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

M.J. Blesa et al. (Eds.): HM 2008, LNCS 5296, pp. 16–28, 2008.

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 with an *eligibility trace* [10,16] that effectively hybridises them with a Monte Carlo