in a *belief space* used to guide their evolution

we describe a CA for POMDPs that hybridises SARSA with a noisy GA that inherits EC convergence properties: CURL (*CUltural RL*)

CURL's belief space is a set of RL state-action values $Q(s, a)$ updated during EC exploration, and conversely used to modify chromosomes

we use it to solve problems from stochastic inventory control by finding memoryless policies for nondeterministic POMDPs

in experiments, neither SARSA nor the GA dominates, but the CA beats the GA, and SARSA on highly non-Markovian instances

POMDPs

MDPs can model sequential decision-making in situations where outcomes are partly random and partly under the control of the agent

their states have the *Markov property*: if the current state of the MDP at time t is known, transitions to a new state at time $t+1$ are independent of all previous states

MDPs can be solved in polynomial time by modelling them as linear programs, but these are hard to solve in practice

without the Markov property we have a POMDP which in general is computationally intractable

this can be caused by partial knowledge, eg a robot must often navigate using only partial knowledge of its environment

machine maintenance and planning under uncertainty can also be modelled as POMDPs when solving a POMDP we must find a *policy*: a strategy for selecting actions based on observations that maximises a function of the rewards, eg total reward

a policy is a function that maps the agent's observation history and its current internal state to an action

a policy may be *deterministic* or *probabilistic*: a deterministic policy consistently chooses the same action when faced with the same information, while a probabilistic policy might not

a *memoryless* (or *reactive*) policy returns an action based solely on the current observation

the problem of finding a memoryless policy for a POMDP is NP-complete and exact algorithms are very inefficient, but there are good inexact methods

RL methods

TD learning algorithms such as Q-Learning & SARSA are a standard way of finding good policies

by Monte Carlo-like simulations they compute a *state-action value* function $Q : S \times A \rightarrow \Re$ which estimates the expected total reward for taking a given action from a given state (some compute a *state value* function $V : S \rightarrow \Re$)

SARSA(λ) usually outperforms two versions of Q-learning with eligibility trace [Sutton & Barto] we take SARSA(λ) with ϵ-greedy behaviour as a representative RL algorithm

RL algorithms have convergence proofs for MDPs, but for some POMDPs applications they still perform well, especially when augmented with an *eligibility trace*

SARSA(0) is equivalent to SARSA, SARSA(1) is equivalent to a Monte Carlo algorithm, $0 < \lambda < 1$ is a hybrid and often better than both

EC methods

these are an alternative approach to POMDPs, and sometimes beat RL algorithms on highly non-Markovian problems

we use the obvious *table-based representation*: each chromosome represents a policy, each gene a state, and each allele an action

our GA is based on the GENITOR algorithm (steady-state, elitist, replaces least-fit chromosome by new one) with uniform crossover applied with a probability p_c (if not applied then 1 parent is selected and mutated)

EC methods

mutation is applied to a chromosome once with probability p_m, twice with probability p_m^2, three times with probability p_m^3...

nondeterminism in the POMDP causes noise in the GA's fitness function, so we average fitness over S samples S: a *Noisy GA* (NGA)

NGAs are usually generational but steady-state NGAs do exist

cultural algorithms

a powerful form of EC: *cultural algorithms* (CAs)

agents share knowledge in a *belief space* to form a consensus (distinct from POMDP *belief space*)

a CA has an *acceptance function* that determines which individuals in the population are allowed to *adjust* the belief space

the beliefs are conversely used to *influence* the evolution of the population

these hybrids of EC and Machine Learning have been shown to converge more quickly than EC alone on several applications

CAs are based on concepts used in sociology and archaeology to model cultural evolution

by pooling knowledge gained by individuals in a body of cultural knowledge, or belief space, convergence rates can sometimes be improved

they have been applied to constrained optimisation, multiobjective optimisation, scheduling and robot soccer, but not to POMDPs, nor have they utilised RL

CURL

we propose a new cultural hybrid of RL and EC for solving POMDPs: *CUltural Reinforcement Learning* (CURL)

it is straightforward and can be applied to different RL and EC algorithms

1 set of RL state-action values $Q(s, a)$ is initialised as in the RL algorithm, and are the CA belief space

a population is initialised as in the EC algorithm

the EC algorithm is executed as usual, but each new chromosome is altered by, and used to alter, the $Q(s, a)$

on generating a new chromosome we replace, with some probability p_l, each allele by the corresponding greedy action given by the modified $Q(s, a)$ values setting $p_l = 0$ prevents any learning, and CURL reduces to the EC algorithm

setting $p_l = 1$ always updates a gene to the corresponding $Q(s, a)$ value, and CURL reduces to SARSA(λ) without exploration

we treat the modified chromosome as in the EC algorithm: fitness evaluation and placement into the population

during fitness evaluation the $Q(s, a)$ are updated by bootstrapping as usual in the RL algorithm, but the policy followed is that specified by the modified chromosome thus in CURL, as in several other CAs, *all* chromosomes are allowed to adjust the belief space

there is no ϵ parameter in CURL because exploratory moves are provided by EC

we combine the GENITOR-based NGA with SARSA(λ) for our CURL implementation

fitness is averaged over S samples (for a deterministic POMDP only one sample is needed to obtain the fitness of a chromosome, so we can set $S = 1$ to obtain a CURL hybrid of SARSA(λ) and GENITOR)

CURL pseudo-code

```
CURL($S$,$P$,$p_c$,$p_m$,$\alpha$,$\lambda$,$p_l$):
(   create population of size $P$
    evaluate population using $S$ samples
    initialise the $Q(s,a)$
    while not(termination condition)
    (   generate an offspring O using $p_c, p_m$
        update O using $p_l$ and the $Q(s,a)$
        call SARSA($\lambda,\alpha$,O) $S$ times to estimate
              O fitness and bootstrap the $Q(s,a)$
        replace least-fit chromosome by O
    )
    output fittest chromosome
)
```

CURL convergence

for POMDPS, unlike MDPs, suboptimal policies can form local optima in policy space

this motivates the use of global search techniques such as EC, which are less likely to become trapped in local optima

hill-climbing has been combined with GAs to form *memetic algorithms* with faster convergence than a pure GA, and this was a motivation for CURL's design

but if bootstrapping is used then optimal policies are not necessarily stable: that is, an optimal policy might not attract the algorithm

so a hybrid might not be able to find an optimal policy even if it escapes all local optima luckily, if $p_l < 1$ and the underlying EC algorithm is convergent then so is CURL: if $p_l < 1$ then there is a non-zero probability that no allele is modified by the $Q(s, a)$, in which case CURL behaves exactly like the EC algorithm (this is not true of all hybrids)

our GA is convergent (modulo averaging) because every gene in a new chromosome can potentially be mutated to an arbitrary allele, so the CURL instantiation is convergent

note

we should now evaluate CURL on standard POMDPs

but this work is motivated by the need to solve large, complex *inventory control* (IC) problems that do not succumb to more traditional methods

we know of no method in the IC literature that can optimally solve our problem in a reasonable time

so we test CURL on POMDPs from stochastic IC: we believe that the problem we tackle has not previously been considered as a POMDP

stochastic IC

the problem we tackle is as follows:

given a planning horizon of N periods and a demand for each period $t \in \{1, \ldots, N\}$, which is a random variable with a given probability density function (assumed to be normal)

demands occur instantaneously at the beginning of each time period and are *non-stationary* (may vary from period to period), and demands in different periods are independent

costs...

- a fixed delivery cost a is incurred for each order (even an order of 0)

- a linear holding cost h is incurred for each product unit carried in stock from one period to the next

- a linear stockout cost s is incurred for each period in which the net inventory is negative (it is not possible to sell back excess items to the vendor at the end of a period)

we must find a *replenishment plan* that minimizes expected total cost over the planning horizon

policies

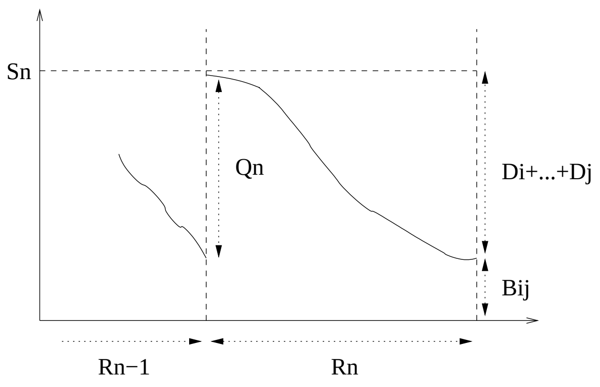
different inventory control *policies* can be adopted to cope with this and other problems

a policy states the rules used to decide when orders are to be placed and how to compute the replenishment lot-size for each order, eg the *replenishment cycle policy* (R, S)

non-stationary demand so it has the form (R^n, S^n) where R^n denotes the length of the n^{th} replenishment cycle and S^n the order-up-to-level for replenishment: we must populate both sets

the order quantity for replenishment cycle n is determined only after the demand in former periods has been realized the order quantity is computed as the amount of stock required to raise the closing inventory level of replenishment cycle $n-1$ up to level S^n

(R, S) policy yields plans of higher cost than the optimum, but it reduces planning instability, and is appealing when items are ordered from the same supplier or require resource sharing



both RL and EC have been applied to several IC problems, neither seems to have been applied to (R, S)

actually, there are efficient algorithms which find optimal policies (under reasonable simplifying assumptions), but if we complicate the problem (add order capacity constraints, drop the assumption of independent demands...) then these efficient algorithms become unusable

so (R, S) is useful as a representative of a family of more complex problems

POMDP model

(R, S) can be modelled as a POMDP as follows:

- *state*: period n
- *action*: choice of an order-up-to level or the lack of an order (denoted by N)
- *reward* r_n to be minus the total cost incurred in period n
- rewards are *undiscounted* (do not decay with time)
- the problem is *episodic* (has well-defined start and end states)

- the POMDP is *nondeterministic* (the rewards are randomised)

- its solution is a policy that is *deterministic* and *memoryless* (actions are taken solely on the basis of the agent's current observations)

the problem has an underlying MDP: if we include current stock level in the state, we have all the information we need to make an optimal decision

but the (R, S) policy does not make optimal decisions: instead it fixes order-up-to levels independently of the stock level.

this is slightly unusual as a POMDP:

- all actions from a state n lead to the same state $n+1$ (though they have different expected rewards): different actions usually lead to different states

- many applications are non-Markovian because of limited available information, but here we *choose* to make it non-Markovian by discarding information for an application-specific reason: to reduce planning instability but we believe that (R, S) is an ideal benchmark for RL and EC methods: easy to describe and implement, hard to solve optimally, have practical importance, and it turns out that neither RL nor EC dominates

notes:

- we cannot use techniques such as forms of memory (eg a recurrent neural network) because the policy would not then be memoryless, so not (R, S)

- similarly we cannot use stochastic policies, which can be arbitrarily more efficient than deterministic policies

experiments

we compare SARSA(λ), the NGA and CURL on 5 benchmark problems with 120 periods, and 30 possible actions from each state: 29 different order-up-to levels at each period, plus the N no-order option

none of the algorithms find optimal policies within 10^8 simulations so these are challenging (a MIP approach also failed given several hours), but they have a special form so we can find optimal policies

we also generate five additional instances by adding an order capacity constraint to each: this problem is NP-hard and we know of no method that can solve it to optimality in a reasonable time (so we do not know the optimum policies) details:

- for NGA and CURL we use a special mutation operator: mutate a gene to N with 50% probability, otherwise to a random order-up-to level

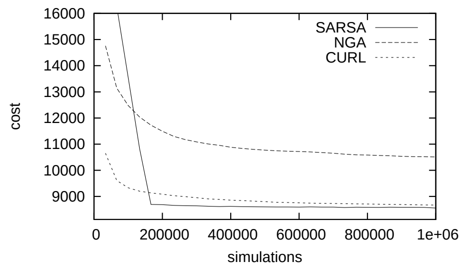
- for SARSA and CURL we initialise all $Q(s, a)$ to the optimistic value of 0, to encourage early exploration

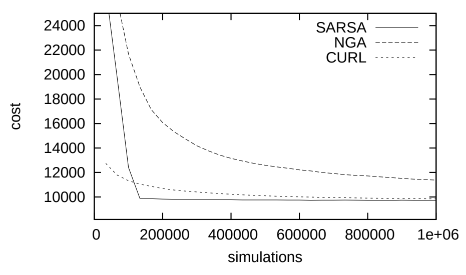
- for SARSA we vary ϵ inversely with time

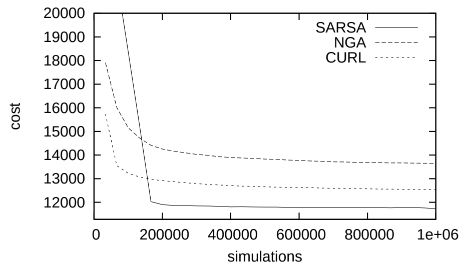
- we tune each algorithm to the middle instance by hill climbing in parameter space

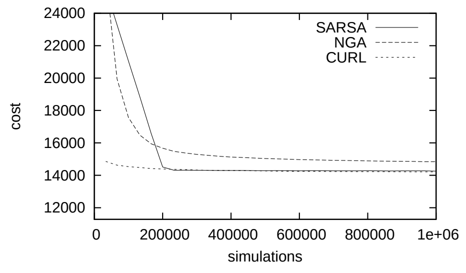
- we set $\lambda = 0$ in SARSA: higher values made little difference

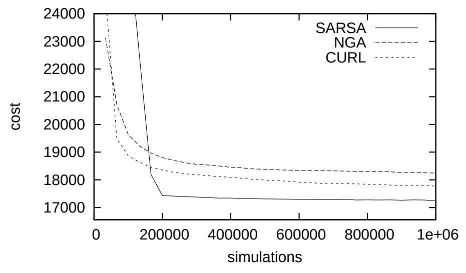
results

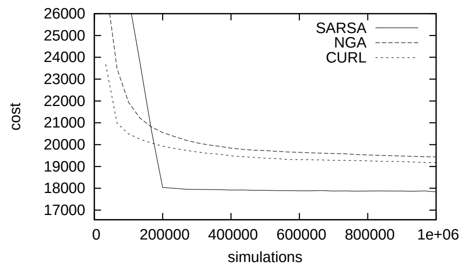

SARSA
NGA
CURL
cost
16000
15000
14000
13000
12000
11000
10000
9000
0
200000
400000
600000
800000
1e+06
simulations

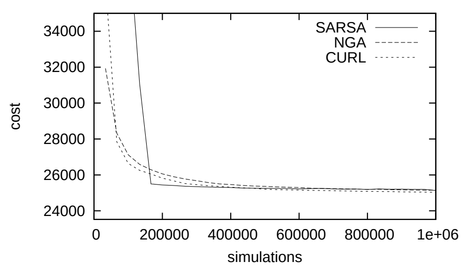

SARSA
NGA
CURL
cost
24000
22000
20000
18000
16000
14000
12000
10000
0
200000
400000
600000
800000
1e+06
simulations

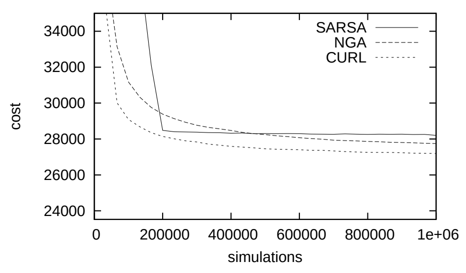

SARSA
NGA
CURL
cost
20000
19000
18000
17000
16000
15000
14000
13000
12000
0
200000
400000
600000
800000
1e+06
simulations

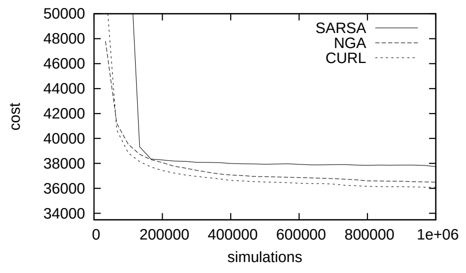

SARSA
NGA
CURL
cost
24000
22000
20000
18000
16000
14000
12000
0
200000
400000
600000
800000
1e+06
simulations

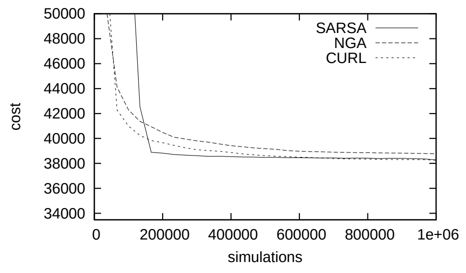

SARSA
NGA
CURL
cost
24000
23000
22000
21000
20000
19000
18000
17000
0
200000
400000
600000
800000
1e+06
simulations


SARSA
NGA
CURL
cost
26000
25000
24000
23000
22000
21000
20000
19000
18000
17000
0
200000
400000
600000
800000
1e+06
simulations


SARSA
NGA
CURL
cost
34000
32000
30000
28000
26000
24000
0
200000
400000
600000
800000
1e+06
simulations


SARSA
NGA
CURL
cost
34000
32000
30000
28000
26000
24000
0
200000
400000
600000
800000
1e+06
simulations


SARSA
NGA
CURL
cost
50000
48000
46000
44000
42000
40000
38000
36000
34000
0
200000
400000
600000
800000
1e+06
simulations


SARSA
NGA
CURL
cost
50000
48000
46000
44000
42000
40000
38000
36000
34000
0
200000
400000
600000
800000
1e+06
simulations

neither SARSA(λ) nor NGA dominates the other, though SARSA(λ) is generally better (this might be caused by our choice of instances)

CURL is uniformly better than NGA, so sometimes better than SARSA(λ)

neither EC nor RL dominates on POMDPs, but EC is better on highly non-Markovian problems, so where NGA beats SARSA(λ) we assume these are highly non-Markovian — so CURL is promising for such problems

(note: adding biased mutation to SARSA(λ) worsens its performance)

related work

several approaches can be seen as hybrids of EC and RL:

- *Learning Classifier Systems* use EC to adapt their representation of the RL problem, and apply RL via the EC fitness function

- *Population-Based RL* uses RL techniques to improve chromosomes, as in a memetic algorithm — similar to CURL but a proposal only

- *GAQ-Learning* uses Q-Learning once only in a preprocessing phase to generate $Q(s,a)$ values, then uses a memetic algorithm with the $Q(s,a)$ values to evaluate the chromosomes

- *Q-Decomposition* combines separate RL agents, and an arbitrator combines their recommendations

- in [Iglesias et al.] a GA & RL are combined to solve a robot navigation problem: apply greedy policy until encountering difficulty; use fittest chromosome to update the $Q(s, a)$ in several RL iterations; use the $Q(s, a)$ to alter chromosomes and get a new population; repeat (convergence is not guaranteed)

conclusion

RL & EC are competing approaches for POMDPs

CURL hybridises them and inherits EC convergence properties

we described new POMDPs from stochastic IC, on which CURL beats EC, and on highly non-Markovian instances beats RL

this work is part of a series of studies in solving IC problems using systematic and randomised methods

future work: develop CURL for more complex inventory problems, and for more standard POMDPs from AI