A Cultural Algorithm for POMDPs from Stochastic Inventory Control

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

[1]Cork Constraint Computation Centre, Ireland
[2]Department of Management, Hacettepe University, Turkey
[3]Faculty of Computer Science, Izmir University of Economics, Turkey
s.prestwich@cs.ucc.ie, armagan.tarim@hacettepe.edu.tr, r.rossi@4c.ucc.ie, brahim.hnich@ieu.edu.tr

Abstract. Reinforcement Learning algorithms such as SARSA with an eligibility trace, and Evolutionary Computation methods such as genetic algorithms, are competing approaches to solving Partially Observable Markov Decision Processes (POMDPs) which occur in many fields of Artificial Intelligence. A powerful form of evolutionary algorithm that has not previously been applied to POMDPs is the cultural algorithm, in which evolving agents share knowledge in a belief space that is used to guide their evolution. We describe a cultural algorithm for POMDPs that hybridises SARSA with a noisy genetic algorithm, and inherits the latter's convergence properties. Its belief space is a common set of state-action values that are updated during genetic exploration, and conversely used to modify chromosomes. We use it to solve problems from stochastic inventory control by finding memoryless policies for nondeterministic POMDPs. Neither SARSA nor the genetic algorithm dominates the other on these problems, but the cultural algorithm outperforms the genetic algorithm, and on highly non-Markovian instances also outperforms SARSA.

1 Introduction

Reinforcement Learning and Evolutionary Computation are competing approaches to solving Partially Observable Markov Decision Processes, which occur in many fields of Artificial Intelligence. In this paper we describe a new hybrid of the two approaches, and apply it to problems in stochastic inventory control. The remainder of this section provides some necessary background information. Section 2 describes our general approach, an instantiation, and convergence results. Section 3 describes and models the problems. Section 4 presents experimental results. Section 5 concludes the paper.

1.1 POMDPs

Markov Decision Processes (MDPs) can model sequential decision-making in situations where outcomes are partly random and partly under the control of the agent. The states of an MDP possess 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 (in the size of their state-space) by modelling them as linear programs, though the order of the polynomials is large enough to make them difficult to solve in practice [14]. If the Markov property is removed then we obtain a Partially Observable Markov Decision Process (POMDP) which in general is computationally intractable. This situation arises in many applications and can be caused by partial knowledge: for example a robot must often navigate using only partial knowledge of its environment. Machine maintenance and planning under uncertainty can also be modelled as POMDPs.

Formally, a POMDP is a tuple $\langle S, A, T, R, O, \Omega \rangle$ where S is a set of states, A a set of actions, Ω a set of observations, $R : S \times A \rightarrow \Re$ a reward function, $T : S \times A \rightarrow \Pi(S)$ a transition function, and $\Pi(\cdot)$ represents the set of discrete probability distributions over a finite set. In each time period t the environment is in some state $s \in S$ and the agent takes an action $a \in A$, which causes a transition to state s' with probability $P(s'|s, a)$, yielding an immediate reward given by R and having an effect on the environment given by T. The agent's decision are based on its observations given by $O : S \times A \rightarrow \Pi(\Omega)$.

When solving a POMDP the aim is to find a *policy*: a strategy for selecting actions based on observations that maximises a function of the rewards, for example the 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 also 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 [12] but there are good inexact methods, some of which we now describe.

1.2 Reinforcement learning methods

Temporal difference learning algorithms such as Q-Learning [32] and SARSA [25] from Reinforcement Learning (RL) are a standard way of finding good policies. While performing 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 RL algorithms compute instead a *state value* function $V : S \rightarrow \Re$.)

The SARSA algorithm is shown in Figure 1. An *episode* is a sequence of states and actions with a first and last state that occur naturally in the problem. On taking an action that leads to a new state, the value of the new state is "backed up" to the state just left (see line 8) by a process called *bootstrapping*. This propagates the effects of later actions to earlier states and is a strength of RL algorithms. (The value γ is a *discounting factor* often used for non-episodic tasks that is not relevant for our application below: we set $\gamma = 1$.) A common *behaviour policy* is ϵ-greedy action selection: with probability ϵ choose a random action, otherwise with probability $1 - \epsilon$ choose the action with highest $Q(s, a)$ value. After a number of episodes the state-action values $Q(s, a)$ are fixed and (if the algorithm converged correctly) describe an optimum policy: from each state choose the action with highest $Q(s, a)$ value. The name SARSA derives from the tuple (s, a, r, s', a').

RL algorithms have convergence proofs that rely on the Markov property but for some non-Markovian applications they still perform well, especially when augmented

```
initialise the $Q(s,a)$ arbitrarily
repeat for each episode
(  $s \leftarrow$ initial state
   choose action $a$ from $s$ using a behaviour policy
   repeat for each step of the episode
   (  take action $a$ and observe $r, s'$
      choose action $a'$ from $s'$ using a behaviour policy
      $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]$
      $s \leftarrow s'$,  $a \leftarrow a'$
   )
)
```

Fig. 1. The SARSA algorithm

with an *eligibility trace* [10, 16] that effectively hybridises them with a Monte Carlo algorithm. We will use a well-known example of such an algorithm: SARSA(λ) [25]. When the parameter λ is 0 SARSA(λ) is equivalent to SARSA, when it is 1 it is equivalent to a Monte Carlo algorithm, and with an intermediate value it is a hybrid and often gives better results than either. Setting $\lambda > 0$ boosts bootstrapping by causing values to be backed up to states before the previous one. (See [30] for a discussion of eligibility traces, their implementation, and the relationship with Monte Carlo algorithms.) There are other more complex RL algorithms (see [13] for example) and it is possible to configure SARSA(λ) differently (for example by using *softmax* action selection instead of ϵ-greedy, and different values of α for each state-action value [30]), but we take SARSA(λ) as a representative of RL approaches to solving POMPDs. (In fact it usually outperforms two versions of Q-learning with eligibility trace — see [30] page 184.)

1.3 Evolutionary computation methods

An alternative approach to POMDPs is the use of Evolutionary Computation (EC) algorithms such as Genetic Algorithms (GAs), which sometimes beat RL algorithms on highly non-Markovian problems [3, 19]. We shall use the most obvious EC model of POMDPs, called a *table-based representation* [19]: each chromosome represents a policy, each gene a state, and each allele (gene value) an action.

The GA we shall use is based on GENITOR [33] but without the refinements of some versions, such as genetic control of the crossover probability. This is a *steady-state* GA that, at each iteration, selects two parent chromosomes, breeds a single offspring, evaluates it, and uses it to replace the least-fit member of the population. Steady-state GAs are an alternative to *generational* GAs that generate an entire generation at each iteration, which replaces the current generation. Maintaining the best chromosomes found so far is an *elitist* strategy that pays off on many problems. Parent selection is random because of the strong selection pressure imposed by replacing the least-fit member. We use standard *uniform crossover* (each offspring gene receives an allele from the corresponding gene in a randomly-chosen parent) applied with a crossover probability p_c: if it is not applied then a single parent is selected and mutated, and the resulting chromosome replaces the least-fit member of the population. Mutation is applied to a chromosome once with probability p_m, twice with probability p_m^2, three times with probability p_m^3, and so on. The population size is P and the initial population contains random alleles.

Nondeterminism in the POMDP causes noise in the GA's fitness function. To handle this noise we adopt the common approach of averaging the fitness over a number of samples S. This technique has been used many times in *Noisy Genetic Algorithms* (NGAs) [4, 6, 17, 18]. NGAs are usually generational and [1] show that elitist algorithms (such as GENITOR) can systematically overvalue chromosomes, but such algorithms have been successful when applied to noisy problems [29]. We choose GENITOR for its simplicity.

1.4 Hybrid methods

Several approaches can be seen as hybrids of EC and RL. *Learning Classifier Systems* [8] use EC to adapt their representation of the RL problem. They apply RL via the EC fitness function. *Population-Based Reinforcement Learning* [11] uses RL techniques to improve chromosomes, as in a memetic algorithm. The paper is an outline only, and no details are given on how RL values are used, nor are experimental results provided. *GAQ-Learning* [15] uses Q-Learning once only in a preprocessing phase, to generate $Q(s, a)$ values. A memetic algorithm is then executed using the $Q(s, a)$ values to evaluate the chromosomes. *Q-Decomposition* [26] combines several RL agents, each with its own rewards, state-action values and RL algorithm. An arbitrator combines their recommendations, maximising the sum of the rewards for each action. It is designed for distributed tasks that are not necessarily POMPDs. Global convergence is guaranteed if the RL algorithm is SARSA but not if it is Q-Learning. In [9] a GA and RL are combined to solve a robot navigation problem. The greedy policy is applied for some time (until the robot encounters difficulty); next the GA population is evaluated, and the fittest chromosome used to update the state-action values by performing several RL iterations; next a new population is generated in a standard way, except that the state-action values are used probabilistically to alter chromosomes; then the process repeats. Several other techniques are used, some specific to robotics applications, but here we consider only the RL-EC hybrid aspects.

2 A cultural approach to POMDPs

A powerful form of EC is the *cultural algorithm* (CA) [21], in which agents share knowledge in a *belief space* to form a consensus. (The *belief space* of a CA is distinct from the *belief space* of a POMDP, which we do not refer to in this paper.) These hybrids of EC and Machine Learning have been shown to converge more quickly than EC alone on several applications. CAs were developed as a complement to the purely genetic bias of EC. They 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. 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. See [22] for a survey of CA applications, techniques and belief spaces. They have been applied to constrained optimisation [5], multiobjective optimisation [2], scheduling [24] and robot soccer [23], but to the best of our knowledge they have not been applied to POMDPs, nor have they utilised RL.

2.1 Cultural reinforcement learning

We propose a new cultural hybrid of reinforcement learning and evolutionary computation for solving POMDPs called *CUltural Reinforcement Learning* (CURL). The CURL approach is straightforward and can be applied to different RL and EC algorithms. A single set of RL state-action values $Q(s,a)$ is initialised as in the RL algorithm, and the population is initialised as in the EC algorithm. The EC algorithm is then executed as usual, except that each new chromosome is altered by, and used to alter, the $Q(s,a)$, which constitute the CA belief space. 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, while $p_l = 1$ always updates a gene to the corresponding $Q(s,a)$ value, and CURL reduces to SARSA(λ) without exploration. We then treat the modified chromosome as usual by the EC algorithm: typically, 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 [22], *all* chromosomes are allowed to adjust the belief space. There is no ϵ parameter in CURL because exploratory moves are provided by EC.

We may use a steady-state or generational GA, or other form of EC algorithm, and we may use one of the Q-Learning or Q(λ) algorithms to update the $Q(s,a)$, but in this paper we use the GENITOR-based NGA and SARSA(λ). The resulting algorithm is outlined in Figure 2, in which `SARSA(`λ`,`α`,O)` denotes a SARSA(λ) episode with a given value of the α parameter, following the policy specified by chromosome `O` while updating the $Q(s,a)$ as usual. As in NGA the population in randomly initialised and fitness is evaluated using S samples. Note that 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.

2.2 Convergence

For POMDPS, unlike MDPs, suboptimal policies can form local optima in policy space [20]. This motivates the use of global search techniques such as EC, which are less likely to become trapped in local optima, and a hybrid such as CURL uses EC to directly explore policy space. CURL also uses bootstrapping to perform small changes to the policy by hill-climbing on the $Q(s,a)$ values. Hill-climbing has often been combined with GAs to form *memetic algorithms* with faster convergence than a pure GA, and this was a motivation for CURL's design. However, if bootstrapping is used then optimal policies are not necessarily stable: that is, an optimal policy might not attract the algorithm [20]. Thus a hybrid might not be able to find an optimal policy even if it

```
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
)
```

Fig. 2. CURL instantiation

escapes all local optima. The possible instability of optimal policies does not necessarily render such hybrids useless, because there might be optimal or near-optimal policies that *are* stable, but convergence is a very desirable property.

Fortunately, it is easy to show that 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 (for example [9]). The GA used in the CURL instantiation is convergent (to within some accuracy depending on the number of samples used), because every gene in a new chromosome can potentially be mutated to an arbitrary allele. Therefore the CURL instantiation is convergent.

2.3 Note

Ideally, we should now evaluate CURL on standard POMDPs from the literature, but we shall postpone this for future work. The work in this paper is motivated by the need to solve large, complex inventory control problems that do not succumb to more traditional methods. In fact we know of no method in the inventory control literature that can optimally solve our problem in a reasonable time (at least, the constrained form of the problem: see below). We shall therefore test CURL on POMDPs from stochastic inventory control. We believe that the problem we tackle has not previously been considered as a POMDP, but we shall show that it is one.

3 POMDPs from stochastic inventory control

The problem is as follows. We have 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; we assume that these distributions are normal. Demands occur instantaneously at the beginning of each time period and are *non-stationary* (can vary from period to period), and demands in different periods are independent. A fixed delivery cost a is

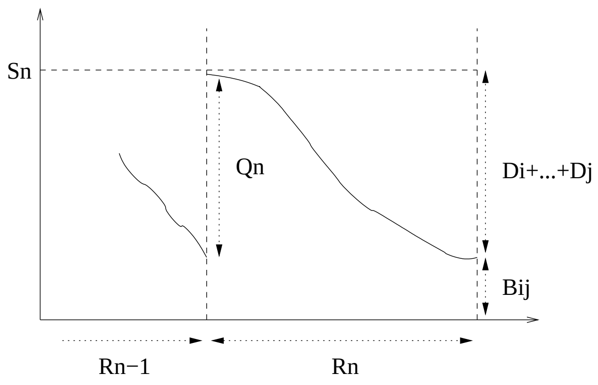

Fig. 3. The (R, S) policy

incurred for each order (even for an order quantity of zero), a linear holding cost h is incurred for each product unit carried in stock from one period to the next, and 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). The aim is to find a replenishment plan that minimizes the expected total cost over the planning horizon. 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.

3.1 Replenishment cycle policy

One possibility is the *replenishment cycle policy* (R, S). Under the non-stationary demand assumption this policy takes 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. In this policy a strategy is adopted under which the actual 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. To provide a solution we must populate both the sets R^n and S^n for $n = \{1, \ldots, N\}$. The (R, S) policy yields plans of higher cost than the optimum, but it reduces planning instability [7] and is particularly appealing when items are ordered from the same supplier or require resource sharing [27]. Figure 3 illustrates the (R, S) policy. R^n denotes the set of periods covered by the nth replenishment cycle; S^n is the order-up-to-level for this cycle; Q_n is the expected order quantity; $D_i + \ldots + D_j$ is the expected demand; B_{ij} is the buffer stock required to guarantee service level α.

Though both RL and EC have been applied to a variety of inventory control problems, some of them POMDPs [31], neither seems to have been applied to this important problem. There are more efficient algorithms which are guaranteed to yield optimal policies (under reasonable simplifying assumptions) so RL and EC would not be applied to precisely this problem in practice. However, if we complicate the problem in simple but realistic ways, for example by adding order capacity constraints or dropping the assumption of independent demands, then these efficient algorithms become unusable. In contrast, RL and EC algorithms can be used almost without modification. Thus the problem is useful as a representative of a family of more complex problems.

Note that the inventory control term *policy* refers to the *form* of plan that we search for (such as the (R, S) policy), whereas a POMDP policy is a *concrete* plan (such as the (R, S) policy with given (R^n, S^n) values). We use the term in both senses but the meaning should be clear from the context.

3.2 POMDP model

The replenishment cycle policy can be modelled as a POMDP as follows. Define a *state* to be the period n, an *action* to be either the choice of an order-up-to level or the lack of an order (denoted here by a special action N), and a *reward* r_n to be minus the total cost incurred in period n. The 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), and its solution is a policy that is *deterministic* and *memoryless* (actions are taken solely on the basis of the agent's current observations). This problem is non-Markovian but has an underlying MDP. Suppose we include the current stock level (suitably discretised or approximated) in the state. We then have the Markov property: the current stock level and period is 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.

The problem is slightly unusual as a POMDP for two reasons. Firstly, 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. Secondly, 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. Neither feature invalidates the POMDP view of the problem, and we believe that instances of the problem make ideal benchmarks for RL and EC methods: they are easy to describe and implement, hard to solve optimally, have practical importance, and it turns out that neither type of algorithm dominates the other.

There exist techniques for improving the performance of RL algorithms on POMDPs, in particular the use of forms of memory such as a belief state or a recurrent neural network. But such methods are inapplicable to our problem because the policy would not then be memoryless, and would therefore not yield a replenishment cycle policy. The same argument applies to stochastic policies, which can be arbitrarily more efficient than deterministic policies [28]: for our inventory problem we require a deterministic policy. Thus some powerful RL techniques are inapplicable to our problem.

4 Experiments

We compare SARSA(λ), the NGA and CURL on five benchmark problems. The instances are shown in Table 1 together with their optimal policies. Each policy is specified by its planning horizon length R and its order-up-to-level S, and the expected cost of the policy per period is also shown, which can be multiplied by the number of periods to obtain the expected total cost of the policy. For example instance (3) has the optimal policy $[159, N, N, 159, N, N, 159, \ldots]$. However, the policy is only optimal if the total number of periods is a multiple of R, and we choose 120 periods as a common multiple of $R \in \{1, 2, 3, 4, 5\}$. This number is also chosen for hardness: none of the three algorithms find optimal policies within 10^8 simulations (a Mixed Integer Programming approach also failed given several hours). We varied only the a parameter, which was sufficient to obtain different R values (and different results: see below). We allow 29 different order-up-to levels at each period, linearly spaced in the range 0–280 at intervals of 10, plus the N no-order option, so from each state we must choose between 30 possible actions. This range of order-up-to levels includes the levels in the optimum policies for all five instances. Of course if none of the levels coincides with some order-up-to-level in an optimal policy then this prevents us from finding the exact optimum policy. But even choosing levels carefully so that the exact values are reachable does not lead to optimal policies using the three algorithms.

Table 1. Instances and their optimum policies

				demand	demand			cost/	cost/120
#	h	s	a	mean	std dev	R	S	period	periods
(1)	1	10	50	50	10	1	63	68	8160
(2)	1	10	100	50	10	2	112	94	11280
(3)	1	10	200	50	10	3	159	138	16560
(4)	1	10	400	50	10	4	200	196	23520
(5)	1	10	800	50	10	5	253	279	33480

As mentioned above, this problem can be solved in polynomial time because of its special form, which is how we know the optimum policies. We therefore also generate five additional instances (1c,2c,3c,4c,5c) by adding an order capacity constraint to instances (1,2,3,4,5) respectively, simply by choosing an upper bound below the level necessary for the optimum policy. For each instance the 30 levels are linearly spaced between 0 and $\lfloor 0.8S \rfloor$ (respectively 54, 89, 127, 156 and 223). This problem is NP-hard and we know of no method that can solve it to optimality in a reasonable time. We therefore do not know the optimum policies for these instances, only that their costs are at least as high as those without the order constraints.

We tailored NGA and CURL for our application by modifying the mutation operator: because of the special nature of the N action we mutate a gene to N with 50% probability, otherwise to a random order-up-to level. This biased mutation improves NGA and CURL performance. We also tailored SARSA and CURL for our application. Firstly, we initialise all state-action values to the optimistic value of 0, because the use of optimistic initial values encourages early exploration [30]. Secondly, we experimented with different methods for varying ϵ, which may decay with time using different methods. [3, 16] decrease ϵ linearly from 0.2 to 0.0 until some point in time, then fix it at 0.0 for the remainder. [30] recommend varying ϵ inversely with time or the number of episodes. We found significantly better results using the latter method, under the following scheme: $\epsilon = 1/(1 + \epsilon' e)$ where e is the number of episodes so far and ϵ' is a fixed coefficient chosen by the user. For the final 1% of the run we set $\epsilon = 0$ so that the final policy cost reflects that of the greedy policy (after setting ϵ to 0 we found little change in the policy, so we did not devote more time to this purely greedy policy).

Each of the three algorithms has several parameters to be tuned by the user. To simulate a realistic scenario in which we must tune an algorithm once then use it many times, we tuned all three to a single instance: the middle instance (3) without an order capacity constraint. For SARSA(λ) we tuned $\epsilon', \alpha, \lambda$ by the common method of hill-climbing in parameter space to optimise the final cost of the evolved policy, restricted to λ values $\{0.0, 0.1, \ldots, 0.9, 1.0\}$ and ϵ', α values $\{0.1, 0.03, 0.01, 0.003, \ldots\}$. This process led to $\alpha = 0.003$, $\epsilon' = 0.001$ and $\lambda = 0.7$. We chose NGA settings $p_c = p_m = 0.5$ and $P = S = 30$ for each instance: performance was robust with respect to these parameters, as reported by many GA researchers. To tune CURL we fixed the GA parameters as above, set $\lambda = 0$, and applied hill-climbing to the remaining CURL parameters, restricted to $p_l \in \{1.0, 0.3, 0.1, 0.03, \ldots\}$, to obtain $\alpha = 0.1$, $p_l = 0.3$. Using $\lambda > 1$ did not make a significant difference to performance (though it necessitated different values for α and p_l): it might be necessary for deterministic problems in which we do not evaluate chromosome fitness over several simulations, but here we have $S = 30$ simulations per chromosome in which to perform bootstrapping so we use the more efficient SARSA(0).

Figures 4 and 5 plots the performances of the algorithms on the instances. The SARSA(λ) cost is an exponentially-smoothed on-policy cost (the policy actually followed by the algorithm during learning). The NGA and CURL costs are those of the fittest chromosome. All graph points are means over 20 runs. We use the number of SARSA(λ) episodes or GA simulations as a proxy for time, and allow each algorithm 10^6 episodes or simulations. This slightly biases the results in favour of SARSA(λ): one of its episodes takes approximately three times longer than a simulation because of its eligibility trace. But there may be faster implementations of SARSA(λ) than ours so we use this implementation-independent metric.

The graphs show that neither SARSA(λ) nor NGA dominates the other over all instances, though SARSA(λ) is generally better (this might be caused by our choice of instances). However, CURL is uniformly better than NGA, and therefore sometimes better than SARSA(λ) also. Previous research into EC and RL on POMDPS has shown that neither dominates over all problems, but that EC is better on highly non-Markovian problems, so we assume that the problems in which NGA beats SARSA(λ) are highly non-Markovian. This implies that CURL is a very promising approach to such POMDPs, though further experiments are needed to confirm this pattern.

It might be suspected that the biased mutation technique unfairly aids NGA and CURL: but adding this technique to SARSA(λ) worsens its performance. Unlike RL algorithms, EC algorithms can benefit from application-specific mutation and recombination operators, and these can also be used in CURL. The current CURL implementation uses a simple table-based representation of the POMDP, which is often the worst choice [19], so we believe that there is a great deal of room for improvement.

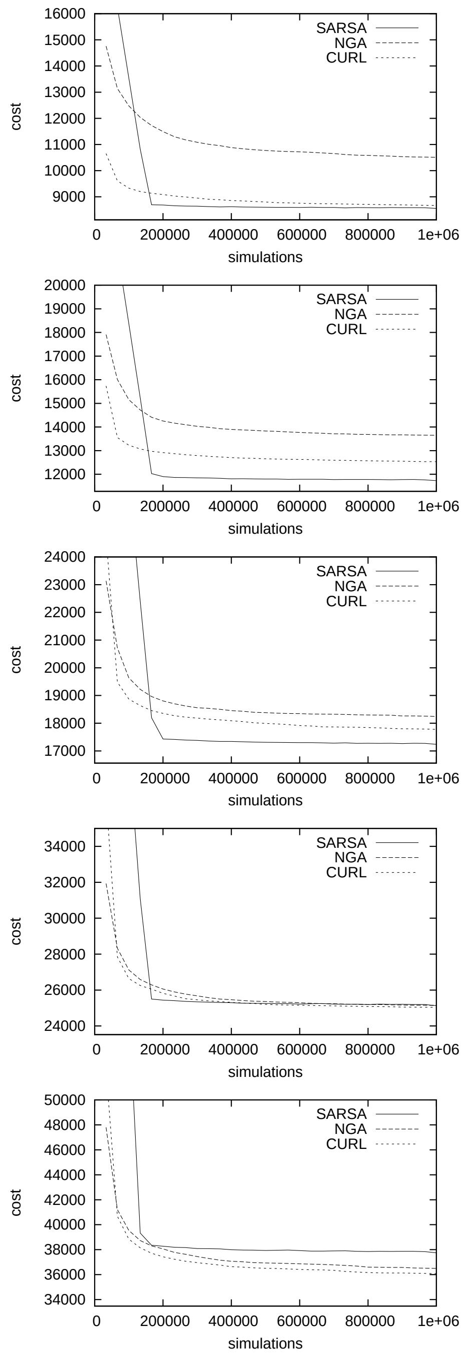

Fig. 4. instances (1,2,3,4,5)

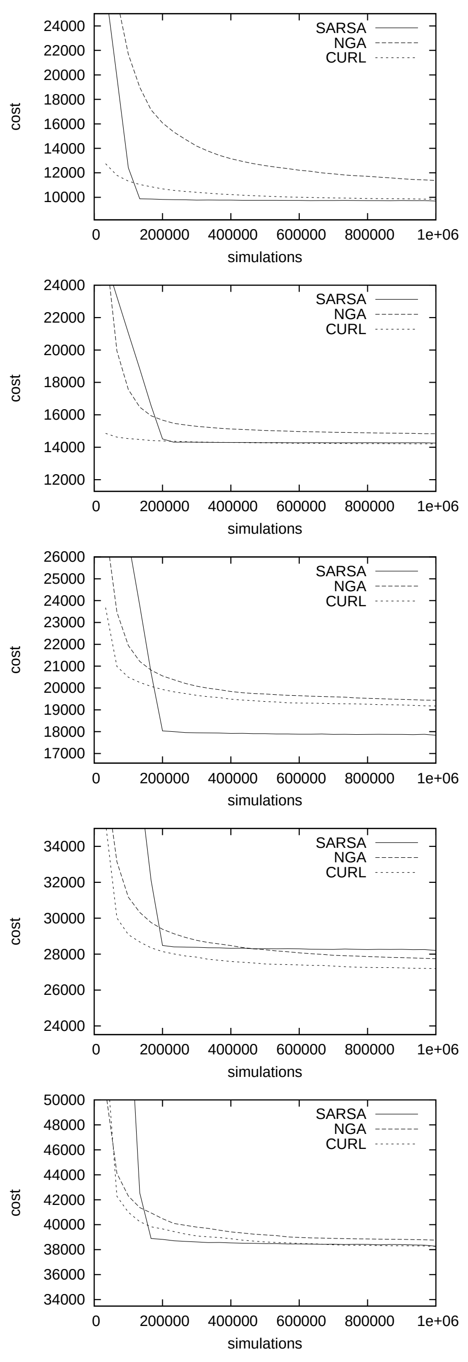

Fig. 5. instances (1c,2c,3c,4c,5c)

5 Conclusion

Reinforcement Learning (RL) and Evolutionary Computation (EC) are competing approaches to solving POMDPs. We presented a new Cultural Algorithm (CA) schema called CURL that hybridises RL and EC, and inherits EC convergence properties. We also described POMDPs from stochastic inventory theory on which neither RL nor EC dominates the other. In experiments a CURL instantiation outperforms the EC algorithm, and on highly non-Markovian instances it also outperforms the RL algorithm. We believe that CURL is a promising approach to solving POMDPs, combining EC and RL algorithms with little modification.

This work is part of a series of studies in solving inventory problems using systematic and randomised methods. In future work we intend to develop CURL for more complex inventory problems, and for more standard POMDPs from the Artificial Intelligence literature.

Acknowledgement This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 05/IN/I886, and under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR). S. A. Tarim and B. Hnich are supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. SOBAG-108K027.

References

1. D. V. Arnold, H.-G. Beyer. Local Performance of the (1+1)-ES in a Noisy Environment. *IEEE Trans. Evolutionary Computation* 6(1):30–41, 2002.
2. R. L. Becerra, C. A. C. Coello. A Cultural Algorithm with Differential Evolution to Solve Constrained Optimization Problems. *9th Ibero-American Conference on Artificial Intelligence, Lecture Notes in Computer Science*, vol. 3315, Springer, 2004, pp. 881–890.
3. G. de Croon, M. F. van Dartel, E. O. Postma. Evolutionary Learning Outperforms Reinforcement Learning on Non-Markovian Tasks. *Workshop on Memory and Learning Mechanisms in Autonomous Robots, 8th European Conference on Artificial Life*, Canterbury, Kent, UK, 2005.
4. J. M. Fitzpatrick, J. J. Grefenstette. Genetic Algorithms in Noisy Environments. *Machine Learning* 3:101–120, 1988.
5. F. Gao and G. Cui and H. Liu. Integration of Genetic Algorithm and Cultural Algorithms for Constrained Optimization. *13th International Conference on Neural Information Processing, Lecture Notes in Computer Science* vol. 4234, Springer, 2006, pp. 817–825.
6. G. Gopalakrishnan, B. S. Minsker, D. Goldberg. Optimal Sampling in a Noisy Genetic Algorithm for Risk-Based Remediation Design. *World Water and Environmental Resources Congress*, ASCE, 2001.
7. G. Heisig. Comparison of (s,S) and (s,nQ) Inventory Control Rules with Respect to Planning Stability. *International Journal of Production Economics* 73:59–82, 2001.
8. J. H. Holland. Adaptation. In *Progress in Theoretical Biology IV*, Academic Press, 1976, pp. 263–293.
9. R. Iglesias, M. Rodriguez, M. Sánchez, E. Pereira, C. V. Regueiro. Improving Reinforcement Learning Through a Better Exploration Strategy and an Adjustable Representation of the Environment. *3rd European Conference on Mobile Robots*, 2007.

10. T. Jaakkola, S. P. Singh, M. I. Jordan. Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems. *Advances in Neural Information Processing Systems 6*, MIT Press, 1994.
11. T. Kovacs, S. I. Reynolds. A Proposal for Population-Based Reinforcement Learning. Technical report CSTR-03-001, Department of Computer Science, University of Bristol, 2003.
12. M. Littman. Memoryless Policies: Theoretical Limitations and Practical Results. *3rd Conference on Simulation of Adaptive Behavior*, 1994.
13. M. L. Littman, A. R. Cassandra, L. P. Kaelbling. Learning Policies for Partially Observable Environments: Scaling Up. *International Conference on Machine Learning*, 1995.
14. M. Littman, T. Dean, L. Kaelbling. On the Complexity of Solving Markov Decision Problems. *11th Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 394–402.
15. H. Liu, B. Hong, D. Shi, G. S. Ng. On Partially Observable Markov Decision Processes Using Genetic Algorithm Based Q-Learning. *Advances in Neural Networks*, Watam Press, 2007, pp. 248-252.
16. J. Loch, S. P. Singh. Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes. *15th International Conference on Machine Learning*, 1998, pp. 3230–331.
17. B. L. Miller. Noise, Sampling, and Efficient Genetic Algorithms. PhD thesis, University of Illinois, Urbana-Champaign, 1997.
18. B. L. Miller, D. E. Goldberg. Optimal Sampling for Genetic Algorithms. *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 6, ASME Press, 1996, pp. 291–298.
19. D. E. Moriarty, A. C. Schultz, J. J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research* 11:241–276, 1999.
20. M. D. Penrith, M. J. McGarity. An Analysis of Direct Reinforcement Learning in non-Markovian Domains. *15th International Conference on Machine Learning*, Morgan Kaufmann, 1998, pp. 421–429.
21. R. G. Reynolds. An Introduction to Cultural Algorithms. *3rd Annual Conference on Evolutionary Programming*, World Scientific Publishing, 1994, pp. 131–139.
22. R. G. Reynolds. Cultural Algorithms: Theory and Applications. New Ideas in Optimization, McGraw Hill, 1999, pp. 367–377.
23. R. G. Reynolds, C. Chung. A Cultural Algorithm Framework to Evolve Multiagent Cooperation With Evolutionary Programming. *6th International Conference on Evolutionary Programming, Lecture Notes in Computer Science* vol. 1213/1997, Springer, 2006, pp. 323–333.
24. D. C. Rivera, R. L. Becerra, C. A. C. Coello. Cultural Algorithms, an Alternative Heuristic to Solve the Job Shop Scheduling Problem. *Engineering Optimization* 39(1):69–85, 2007.
25. G. A. Rummery, M. Niranjan. On-line Q-learning Using Connectionist Systems. Technical report CUED/F-INFENG/TR 166, Cambridge University, 1994.
26. S. J. Russell, A. Zimdars. Q-Decomposition for Reinforcement Learning Agents. *20th International Conference on Machine Learning*, AAAI Press, 2003, pp. 656–663.
27. E. A. Silver, D. F. Pyke, R. Peterson. Inventory Management and Production Planning and Scheduling. John-Wiley and Sons, New York, 1998.
28. S. Singh, T. Jaakkola, M. Jordan. Learning Without State-Estimation in Partially Observable Markovian Decision Processes. *11th International Conference on Machine Learning*, Morgan Kaufmann, 1994, pp. 284–292.
29. P. D. Stroud. Kalman-Extended Genetic Algorithm for Search in Nonstationary Environments with Noisy Fitness Functions. *IEEE Transactions on Evolutionary Computation* 5(1):66–77, 2001.
30. R. S. Sutton, A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
31. J. T. Treharne, C. R. Sox. Adaptive Inventory Control for Nonstationary Demand and Partial Information. *Management Science* 48(5):607-624, 2002.

32. C. J. C. H. Watkins. Learning From Delayed Rewards. PhD thesis, Cambridge University, 1989.
33. D. Whitley, J. Kauth. GENITOR: A Different Genetic Algorithm. *Rocky Mountain Conference on Artificial Intelligence*, Denver, CO, USA, 1988, pp. 118–130.