A Proposal for Population-Based Reinforcement Learning

Tim Kovacs and Stuart I. Reynolds

Department of Computer Science
The University of Bristol
Bristol BS8 1UB England
{kovacs,reynolds}@cs.bris.ac.uk
http://www.cs.bris.ac.uk/

January 14, 2003

Abstract

We propose novel ways of solving Reinforcement Learning tasks (that is, stochastic optimal control tasks) by hybridising Evolutionary Algorithms with methods based on value functions. We call our approach Population-Based Reinforcement Learning. The key idea, from Evolutionary Computation, is that parallel interacting search processes (in this case Reinforcement Learning or Dynamic Programming algorithms) can aid each other, and produce improved results in less time than the same number of search processes running independently. This is a new and general direction in Reinforcement Learning research, and is complementary to other directions as it can be combined with them. We briefly compare our approach to related ones.

1 Introduction

We propose novel ways of solving Reinforcement Learning (RL) tasks (that is, stochastic optimal control tasks) by hybridising Evolutionary Algorithms with methods based on value functions. The latter include Reinforcement Learning algorithms and Dynamic Programming (DP). Unlike evolutionary methods, RL and DP methods are very data efficient, but make stronger assumptions about the task that makes it difficult to scale them up to solve many real-world problems.

We call our approach *Population-Based Reinforcement Learning*. The key idea, from Evolutionary Computation, is that parallel interacting search processes (in this case RL or DP algorithms) can aid each other, and produce improved results in less time than the same number of search processes running independently. Furthermore, the bulk of computation can also be performed independently, making it easy to distribute across cheap clusters of workstations (such as departmental workstations that often sit idle, or the forthcoming GRID). In this way, real-time learning costs can be reduced even if the total learning costs of the new approach prove to be more expensive. This is a new and general direction and promises to be fruitful with, eventually, a broad range of applications. It is complementary to other directions in RL and DP research since it can be combined with existing methods from those fields.

This document outlines the approach we intend to take and contrasts it with existing work in related areas. Section 2 provides some background on the relevant areas: RL and Evolutionary Algorithms. Section 3 introduces the main aspects of our population-based approach, along with some interesting, but non-essential, opportunities it affords. Finally, section 4 briefly compares our approach with related ones.

2 Background

This section very briefly introduces the tasks we wish to address, followed by the two types of solution methods which our approach hybridises.

2.1 Reinforcement Learning Tasks

Reinforcement Learning is a computational approach to learning how to obtain rewards and avoid punishments through trial and error, directly through interaction with the task. The learner has some *policy* which specifies an action for each state the system may be in. The goal of the learner is to find an optimal policy, which maximises the reward it receives in the long term. Successful applications of RL to real-world problems include elevator dispatching [2], space shuttle payload scheduling [12], and a master-level backgammon player [11]. In each case RL provided solutions that outperformed best known prior solutions.

Concretely, consider the case of an offshore wave-powered electricity generator. Segments of the system are articulated; as they ride the waves they drive shafts through magnetic fields and so generate power.[1] The aim of the system is to maximise the power output over a period of time. Thus, the power output supplies a reinforcement signal which the system aims to maximise. The system has various sensors that indicate its state (e.g. the general size of waves, relative position of segments, position upon the current wave, and so on). In each state the Reinforcement Learning algorithm can take various actions, such as controlling its buoyancy or the stiffness of joints, each of which affects how much power is generated. Also, the effects of some actions may not be immediate; actions taken now may result in rewards later. For instance, the stiffness of the joints in the trough of a wave may affect how much power is generated at its peak. Reinforcement Learning algorithms solve this *delayed credit assignment problem*.

2.2 Reinforcement Learning Algorithms and Dynamic Programming

Modelling a system such as the one above in simulation can be difficult, costly and error-prone (e.g. real-world factors such as mechanical wear are difficult to factor in). RL algorithms require no such model. They simply observe changes in reward following their actions and use this to learn better actions in the future. They do this by learning a *value-function*; an evaluation of the long-term expected reward of taking actions in states.

In cases where a model of the environment is available, or can be learned, then Dynamic Programming (DP) methods also apply. Their approach to the solution is similar to RL methods. From here on we refer to both fields generically as "RL".

In cases where the RL task can be formalised as a small discrete (but unknown) Markov Decision Process (MDP), due to their rigorously proven statistical foundations, several RL algorithms provide guarantees of convergence upon optimal policies. However, this formalism seldom applies for real-world tasks where environments are either often partially observable (i.e. non-Markov), very large or have continuous state. As a result, despite much initial success, RL often has difficulty scaling to interesting tasks. In addition, RL suffers from the "curse of dimensionality"; computational requirements tend to increase exponentially with the number of input variables.

2.3 Evolutionary Computation

In contrast to RL, Evolutionary Computation (EC) methods make much weaker assumptions about the task that allows them to succeed in places where RL fails. Also, the fundamental way in which they search for the problem's solution is different; EC methods directly search in the space of policies, while RL methods search in the space of value functions (from which policies are indirectly derived). EC methods work as follows:

1. create a population P of n random policies
2. repeat
3. evaluate each policy in P
4. replace P with a new population created with evolutionary methods

Step 4 consists of preferentially selecting higher-valued ("fitter") policies as parents, and applying evolutionary operators (e.g. crossover and mutation) to transform them stochastically into offspring.

We believe that directly searching for policies in this way can be more effective than RL in some cases, e.g. when the Markov assumption does not hold. See also, e.g. [1], on the advantages of policy search. On the other hand, a disadvantage of evolutionary search is that it may be prone to finding sub-optimal

[1] This example was inspired by the Pelamis wave energy converter currently in development by Ocean Power Delivery Ltd. of Edinburgh.

solutions. Additionally, for EC methods to improve their policies, they must evaluate their fitness. This can be slow in cases where the environment is stochastic and there is a long delay between actions and the rewards they generate. In contrast, RL methods evaluate policies incrementally, more efficiently and at a much finer level of detail.

The power of our approach will be derived from combining the benefits of both methods, finding a middle ground that is more powerful than either extreme.

3 Three Approaches to Scaling RL

We will attempt to scale RL up to larger tasks in three different ways as described in the following sections. Section 3.1 details the main innovation. Sections 3.2 and 3.3 detail some interesting opportunities and challenges that follow.

3.1 Population-Based Reinforcement Learning

To date, EC and standard (i.e. value function-based) RL have been considered quite separate. However, the methods are complementary and we can combine them in the same algorithm. For example:

1. create a population P of n random policies
2. repeat
3. evaluate each policy in P
4. improve each policy in P based on value functions
5. replace (part of) P with a new population created with evolutionary methods

The details of this abstract algorithm are left unspecified; any number of instantiations are possible that combine existing techniques derived from EC and RL. See [6] for more on the relationship between RL and EC.

Why is it useful to combine these two approaches? Our approach has several novel advantages. *i*) From EC we will take methods that allow direct search through policy space. Although value function methods (RL methods) are efficient (and in some cases provably convergent), evolutionary operators allow larger jumps in policy space. *ii*) From RL we will take rapid incremental learning methods that produce detailed statistics about the quality of the learner's policy in particular states that are not available to standard evolutionary methods. This information will allow us to provide more directed alternatives to the mutation and crossover operators standardly used by EC methods. *iii*) Because evolutionary search evaluates and reproduces entire policies, it does not suffer in the ways that value function methods do when the Markov property is violated [10]. In particular, existing attempts to scale up RL using function approximators (e.g. with neural networks) have been shown to be unstable if attempts are made to improve the policy during learning. Our approach allows the policy to be fixed from the perspective of the RL element, yet the evolutionary element allows continued improvements to it. *iv*) Our method allows the algorithms to make efficient use of experience through learning by observation (see section 3.2). *v*) Instances of our method are easily parallelised (see section 3.3). These five advantages should allow us to solve RL tasks which cannot currently be solved.

Hybridising evolutionary search and local search [9] has often proved effective. If we consider the policy improvement phase of the algorithm above as a form of local search, we can consider this algorithm as another instance of the hybrid evolutionary/local search approach.

Why has this hybrid approach not already been investigated? First, Evolutionary Computation and Reinforcement Learning are two different fields composed of different researchers. Second, they take very different approaches to learning policies. Although the similarity between the two at a very abstract level is apparent from the algorithms presented above, they are not usually presented in this way. Third, although EAs can be applied to RL there has been little work in this area (apart from work on Learning Classifier Systems), which helps explain why this novel approach to hybridising them has not occurred until now.

Challenges 1. We will need to characterise the space of policies as a fitness landscape, in order to bridge the two approaches conceptually. 2. We anticipate that the presence of multiple optimal policies will produce multiple global optima in the fitness landscape, introducing difficulties for crossover operators, the need for restricted mating and, perhaps, other means of encouraging diversity.[2] 3. Sophisticated forms of crossover which extract above-average subpolicies based on the value function and state transition information are possible. However, it is as yet unclear which may be best. 4. It would be desirable to derive policy fitness from the value function (mainly because Temporal Difference methods perform evaluation rapidly), rather than using the obvious Monte Carlo approach. However, this will require estimating the error in the value function, to avoid reproducing optimistic value functions preferentially.

3.2 Learning By Observation

In tasks such as the power generation scenario, there are different ways in which our hybrid population of learners may interact together and with the environment. There may be many independent, but similar controllable processes each with a Reinforcement Learning controller (e.g. in our scenario, consider that we might have an array of power generators). However, given that experience of the environment is being collected simultaneously and in parallel, and that such experience may be costly to obtain, it makes sense for each agent to attempt to exploit the experience generated by the others by observing their actions and their consequences. We call this model *learning by observation*. Some existing work tackles this problem, but generally in the context of multi-agent RL, and differs from our approach in ways discussed in section 4. One of our goals will be to find methods of learning by observation which suit the methods of section 3.1.

3.3 Parallel Implementation

Writing algorithms to exploit parallel processing machines is often costly and difficult. Evolutionary Algorithms, however, are easily parallelised since (typically) each element in the population can have its fitness evaluated independently. In the context of RL, this means each individual's policy can be evaluated independently on its own processor. Furthermore, clusters of workstations are common and inexpensive compared to massively parallel shared-memory machines. We expect to be able to employ clusters to dramatically reduce real-time running costs. However, there is a communications latency between such machines which makes learning by observation more difficult: by the time one machine can broadcast its experience to another, a second can often generate much more experience independently.[3] However, even when communications latencies preclude learning by observation, workstation clusters are still compatible with our hybrid search approach. In this case, machines communicate at the policy level, rather than at the experience level. Communicating policy information (i.e. using crossover) occurs infrequently, so network latencies are not a significant problem.[4]

3.4 Summary

Existing RL methods have trouble scaling up as they make assumptions that do not apply for real-world applications, and suffer from the curse of dimensionality. We propose to extend the range of tasks that RL can handle using an approach which is novel in two ways: 1) we will combine evolutionary search with existing RL methods yielding more efficient search, and 2) unlike existing RL methods, the resulting algorithms will be well-suited to parallel implementation on inexpensive clusters of workstations. Where possible, our approach will take advantage of learning by observation among the population of agents.

We wish to emphasise that we expect the performance of the new approach will be as good or better than standard RL per unit of real-time. That is, the worst case is where only the best performing agent is ultimately used and the others are disregarded. Finally, we note that our approach is universal in the sense that it can be used with any RL algorithm, including other approaches to extending scalability (such as function approximation).

[2] In the present context, a fitness landscape is a mapping from each policy to a measure of its value, i.e. a value function over complete policies. It is this function which evolutionary search attempts to optimise.

[3] There are situations where generating experience is slow and communications latency is not an obstacle to sharing it, for example using real robots, whose physical movements involve greater latencies than the communications latencies.

[4] This parallels the situation in nature where children cannot inherit the experiences of their parents but only their genes.

4 Novelty and Related Work

Our approach differs from existing, loosely related work using evolutionary methods, Learning Classifier Systems, ensemble methods, hierarchical RL, and Ant-Q algorithms. None of these methods apply a population of non-interacting agents with homogeneous reward functions to the same task, as our approach does. All these areas either address different problems to ours, or address our problem in a fundamentally different way.

Evolutionary Methods and Learning Classifier Systems It must be emphasised that we are neither simply proposing a form of evolution-only search in policy space, nor are we proposing any form of Learning Classifier System. We are, rather, proposing a new class of algorithms, which combine features of both value function methods and Evolutionary Algorithms. Although Learning Classifier Systems are hybrids of value function and Evolutionary Algorithms, the proposed work hybridises them in a fundamentally different way. Where Learning Classifier Systems use evolutionary methods for generalisation and value function methods for policy learning, in the hybrid approach proposed here, both are applied to policy learning. Generalisation is an orthogonal issue, which can be addressed using the range of function approximators normally employed in RL (e.g. neural networks or Learning Classifier Systems).

Multi-agent RL The proposed work differs from the considerable work on multi-agent RL in that in multi-agent RL agents may affect each other through the environment and consequently are attempting to optimise Multi-Agent Markov Decision Processes (MAMDPs). For example, they may compete or collaborate, or simply change the state of the environment, all of which complicates matters considerably. Our approach (in its basic formulation) optimises MDPs, not MAMDPs, and so many of the difficulties of multi-agent RL do not apply. We expect to extend convergence proofs to our methods in special cases.

Of note are two works on multi-agent RL which employ forms of learning by observation. Crites's work on elevator scheduling employed a population of agents (one for each elevator), which, in one configuration, updated a common value function [2]. Pendrith's Distributed Q-learning (DQL) resembles Crites's work, except that only nearby agents can sense each other, and the Q-updates are averages of the values generated by the agents taking a given action in a given state [7]. Both, however, apply agents with a common policy to MAMDPs, whereas our approach applies agents with different policies to MDPs.

Ensemble Methods Our approach differs from ensemble methods [8, 3], in which heterogeneous agents share one body and adapt to different aspects of the task. In our basic approach we assume that agents are homogeneous, but we could extend our approach to use agents with different representations (e.g. agents which use different function approximators to represent their value functions) in which case ensemble methods might be applicable. Ensemble methods are typically applied to classification tasks and we are unaware of any application of ensemble methods per se to RL (but see the discussion of Humphrys's work below).

Hierarchical RL Humphrys's work on RL with a collection of agents could be seen as a form of ensemble method for RL [5]. In it, a task is divided into sub-goals, and one agent assigned to each sub-goal. Each agent has its own reward function, and access to its own subset of the state space. It is therefore a way of decomposing RL tasks into simpler tasks, and achieving (hand-coded) generalisation by restricting access to the state space. The main focus of Humphrys's work was an investigation of methods for coordinating the agents involved. Our approach differs in that each agent has the same task and reward function, and the same access to the state space. Our approach could, however, be extended to incorporate ideas from Humphrys's work; we could give agents heterogeneous reward functions and subsets of the state space. Evolutionary methods could be used to adapt both. In this way, the task might be decomposed automatically rather than by hand.

The Ant-Q Algorithm The Ant-Q algorithm [4] resembles our population-based approach in that a population of agents collectively solve an MDP. The approaches differ in that the ants share a single value function and derive different policies from it, whereas our agents each have their own policy and value function. Ant-Q was developed for the travelling salesman problem, and consequently includes a form of memory and a shorter-path heuristic which our approach does not.

References

[1] Charles W. Anderson. Approximating a Policy Can be Easier Than Approximating a Value Function. Technical Report CS-00-101, Colorado State University, 2000.

[2] R. H. Crites. *Large-Scale Dynamic Optimization Using Teams of Reinforcement Learning Agents.* PhD thesis, University of Massachusetts, Amherst, 1996.

[3] Thomas G. Dietterich. Ensemble Methods in Machine Learning. In J. Kittler and F. Roli, editors, Proceedings of the First International Workshop on Multiple Classifier Systems (MCS 2000), pages 1–15. Springer–Verlag, 2000.

[4] L. M. Gambardella and M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In A. Prieditis and S. Russell, editors, *Proc. 12th International Conference on Machine Learning (ICML)*, pages 252–260. Morgan Kaufmann, 1995.

[5] Mark Humphrys. *Action Selection Methods using Reinforcement Learning.* PhD thesis, Cambridge University, 1997.

[6] Tim Kovacs. *A Comparison of Strength and Accuracy-Based Fitness in Learning Classifier Systems.* PhD thesis, University of Birmingham, Birmingham, UK, 2002.

[7] Mark D. Pendrith. Distributed reinforcement learning for a traffic engineering application. In C. Sierra, M. Gini, and J. S. Rosenschein, editors, *Proc. of the 4th International Conference on Autonomous Agent (Agents 2000).* ACM Press, 2000.

[8] Michael P. Perrone and Leon N. Cooper. When Networks Disagree: Ensemble Methods for Hybrid Neural Networks. In R. J. Mammone, editor, Neural Networks for Speech and Image Processing, pages 126–142. Chapmann-Hall, 1993.

[9] Vic J. Rayward-Smith. A Unified Approach to Tabu Search, Simulated Annealing and Genetic Algorithms. In Vic Rayward-Smith, editor, *Applications of Modern Heuristic Methods*, pages 17–38. Alfred Waller Ltd, 1995.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[11] G. J. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38:58–68, 1995.

[12] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1114–1120. Morgan Kaufmann, 1995.