Department of Computer Science,
School of Engineering and Computer Science,
University of Exeter. EX4 4QF. UK.
http://www.ex.ac.uk/secs

UNIVERSITY of EXETER

Controlling Genetic Algorithms with Reinforcement Learning

James E. Pettinger[*] and **Richard M. Everson**[†]

4th March 2003

Abstract

The choice of genetic operators and the frequency with which they should be applied in evolutionary algorithms is an important open question. Here we cast evolutionary algorithms in a reinforcement learning (RL) framework and present a hybrid reinforcement learning controlled evolutionary algorithm. The RL agent uses Q(λ) learning to estimate state-action utility values of choosing particular evolutionary operators and the classes of parent chromosomes to which the operators are applied. The RL agent's choices are based on characteristics of the optimisation represented by the operator state: the time into the optimisation, the diversity of the population and the population mean fitness. The RL agent is trained to maximise the discounted reward it obtains from crossover and mutation operations that improve the fitness of the offspring.

The hybrid RL-GA system is shown to learn faster than a conventional GA on the travelling salesman problem, and we show that it learns to discriminate between effective and ineffective operators.

Keywords: Reinforcement learning, genetic algorithm, travelling salesman problem, combinatorial optimisation, hybrid, adaptive.

1 Introduction

One of the weaknesses of genetic algorithms (GAs) is that designing and running them is somewhat of a black art. There is a myriad of choices to be made in their implementation, such as how to represent the problem as a chromosome, how frequently to crossover and mutate, how to determine which individuals survive, which individuals will be selected for breeding, what operators to use for crossover and mutation, and so on. Studies such as [7, 10] attempt to answer the question of which genetic operators are best and schemes such as those presented in [4, 6, 13, 14] dynamically adjust the probabilities with which the operators are applied. However it is unclear that recommendations obtained for one particular problem can be usefully transfered to others; indeed it seems unlikely that a general solution, applicable to all problems, would be available. Furthermore, it is likely that one operator will be effective near the beginning of an optimisation, but another more

[*]Currently with Neurascript, email: JEP@neurascript.com
[†]Corresponding author, email: R.M.Everson@exeter.ac.uk

useful near to the end. In this study attempt we examine the use of reinforcement learning methods [12] to automatically and adaptively control aspects of a GA, namely which genetic operators.

Reinforcement learning (RL) is often characterised as the 'stick and carrot' approach to machine learning. An agent takes actions, a_t and each action is rewarded or penalised with a reward, r_t. The agent's goal is to maximise its total reward, but unlike supervised learning methods the agent is not instructed as to the 'correct' action; instead it must learn the optimal action in a particular situation or state. Clearly an agent must balance exploiting immediately rewarding actions with exploratory actions that might bring long-term benefit.

Function optimisation itself may be regarded as a reinforcement learning problem. The overall reward is the maximisation of the function, and the optimisation algorithm (*e.g.,* GA, evolutionary strategy, etc) must decide which solution to evaluate next based on the reward (the function evaluation) just received. Indeed, Boyan & Moore [3] have used RL to learn efficient initial points for a gradient based optimiser, and Zhang & Diettrich [18] have used RL for job shop scheduling.

The idea of adapting operators during an optimisation is not new, see [11] for a review. Davis [4] suggested an algorithm which assigns credit to operators that contributed to the formation of fit offspring; in Davis's algorithm credit is assigned at intervals. The ADOPP scheme suggested by Julstrom [6] adapts operator probabilities after every genetic crossover or mutation operation. In several respects the data structures used for recording credit in ADOPP are similar to the eligibility traces and state-action tables used in our reinforcement learning scheme. More recently Tuson & Ross [13] have examined a heuristic which assigns operator probabilities according to their productivities (the average increase in fitness when a child was fitter than its parents), however, they found that the initial operator probabilities were the most important factors in determining performance.

Here we examine the problem of operator selection in a RL framework, and show that RL may be used for controlling the selection of crossover and mutation operators. Our model itself contains two parts: a genetic algorithm and a reinforcement learning agent. The GA is capable of functioning independently but it is non-adaptive. This means it can not learn which actions (genetic operators) are likely to lead to success in a given state. The second part of the system is a reinforcement learning based agent. This agent receives information about the GA and its population of individuals. It knows which action has been taken at the last time-step and the consequences of that action. This state information is used to learn which actions are likely to be useful in a given state and which are not. Using this learnt knowledge, the agent alters the selection probabilities of each action available to the GA, so that actions that likely to lead to better rewards (higher fitness) in the current system state are more likely to be chosen. These action selection probabilities implicitly control crossover and mutation operator selection, crossover and mutation rate and the selection of breeding individuals within the GA.

In Section 2 we give an overview of reinforcement learning (RL) and introduce the particular method we use in our study, Watkin's Q(λ)-learning [15]. In Section 3 we describe our hybrid reinforcement learning controlled genetic algorithm, and in section 4 we performance of the algorithm is evaluated on the travelling salesman problem.

2 Reinforcement Learning

Here we give a very brief introduction to reinforcement learning; readers are referred to Sutton & Barto's expository book [12] for an extensive discussion. Reinforcement learning has been successfully applied to a number problems that, though conceptually simple, cannot be solved using traditional engineering techniques, usually because of the difficulty of accurately modelling a system's response to particular inputs or actions. Consider the problem of riding a bicycle, a conceptually simple task but one for which it would be very difficult to hand-craft a machine algorithm. Despite vast amounts of sensor data that could be collected (speed, frame angle, gear, road image) it would be extremely difficult to design a function that would effectively take this data from the environment and give an appropriate action as an output. A supervised learning approach would also be unsuitable, as it would require examples of the correct action to take in any given situation to train a supervised learning agent; such examples are seldom practically obtainable.

Unlike a supervised learning agent, a reinforcement learning agent is not told the correct action to take in a certain situation when it is learning. Instead, the agent is given a goal and learns by experience how best to achieve it. More precisely, reinforcement learning agent at time step t is characterised by a state, $s_t \in \mathcal{S}$. The state signal the agent receives represents the sum total of knowledge the agent possesses about its environment. The particular action, $a_t \in \mathcal{A}(s_t)$ taken depends upon the *policy*, $\pi(s_t, a_t)$, the probability of taking action a_t in state s_t. The policy constitutes a mapping from the state to the action and learning the optimal policy is the essence of reinforcement learning. While choosing actions $a_t = \arg\max_a \pi(s_t, a)$ for all t will yield a sequence of actions that optimally exploit the current estimates of a 'good' action, exploration is ensured by requiring that all actions $\mathcal{A}(s)$ from state s have a non-zero probability $\pi(s, a)$.

The agent's goal is to maximise the *expected return*:

$$R_t = \sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau+1} \tag{1}$$

where r_{t+1} is the reward obtained in response to the action a_t and γ is the rate at which future rewards are discounted $0 \le \gamma \le 1$. Of course, learning often stops after a finite number of steps, T.

It is generally assumed that the environment of the agent has the Markov property, so that the state and reward at time $t+1$ depend only upon the state and action at time t:

$$P(s_{t+1}, r_{t+1} \,|\, s_t, r_t, a_t, s_{t-1}, r_{t-1}, a_{t-1}, \ldots, s_0, r_0, a_0) = P(s_{t+1}, a_{t+1} \,|\, s_t, r_t, a_t) \tag{2}$$

2.1 Q-learning

The merit of taking an action a from a state s under a policy π may be assessed by the *action-value function*, which is the expected return under π:

$$\begin{aligned} Q(s,a) &= E_\pi \{R_t \,|\, s_t = s, a_t = a\} && (3) \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\, s_t = s, a_t = a \right\} && (4) \end{aligned}$$

A policy may be simply derived from Q via the softmax function:

$$\pi(s,a) = \frac{e^{\beta Q(s,a)}}{\sum_{a'=1}^{N} e^{\beta Q(s,a')}} \tag{5}$$

The action-value utility may be estimated by keeping track of the rewards accrued starting from (s, a). However, these Monte-Carlo methods necessitate waiting until the end of learning to discover the actual return. While this may be practicable in situations where learning can be performed over several short runs, it is impractical for function optimisation, where resources cannot be expended on multiple runs.

Q-learning, introduced by Watkins [15], is a temporal differencing method of updating the action-value function. Having taken action a_t from state s_t to state s_{t+1} and received reward r_{t+1}, the estimate of the utility $Q_{t+1}(s,a)$ at time $t+1$ is updated as follows:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q(s_t,a_t) \right] \tag{6}$$

where α is a learning rate. The the update looks one step ahead and updates $Q(s_t,a_t)$ with the reward accrued from taking action a_t and the discounted maximum (estimated) return from an action at time $t+1$.

Q-learning approximates the expected return by looking one step ahead, in contrast to Monte Carlo methods which look right to the end of learning. $Q(\lambda)$-learning extends the look ahead of temporal difference methods by keeping track of which sequence of state-action pairs contributed each reward.

In $Q(\lambda)$-learning each state-action pair is associated with an eligibility trace, $e(s,a)$. Every time that state-action pair is visited its trace is incremented by 1. At every subsequent time-step, where the pair is not visited again, the trace decays exponentially by a factor λ. An eligibility trace therefore provides a record of the eligibility of a state action pair, (s,a) to share in the rewards gained from an action taken at a subsequent time-step. Thus if an action has been taken from a particular state frequently or recently then it deserves a greater share of the reward gained at the current time-step, because, presumably, it has influenced the current system state more than a state-action pair that has not been visited for a long time or with any great frequency. Under $Q(\lambda)$-learning the state-action utility estimates are updated according to:

$$Q_{t+1}(s,a) = Q_t(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q(s_t,a_t) \right] e(s_t,a_t) \tag{7}$$

$Q(\lambda)$-learning is an off-policy learning method. This means that the policy under which the agent learns is not necessarily the policy adhered to for selecting actions. In the RL-GA system learning takes place under a presumed greedy policy, that is, it is assumed that the action selected is the optimal (exploitative) action according to the current $Q(s,a)$ estimates. If the action a_{t+1} selected from the new state from s_{t+1}, is the optimum action, (*i.e.*, $a_{t+1} = \arg\max_a Q(s_{t+1},a)$), then the eligibility traces for state-action pairs are updated as described above. If, on the other hand, a_{t+1} is exploratory, rather than exploitative, then any reward gained subsequently cannot be used to update back beyond the current time-step. Hence all eligibility traces are reset to 0 ready for the next update. The update of the eligibility traces is summarised as:

$$e_{t+1}(s,a) = \delta_{s,s_{t+1}} \delta_{a,a_{t+1}} + \begin{cases} \gamma\lambda e_t(s,a) & \text{if } a_{t+1} = \arg\max_a Q_t(s_{t+1},a) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $\delta_{m,n}$ is the Kronecker delta. We note again that the eligibility traces play the same role as the trees of operator ancestors constructed for credit assignment in ADOPP [6].

In this application to controlling GAs we use a state-action lookup table to represent $Q_t(s,a)$ and another table to represent $e_t(s,a)$, as the number of states and actions is relatively small. In larger problems, where such a table would be too large to populate efficiently, a neural network or some other function approximator might be used to replace this table.

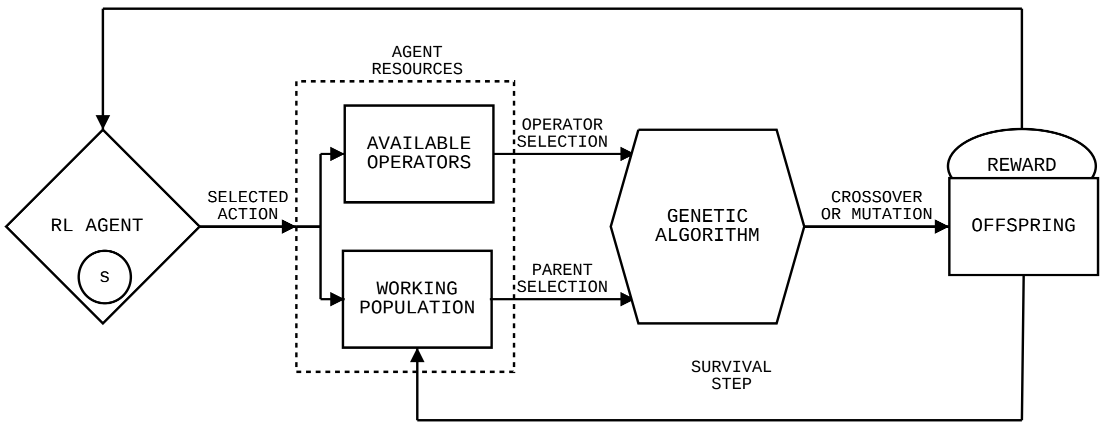

Figure 1. Flowchart representation the RL-GA hybrid system. Beginning in state s operators and individuals are selected according to $Q(s,a)$, following genetic crossover or mutation the reward is calculated and the offspring returned to the working population. The new state is calculated and the cycle repeats.

3 Reinforcement Learning control of a GA

A RL agent may be used to control the crossover and mutation operations of GA in the following manner. Rather than deal with the population as a whole, each action the agent takes is the selection of an individual for mutation or a pair of individuals for mutation. After selection the offspring or mutated individual replace the weakest individuals in the working population. This ensures that the offspring formed using an action for which reward has been received survive to the next step, giving them at least one chance of being selected to breed. A reward based on the fitness of offspring relative to the parents is returned to the agent. A flowchart depicting the algorithm is shown in figure 1.

Denote by $\{\mathbf{x}_m^t\}_{m=1}^M$ the population of M individuals at time t and let the function to be optimised be $f(\cdot)$, so that the fitness of $\mathbf{x}_m$ is $f(\mathbf{x}_m)$.

State. The state, s_t is a vector of the following quantities which characterise the genetic population:

- Generation measured on a logarithmic scale, $\log t$. The logarithmic time was divided into 4 equally spaced (on a log scale) epochs for use in the state-action and eligibility trace tables.
- Average population fitness normalised by the initial population fitness:

$$\bar{f}^t = \frac{\sum_{m=1}^M f(\mathbf{x}_m^t)}{\sum_{m=1}^M f(\mathbf{x}_m^0)} \tag{9}$$

  The average fitness was binned into four intervals ($[0, 0.3)$, $[0.3, 0.4)$, $[0.4, 0.6)$, $[0.6, 1]$) for use in the state-action and eligibility trace tables.

- The entropy of the population fitness, which measures the diversity of fitnesses in the current population. If $p_m = f(\mathbf{x}_m)/\sum_m^M f(\mathbf{x}_m)$, then the Shannon entropy of the distribution $\{p_m\}$ is

$$H = \sum_{m=1}^{M} p_m \log_2 \frac{1}{p_m} \tag{10}$$

The entropy is minimum when all individuals in the population have the same fitness, and it is maximum when fitnesses are uniformly distributed. In a similar manner to the generation and average fitness, the entropy was divided into three equally sized discrete states.

Actions The principal function of the RL agent is to choose a genetic operator from a range of crossover or mutation operators. The operators themselves are described in more detail below. However, we also permit the RL agent to select the *type* of individuals to which the operator is applied. To this end we divide the population into two classes: fit (F) and unfit (U). An individual in the fittest 10% of the working population is deemed fit; all others are unfit. For each crossover operator there are therefore four parental combinations: $\{FF, FU, UF, UU\}$; although FU and UF are equivalent we treated them separately to confirm that they were learned in similar manner. From each original crossover operator, for example, MPX [8], we thus derive four crossover operators: MPX-FF, MPX-FU, MPX-UF and MPX-UU. If an action selects one of these operators, a parent is selected from each of the relevant classes by roulette wheel selection within that class and the crossover operation applied to the two selected parents. In a similar manner, two mutation operators—a F and a U variant— are derived from each ordinary mutation operator. Actions are selected with probability given by the policy $\pi(s, a)$ derived from the state-action table via the softmax function (equation 5) which was found empirically to be more useful than an ϵ-greedy method.

Reward The reward for a crossover or mutation is calculated as the improvement in fitness of the best offspring over the best parent, normalised by the fitness of the best parent:

$$r = \frac{\max\{f(\mathbf{x}) \,|\, \mathbf{x} \in \text{parents}\} - \max\{f(\mathbf{x}) \,|\, \mathbf{x} \in \text{offspring}\}}{\max\{f(\mathbf{x}) \,|\, \mathbf{x} \in \text{parents}\}} \tag{11}$$

For a mutation action there is only a single parent with a single offspring, so the reward is just the fractional improvement in fitness of the mutated individual. This form of reward yields a positive reward if at least one of the offspring is fitter than the fittest of the parents. The agent is thus rewarded for genetic operations that return at least one individual to the population that is fitter than either of the individuals that left it. We note, however, that most genetic operations produce negative reward, so much of the time the agent is learning the least bad operator.

Experiments were conducted with a number of alternative reward functions and this was found to be as effective as any [9]. We remark that rewards which fail to penalise decreases in fitness were found to be ineffective for controlling the GA, although other studies [4, 6] have used this type of reward, and the rewards used by Zhang & Dietterich [18] apply a small fixed penalty for all 'poor' actions.

4 Illustration

We illustrate the performance of the RL-GA system on a simple symmetric travelling salesman problem (TSP). As a model combinatorial optimisation problem the TSP has been extensively studied and is relatively well understood, but remains of considerable importance in applications, for example [1, 2].

We optimise the path length for a tour of $N = 40$ cities with randomly chosen locations. The genetic population comprised of $M = 30$ individuals.

We used three basic crossover operators: the maximum preservation crossover [8] (MPX); the genetic edge recombination [16, 17] (GER), and the partial mapping crossover [5] (PMX). Late in this study a mistake in the implementation of the GER operator was discovered, which means that, in fact, our own GER operator mixes and strongly randomises the chromosomes of the parents to form the offspring.

The four basic mutation operators (MOVE, SCRM, INVR, and SWAP) were of our own devising; each mutates a single parent to yield a single offspring. The MOVE operator repositions a section of parental chromosome to a location further along the chromosome. The length, l, of the moved section is between 2 and 5 cities in length, determined randomly. The start point of the section to be moved is determined randomly (but must be more than l cities from the end of the chromosome). The reinsertion point is a randomly selected point between the end of the section to be moved and the end of the chromosome.

The inversion operator (INVR) takes a section of a single parent chromosome between 3 and 5 cities in length, determined randomly, and inverts it, reversing the order in which the cities appear in the single offspring.

The swap operator (SWAP) selects two randomly located, non-overlapping sections from the single parent. Each section is 3 cities in length. These sections are then transposed to give a single offspring. The order of the cities within the swapped sections remains unchanged.

The scramble operator (SCRM) takes a randomly located section of the single parent, between 3 and 5 cities in length, determined randomly, and replaces it with a random permutation of itself to give a single offspring.

There are thus a total of $3 \times 4 + 4 \times 2 = 20$ actions available to the RL agent and with the discretized state described above, the state-action table has 48×20 cells. Values of $\gamma = 0.5$, $\alpha = 0.1$ and $\beta = 0.05$ were used for all experiments. Optimisations were run using three values of $\lambda \in \{0, 0.3, 0.5\}$, which represent an immediate, short and longer term reward allocation respectively (as it controls the eligibility trace decay rate, $\gamma\lambda$).

Since the aim of a TSP optimisation is to minimise the tour length, $L(\mathbf{x})$, the fitness is effectively $f(\mathbf{x}) = -L(\mathbf{x})$ and the reward is given by

$$r = \frac{\min\{L(\mathbf{x}) \mid \mathbf{x} \in \text{parents}\} - \min\{L(\mathbf{x}) \mid \mathbf{x} \in \text{offspring}\}}{\min\{L(\mathbf{x}) \mid \mathbf{x} \in \text{parents}\}} \tag{12}$$

In order to determine how well the agent has learned state-action utilities, we used a training and testing scheme. This involves a training phase, during which the agent produces utility estimates for all state-action pairs, followed by a testing phase in which these utility estimates were frozen and used to produce action selection probabilities for a test GA. The performance of the test GA was then compared to that of a non-learning control system.

In the experiments described here we used 150 repeats for training the system. Each training repeat used a randomly selected set of 40 cities. The action utilities found at the end of each repeat became the initial action utilities for the subsequent repeat. The initial utilities for the first repeat were all equal. Eligibility traces were reset at the beginning of each new repeat. Later investigations found that utility values converge reliably after far fewer than 150 repeats, which makes learning in an on-line fashion practicable.

A control GA, not incorporating any adaption, was used for comparison purposes. The control GA was identical to the RL-GA, but without any agent control—all action selection probabilities were equal. We used 50 repeats for testing to remove random fluctuations, and to ensure that differences in performance between the learning and control (CTRL)

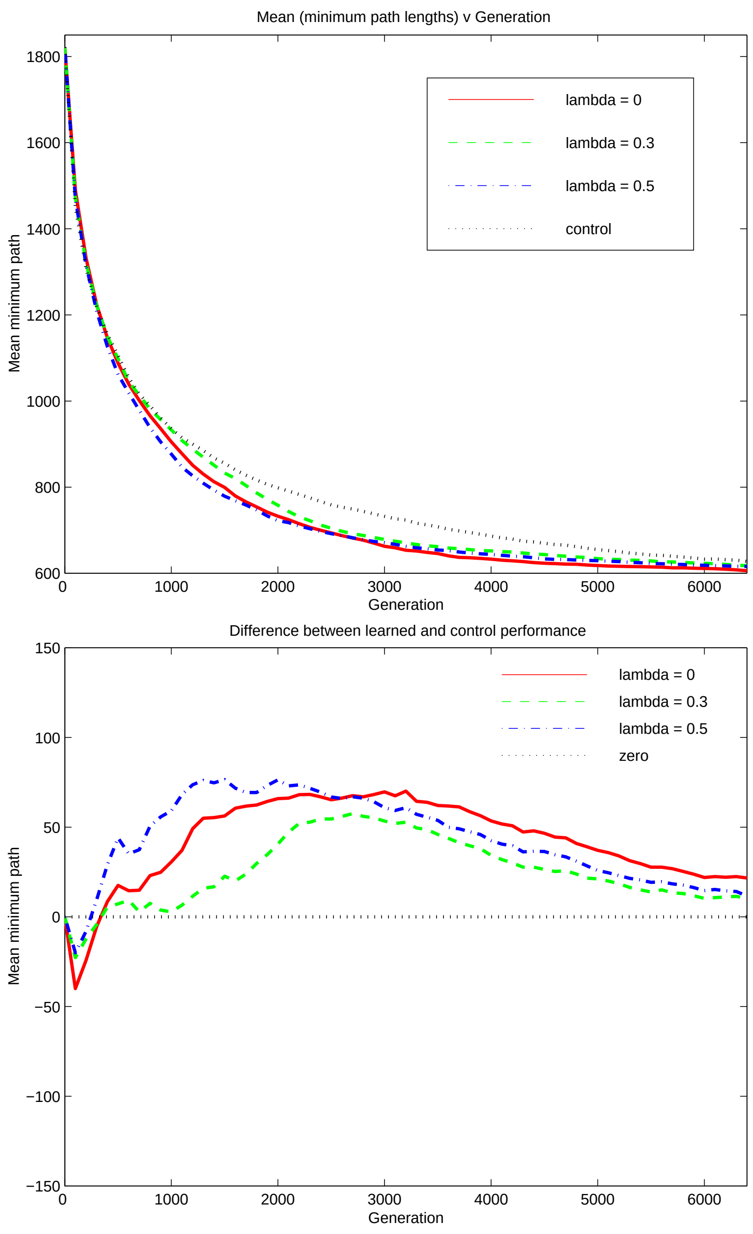

Figure 2. *Top:* Mean (over 50 repeats) tour length for control and RL-GA versus t. *Bottom:* Difference of mean tour length from control optimisation.

systems were due to the influence of the learning agent and not the sets of cities used to assess that performance. The same 50 sets of 40-cities was used for all testing and control runs.

4.1 Results

Figure 2 (top) shows the mean (over 50 repeats) minimum path length versus generation for the control GAs and the RL-GA with $\lambda = 0, 0.3, 0.5$. This gives an indication of how well each GA performed on average, throughout the optimization, using the frozen selection probabilities learned by the agent. It can be seen that the RL-GAs produce a better solution overall (for all values of λ) and also learn the solution faster. We emphasize that units for the time axis correspond to a single crossover or mutation step: since there were $M = 30$ individuals, the total run comprises about 213 generations of a conventional GA.

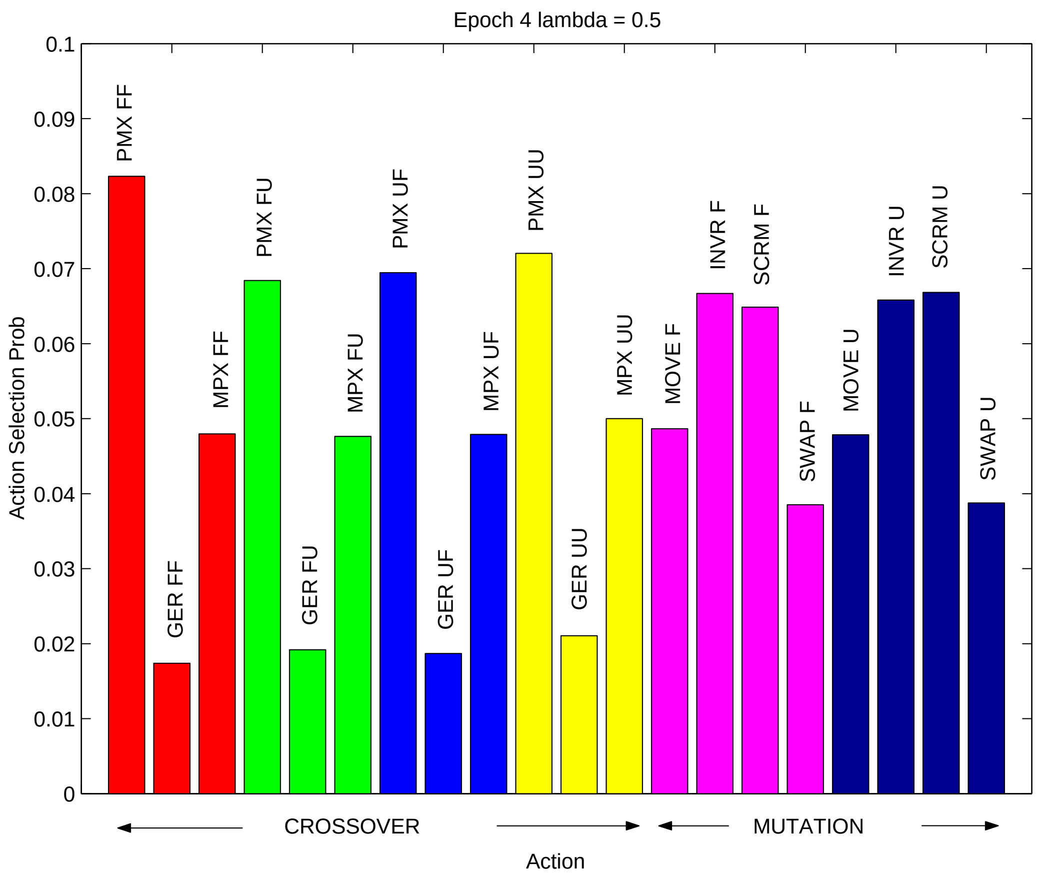

Figure 3. Selection probabilities for the genetic operators, corresponding to the $3200 < t < 6400$.

As the bottom plot of Figure 2 shows, the RL-GAs show greatest advantage over the non-adaptive system during the middle period of the optimisation. This is largely due to two effects. Recall that the component of the state corresponding to logarithmic time was partitioned into four discrete segments. The first of these covers generations 1 to 400, a relatively small number of generations compared with later segments. As a consequence the state-action table corresponding to the earliest time is relatively sparsely populated and the variances of the entries are large. This can lead to suboptimal decisions and occasionally the hybrid systems perform more poorly than the controls. At later generations the decreasing advantage of the RL-GA is simply due to the fact that the solutions have converged to the minimum path length and the control optimizers are catching up. For larger problems, requiring longer to optimize, the RL-GA would display more rapid convergence over a longer period.

Figure 2 also shows that the algorithm is relatively insensitive to the value of λ, the rate at which eligibility traces decay. Although in this instance $\lambda = 0$ (Q-learning rather than Q(λ) learning) appears optimal, small, non-zero λ is more effective in other situations. The relatively rapid decay of the eligibility traces is thought to be due to the noisy, stochastic nature of the search and to the fact that 20 actions are available, so that a non-optimal action is taken frequently at which point $e(s, a)$ is reset to zero.

Figure 3 shows the genetic operator selection probabilities learned for the final epoch of the optimization ($3200 < t \leq 6400$), with $\lambda = 0.5$. These are derived directly from the state-action utility table $Q(s, a)$ using the softmax rule (equation 5); here $\beta = 0.05$, but increasing β emphasizes the differences between the probabilities.

It is clear that the system has learned not to use the (broken) GER crossover operator.

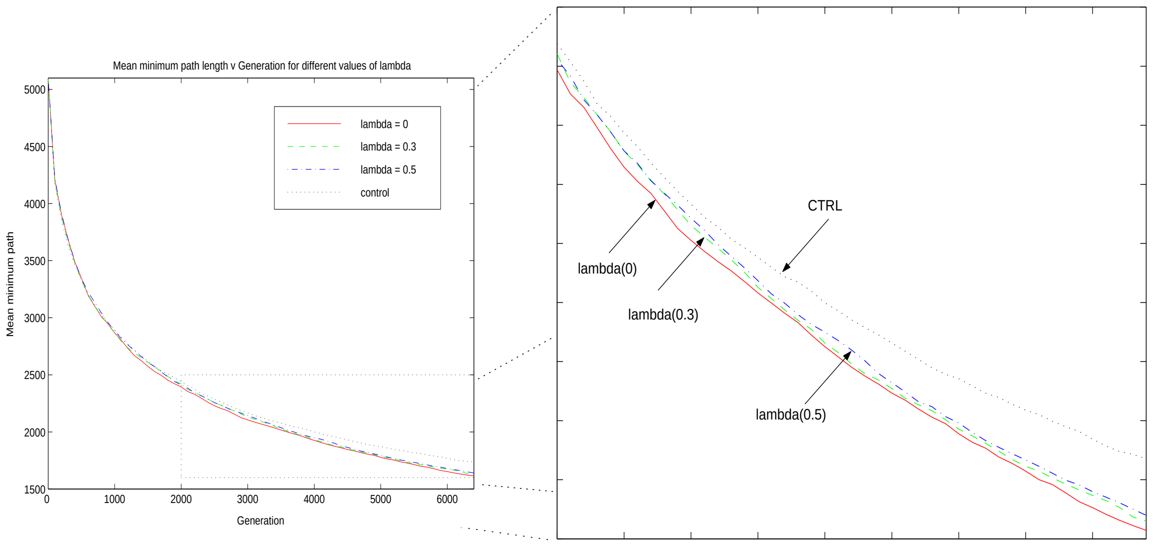

Figure 4. Mean minimum path lengths (averaged over 50 repeats) for a 100 city TSP with the RL agent's hyper-parameters unchanged from the 40 city problem. The right-hand panel shows a magnified view of the outlined box on the left.

In an additional experiment (not shown here) with a correct implementation of the GER operator the selection probabilities were found to be more uniform. In general the PMX operator is seen to be more effective than the MPX operator. It is also interesting to observe that action most likely to give a positive reward is the PMX operator applied to two fit parents. The MPX operator on the other hand appears to be no more or less effective on particular combinations of parents.

The inversion and scrambling operators (INVR and SCRM) are the most effective mutation operators, although no mutation operator appears to be particularly effective for either fit or unfit individuals.

We note that the effective crossover to mutation rate is being set by the action of the RL agent. During the initial stages of optimisation a crossover operator is selected for approximately 41% of the actions, whereas in the final stages crossover occurs more frequently, approximately 49% of actions. (We have counted the broken GER operator as a mutation operator as its principal effect is to randomise the chromosome.) Also, it should be recognised that the crossover operators frequently act as mutations. Nonetheless, like Julstrom [6], we find a moderate shift towards crossover towards the final stages of evolution.

The values of the parameters pertaining to the RL agent ($\beta = 0.05$, $\gamma = 0.5$, λ and $\alpha = 0.1$) were chosen after some experimentation on the 40 city TSP. Performance was most sensitive to the value of β; choosing β too large means that differences in $Q(s,a)$ due to statistical fluctuations are magnified into large action selection probabilities, but if β is small discrimination between the utilities is lost. To test whether the *agent* configuration is transferable to other problems, we applied the RL-GA hybrid to a 100-city TSP with these *hyper-parameters* unchanged. Figure 4 shows the mean performance on the 100 city TSP, after 150 'training' runs for learning $Q(s,a)$. Though the optimisation is still at a relatively early stage in the larger problem after 6400 generations, when compared to the smaller problem, it is still clear that the hybrid system is producing better solutions, on average, than the control.

5 Conclusions

We have demonstrated how a reinforcement learning agent may be used to control a GA, by rewarding genetic operations that tend to promote optimisation and penalising those that do not. The RL-GA system applied to a simple TSP consistently learns the global optimum faster than a non-adaptive GA. The reinforcement learning paradigm provides a formal framework for addressing questions of operator selection and tuning evolutionary algorithms.

Use of the RL paradigm necessitated modifying the GA so as to select and crossover or mutate parents one by one. The resulting scheme gives the RL agent additional control over not just the sort of operator to use, but also over the selection of which sort (fit or unfit) of individual to which to apply the operator. These initial studies on the TSP show that the RL agent learns to adapt the operator and parent probabilities throughout an optimisation run.

Although we have used a tabular representation of the state-action function here, we anticipate that the use of a neural network representation of the value function would permit more flexible control. A more compact representation would also permit the control of other aspects of the operators, for example, the length of chromosome moved or randomised in a mutation operator. It will also be important to investigate features of the population (other than the mean fitness and diversity) that indicate when a particular operator should be used or a particular parent selected.

It might be argued that we have merely swapped one set of parameters controlling the GA for another set controlling the RL agent. While these hyper-parameters do have to be set, the performance of the system is relatively insensitive to them, and the transfer of the parameters from a 40 city TSP to a 100 city TSP demonstrates that the RL agent does not have to be highly tuned. We would therefore anticipate configuring the RL agent on a smaller version of an optimisation problem before using it on the full problem.

Finally, we remark that the additional computations required for an adaptive system are relatively cheap; the main computational expense going on updating the state-action table. In an optimisation where the objective evaluation is even moderately expensive the additional cost of an RL-GA hybrid would be far out-weighed by them, even moderate, gains in improved learning rate.

References

[1] R. Argawala, D. Applegate, D. Maglott, G. Schuler, and A. Schaffer. A fast and scalable radiation hybrid map construction and integration strategy. *Genome Research*, 10:350–364, 2000.

[2] C.A. Bailey, T.W. McLain, and R.W. Beard. Fuel saving strategies for separated spacecraft interferometry. In *AIAA Guidance, Navigation, and Control Conference*, 2000.

[3] J. Boyan and A. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

[4] L. Davis. Adapting operator probabilities in genetic algorithms. In *Third International Conference on Genetic Algorithms*, San Mateo, California, 1989. Morgan Kaufmann.

[5] D. Goldberg and J. Lingle. Loci and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 154–159, Hillsdale, NJ, 1985. Lawrence Erlbaum.

[6] B.A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algoritm. In *Proceedings of the Sixth International Conference on Genetic Algorithms and their Applications*, pages 81–87, Pittsburgh, PA, 1995.

[7] P. Larranaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators, 1999.

[8] H. Muhlenbein, M. Gorges-Schleuter, and O. Kramer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[9] J.E. Pettinger. Using reinforcement learning techniques to improve the performance of genetic algorithms. Master's thesis, The University of Exeter, Exeter, UK, May 2002.

[10] J.D. Schaffer, R.A. Caruana, L.J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *International Conference on Genetic Algorithms*, pages 51–60, 1989.

[11] J.E. Smith and T.C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2):81–87, 1997.

[12] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[13] A. Tuson. Adapting operator probabilities in genetic algorithms. Master's thesis, Department of Artificial Intelligence, Univeristy of Edinburgh, Edingburgh, UK, 1995.

[14] A. Tuson and P. Ross. Co-evolution of operator settings in genetic algorithms. In T. C. Fogarty, editor, *Proc. of the Third AISB Workshop on Evolutionary Computing*, Berlin, 1996. Springer.

[15] C.J.C.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, Cambridge, UK, 1989.

[16] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proc. of ICGA'89*, pages 133–140. Morgan Kaufmann, 1989.

[17] D. Whitley, T. Starkweather, and D. Shaner. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 350–372. Van Nostrand Reinhold, New York, 1991.

[18] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*, 1995.