Direct Policy Search using Paired Statistical Tests

Malcolm J A Strens[1] MJSTRENS@DERA.GOV.UK
Defence Evaluation & Research Agency. 1052A, A2 Building, DERA, Farnborough, Hampshire. GU14 0LX. U.K.
Andrew W Moore AWM@CS.CMU.EDU
School of Computer Sciences, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh. PA 15213.

Abstract

Direct policy search is a practical way to solve reinforcement learning problems involving continuous state and action spaces. The goal becomes finding policy parameters that maximize a noisy objective function. The Pegasus method converts this stochastic optimization problem into a deterministic one, by using fixed start states and fixed random number sequences for comparing policies (Ng & Jordan, 1999). We evaluate Pegasus, and other paired comparison methods, using the mountain car problem, and a difficult pursuer-evader problem. We conclude that: (i) Paired tests can improve performance of deterministic and *stochastic* optimization procedures. (ii) Our proposed alternatives to Pegasus can generalize better, by using a different test statistic, or changing the scenarios during learning. (iii) Adapting the number of trials used for each policy comparison yields fast and robust learning.

1. Introduction

"Reinforcement Learning" is a problem description rather than a specific solution method (Sutton & Barto, 1998). The problem is to optimize the performance of an agent through trial-and-error learning of the relationship between observed states, available actions, and delayed rewards. This trial-and-error learning can take place in the physical world, or utilizing a simulation of the agent and its environment.

One way to categorize RL methods is to distinguish between *model-based* and *model-free* methods. Model-free (policy search and policy gradient) methods attempt to find a good policy directly, whereas model-based methods estimate a model of the interaction (e.g. a Markov decision process) then compute a policy from this model (e.g. by dynamic programming). There are also several methods that come part way between these two extremes. Q-Learning (Watkins, 1989) estimates state-action values which summarize rather than estimate an underlying model. An estimated state-action value function can also be used to reduce variance of a stochastic policy gradient (Sutton et al., 2000). The VAPS method (Baird, 1999) follows policy and state-action value gradients simultaneously. Ng, Parr and Koller (1999) introduced another hybrid method that uses time-slice state-occupancy distributions (model information) to aid direct search in the space of policies.

Model-based approaches can use either a certainty-equivalent or full Bayesian representation during model estimation (Barto, Bradtke & Singh, 1995; Dearden, Friedman & Andre, 1999). Intermediate to these two extremes are Interval Estimation methods that estimate confidence bounds on important quantities. In the Bayesian approach, ideal exploratory behavior is difficult to compute; some schemes use a sample from the model distribution (Strens, 2000) or backpropagation of uncertainty (Meuleau & Bourgine, 1999; Dearden et al., 1999). The main drawbacks of the full Bayesian approach are the representational problem of choosing an appropriate model class and the computational complexity of estimating the posterior distribution over models, then finding good policies.

The model-free approach is rather different and calls upon a simplicity argument: the aim of learning is to solve the learning problem (e.g. find the best policy), irrespective of the deep structure in the observed data (i.e. the system model). Why should one have to understand the world in order to exploit it? If the learning process can directly yield a solution without performing an intermediate system identification process, then the results can be robust even when the underlying system is too complex to estimate. This is the idea behind support vector machines in supervised learning, and a justification for direct policy search methods in reinforcement learning. There is much current research into policy gradient methods (Baxter & Bartlett, 1999; Sutton et al., 1999), but finding an

[1] Research performed while visiting Carnegie Mellon University.

analytical policy gradient in our application problem seems intractable, due to continuous, partially observable states and actions. Therefore we concentrate on direct search methods.

A further important consideration is that, in RL applications, a system *simulation* may also be available. A simulation might seem only to be a way to speed-up learning, eliminating real-time constraints. But simulation is much more important than this; a simulation provides a working model, as a slave process to the reinforcement learner. The learner can restart the simulation in any state, observe its hidden state, or even control its random number generator. This provides model-free policy search methods with the working model that they are missing, and provides a justification for why system identification may not be necessary: the human designer of the simulation has already provided an accurate system model at an appropriate level of abstraction.

This implies that policy search methods will be successful if they are *closely integrated* with the simulation. We concentrate on one particular form of integration: comparison of policies using identical hidden start states and sometimes identical random number sequences. This allows *paired* statistical tests to be used, making optimization procedures both faster and more reliable.

2. Direct Policy Search

We assume that learning is organized in trials variable, finite duration; furthermore a scalar return signal is available from the simulation at the end of each trial. We do not need to distinguish between the various formulations for return (e.g. integral of discounted reward, average reward, or total reward). The policy $p_?(\boldsymbol{s},\boldsymbol{a})$ is a stochastic function parameterized by $?$ for choosing action $\boldsymbol{a}$ given observed state $\boldsymbol{s}$. In general, $\boldsymbol{s}$, $\boldsymbol{a}$ and $?$ are continuous vector-valued quantities. The value $V(?)$ is the expected return when $p_?$ is used to control the simulation, given a prior distribution for starting states. In a partially observable setting, this prior distribution is defined over hidden states, rather than observation states. The RL problem is to find $?^*$ that maximizes $V(?)$.

Let random variable F_q denote the return from a simulation trial (with start state chosen from the prior). Policy search is a stochastic optimization problem: using the same policy in two simulation trials can yield different returns. The difference is attributable to stochasticity in the environment, or different starting states. Let $\{ f_i \}$ be the returns from N simulation trials (a sample of size N from F_q). The value of a policy is the limit of the empirical average return as N approaches infinity:

$$V(q) = E[F_q] = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f_i$$

2.1 Estimating the Value of a Policy

Limits on computation time require a finite N to be used in computing the sum, hence we obtain inaccurate or noisy estimates of $V(?)$. Choosing larger values of N implies that a smaller number of policies can be examined in a given total simulation time. The practical goal is an optimization method that will, with high probability, find a value of $?$ such that $V(?^*)-V(?)$ is small, given the availability of a fixed number of simulation runs. The simplest approach would be to choose very large N so that a deterministic function optimization method can be applied reliably. Another possibility is direct application of a stochastic optimization method, modeling inaccuracy in the sum as noise. Both of these naïve approaches will be shown to be an inefficient use of simulation trials.

2.2 Paired Comparison of Policies

Paired statistical tests are means for making a comparison between two samples (from the same or different populations), when there is some direct relationship between individuals of one sample and individuals of the other. For example, in determining whether a medical treatment has worked, it is possible to make some measurement (e.g. blood pressure) before and after the treatment on the *same individual*. The measurements before and after treatment are likely to be highly correlated on an individual basis: the presence of an increase or decrease in a measurement is more important than its absolute value. By taking advantage of this *pairing* between measurements, stronger statistical comparison tests can be made than if the association is ignored and only whole-group statistics are used. This “before-and-after” comparison is not the only setting in which paired tests are useful: whenever comparing two samples in which a pairwise association exists, and this association implies a correlation between measurements, paired tests can be used.

Many optimization methods (for policy search) do not require accurate evaluation of the objective function; instead they require comparative information: i.e. tests of the form “is policy *A* better than policy *B*?”. It may be possible to make these decisions reliably with smaller N than is needed to reliably estimate $V(?)$. The difference in return from any two simulation trials has 3 causes: (i) different policies; (ii) different start states; (iii) stochasticity in the environment. When comparing two policies, the effects of (ii) and (iii) might be greater than the effect of (i), so it can take many trials to “integrate out” these effects and obtain a true measure of the difference. If, when comparing two policies, the same, fixed start states are used, then the variability in the outcome will be reduced. This is a form of pairing, and so paired statistical tests can be applied.

Some optimization procedures (e.g. downhill simplex method, differential evolution, and discrete grid methods) require *only* comparisons between policies, and some of these are robust even when the comparison is sometimes wrong. We will call these methods "procedural pair-wise comparison" to distinguish them from the optimization methods that require the function value (e.g. quadratic approximation methods, simulated annealing, and some direction-set/conjugate gradient methods). Powell (1998) describes many direct search methods.

Let the random variable $F_{q,x}$ denote the result of a simulation trial with start state $\boldsymbol{x}$. If $\boldsymbol{x}$ is drawn from the known prior, a draw from $F_{q,x}$ is an unbiased estimator for $V(?)$. *Paired* statistical tests model the difference between two policies evaluated with the same $\boldsymbol{x}$: $D_x(?_1, ?_2) \sim F_{q1,x} - F_{q2,x}$. N such differences are obtained, each with a different $\boldsymbol{x}$. Paired statistics can (i) indicate whether the differences are significant at some confidence level, or (ii) estimate the probability that $V(?_1) > V(?_2)$. This allows the optimization procedure to take more trials until significance is reached, or abandon a comparison when significance cannot be achieved. We will investigate the use of the Paired *t* test and the Wilcoxon non-parametric signed rank sum test for comparing policies.

The Paired *t* test works with the sum of the *N* differences and indicates whether the mean difference is non-zero, assuming a Normal distribution. The Wilcoxon test uses the ranks of the difference magnitudes to test whether the median difference is non-zero. If the difference distribution is symmetrical, this is equivalent to testing whether its mean is non-zero. The actual difference distribution for small *N* is often highly irregular and unlikely to be either Normal or symmetrical, so it is not clear which test is best. The natural choice is the Paired *t* test because it uses the asymptotically correct statistic for large *N*. However, the Wilcoxon test pays more attention to small improvements across all scenarios, than to large changes of return in any one. Hence it might be more reliable for small *N*.

2.3 Pegasus

If causes of variability (ii) and (iii) (above) are eliminated when comparing two policies, then the whole difference in simulation return is attributable to the change in policy. The Pegasus approach converts the stochastic optimization problem into a deterministic one by evaluating policies over a fixed set of start states (scenarios) to eliminate (ii) and using fixed random number sequences to eliminate (iii). We write $f_{q,x,y}$ for the *deterministic* return of simulating policy $?$ from start state $\boldsymbol{x}$ using random number seed *y* (for a deterministic random number generator).

Given *N* start states $\{\boldsymbol{x}_i\}$ (sampled from the known prior) and *N* unique seeds $\{y_i\}$, the return from simulation becomes a sum of deterministic functions:

$$V_{PEG}\left(q \left| \{(\boldsymbol{x}_i, y_i)\}\right.\right) = \frac{1}{N}\sum_{i=1}^{N} f_{q,\boldsymbol{x}_i,y_i}$$

A deterministic optimization method can then be used to find the maximum of $V_{PEG}(?)$. A perceived advantage of the Pegasus method is that the gradient of $V_{PEG}(?)$ can be computed numerically, so a wide selection of optimization methods are available. However, in *m* dimensions, computing a numerical gradient requires at least *mN* function evaluations. (For our application, this equates to 12 x 128 = 1536 simulation runs.) There is also a risk that $V_{PEG}(?)$ is not smooth because a small change in policy can affect the way the fixed sequence of random numbers are used. $V_{PEG}(?)$ is likely to be much less smooth than $V(?)$, because $V(?)$ averages over all random number sequences and starting conditions. This could seriously affect convergence of gradient-descent methods.

The number of scenarios required for $V_{PEG}(?)$ to be a good estimate for $V(?)$ is a polynomial in horizon time, and in various measures of the complexity of states, actions, rewards, system dynamics, and policies (Ng & Jordan, 1999). We will assess Pegasus with numbers of scenarios much less than that required by the bound. Paired statistical tests can be applied to the set of differences { $f_{q1,x_i,y_i} - f_{q2,x_i,y_i}$ }. This provides a search algorithm with additional information that can be related to the true optimization objective - maximizing $V(?)$, not $V_{PEG}(?)$. (The paired statistical tests infer information about the expected return over the *whole population* of scenarios, given only the returns from a sample.)

The actual values $\{(\boldsymbol{x}_i, y_i)\}$ can be changed repeatedly during the course of an optimization procedure, in order to reduce the risk of a type of overfitting that we identify in our experiments with Pegasus. The disadvantage is that the objective function changes each time, affecting convergence properties of the optimization procedure.

3. Optimization procedures

We have identified several ways to approach the policy search: (i) stochastic optimization of $V(?)$; (ii) procedural pair-wise comparison of policies with stochastic policy differences $D_x(?_1, ?_2)$; (iii) optimizing a closely related stationary deterministic function $V_{PEG}(?)$; (iv) procedural pair-wise comparison of policies with a non-stationary deterministic objective function (Pegasus with changing scenarios). The ideas (i) through (iv) need to be combined with an optimization procedure. We chose three different procedures to demonstrate several ways in which paired statistical tests can help in the search for a good policy.

3.1 Random Search

Random search is the simplest method to implement and is used, firstly, as a control to demonstrate that each task is non-trivial. Values for $?$ are chosen uniformly from a region of interest (set to the whole parameter space in control experiments). The corresponding policies are compared, using a fixed number of scenarios, with the best policy found so far, which is replaced accordingly. We later show that random search with a dynamic region of interest can be made into an effective policy search algorithm by adapting *N*, using paired statistical tests and asymmetric confidence intervals. The bounds of the region of interest are shrunk towards the current best point at regular intervals. Each dimension of the region of interest reduces by a factor of $\sqrt{2}$, and this happens 20 times during learning.

3.2 The Downhill Simplex Method

The downhill simplex method (DSM) is a direct search method which "crawls" up or down a function, without computing numerical gradients (Nelder & Mead, 1965). The state of the algorithm is not a single point, but a simplex of m+1 vertices in the m-dimensional search space. New points are proposed based on geometrical operations on the vertices of the simplex. A simplex vertex is replaced by the new proposal based on comparisons with the worst, second-worst or best existing vertex. Usually the proposal is a linear combination of one vertex with the centroid of the others, but occasionally progress stops and all vertices are moved towards the best. Likas & Lagaris (1999) have used a similar algorithm for policy search.

DSM is very efficient in terms of the number of function evaluations required and is reasonably robust to incorrect decisions. However, it has several failings including the risk of stagnation and the fact that it only finds a local minimum. These risks can be reduced by improvements such as random restarts and oriented restarts (Kelley, 1997; Wright, 1995). DSM can be implemented using only pair-wise comparisons of policies, rather than reference to the actual function value. With some modification, parametric or non-parametric paired tests can be used for these comparisons.

3.3 Differential Evolution

Differential Evolution (DE) is a method based on the principles of genetic algorithms, but with crossover and mutation operations that work directly on continuous-valued vectors (Storn & Price, 1995). It has been shown to be effective for global optimization of reasonably complex functions. DE's proposals are much more random than DSM, but more sophisticated than the random search described above. DE keeps a larger population of points than DSM (at least 2m). At any given time, one candidate in the population is chosen (systematically) for improvement. A separate, random "parent" is chosen from the population. Then the vector difference between two more randomly chosen population points is added to the parent (weighted by a scalar parameter, *F*). Some implementations add two or more such vector differences. Crossover takes place between this and the candidate, to obtain a proposal point. A test is then performed for whether this is better than the candidate, and if so, the proposal replaces the candidate. Crossover is implemented here by selecting each element of the proposal vector from either the candidate or the new vector with some pre-specified probability r .

DE has a very useful property; replacing any one population member due to an occasional incorrect comparison is not catastrophic. It may suffice that the comparison be unbiased, and correct only slightly more often than 50%. This means that it should be possible to use many fewer trials (or even *N*=1) for each comparison, given a reasonable population size. It is again possible to use paired statistical tests for the individual comparisons.

4. Multi-Pursuer Evader Problem Description

We have designed an application problem in which two pursuers must co-operate to defeat a reacting evader. This yields a 12-dimensional policy search problem (6 dimensions for each pursuer). The evader's policy is fixed but stochastic. This is a highly simplified version of guiding missile salvoes against reacting targets. Space permits us only to give a brief description.

4.1 Simulation

Two pursuers and one highly maneuverable evader move in a 2-dimensional plane. The pursuers start close to the origin, always with the same initial direction of motion, but slightly different speeds. The initial location of the evader is chosen randomly. The pursuers move twice as fast (on average), so they have much larger 'turning circles' than the evader. At each time step (0.002 s), the evader and pursuer positions are updated, and every 4th time step, each chooses an action in the range [-1,1]. These actions are scaled by the maximum accelerations to give a turning effect.

Each trial ends when 4 seconds have elapsed (failure), or one of the pursuers has come within one "miss distance" ($D = 2$ meters) of the evader (success). We assume that the pursuers have perfect information about the position of the evader and vice versa. There is a latency of 0.008 seconds in the measurement process. The goal is to find optimal policies for the pursuers, in order to maximize expected return. The return from a simulation trial is given by $2D/(r_{\min} + D)$ where $r_{\min}$ is the distance of the closest pursuer from the evader at the end of the trial. The return is in the range $[0,1]$.

The evader stochastically switches between 4 internal states, based on the distance of the closest pursuer. The pursuers are unable to observe this hidden state. The evader continues in its initial direction of motion until the

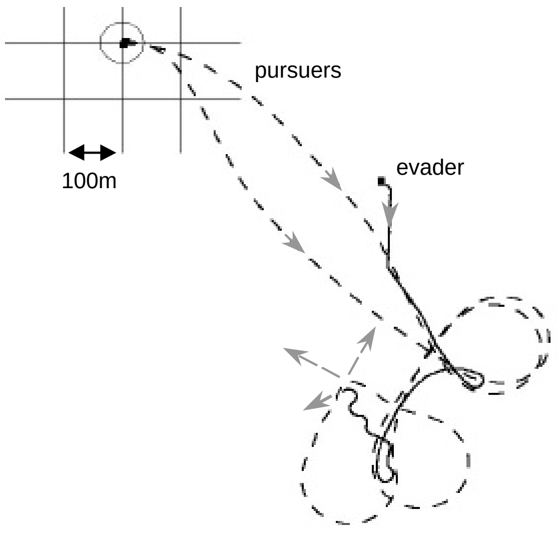

Figure 1. Pursuer-evader simulation trial.

closest pursuer is within approximately 300 meters, when it switches to the "evade" state with a certain probability at each time step. In the evade state, the evader turns directly away from the closest pursuer, to maximize the time before they meet. If the closest pursuer is within approximately 20 meters, stochastic transitions into a maneuver state are possible. The evader enters a circular trajectory (maximum turning rate) to throw the pursuer off course. Similar rules allow the evader to return to its original course after a successful maneuver.

4.2 Pursuer Policy Parameterization

The two pursuers must co-operate to defeat the evader by developing a strategy that "hedges" against possible maneuvers by taking different paths. The observed state for each pursuer is a vector of 5 measurements:

a. Relative angle of the target f
b. A function of evader range: $\exp(-r_i/100)$
c. A function of sight line rate: $\tanh(0.016p\, \dot{f})$
d. Range rate: $\dot{r}_i/500$
e. Relative range of pursuers $(r_i - \bar{r})/(r_i + \bar{r} + D)$

Only (e) conveys information about one pursuer to the other, through the mean pursuer distance $\bar{r}$. We construct a 6-element vector $\boldsymbol{s}$ from these measurements and a constant (equal to 1) in the sixth element. The policy for pursuer i is then given by $a_i = \tanh(2a \cdot \boldsymbol{s}_i)$ in parameters a , ignoring time subscripts and latency for clarity.

The policy deterministically maps the observed state through a linear function, then a sigmoid squashing function (tanh), to yield an action for the pursuer. Concatenating the values of a for the two pursuers yields a 12-element vector ? (the target for optimization). (DSM

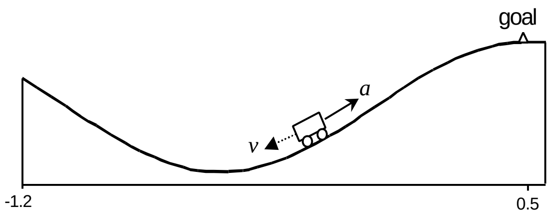

Figure 2. The mountain car problem.

required an intermediate linear transformation of ? to eliminate one symmetry in the policy space.)

4.3 Example

Figure 1 shows the paths taken by pursuers and evader in a simulation trial of a partially trained system. The pursuers start close to the origin, whereas the evader's initial state is chosen randomly. The evader observes the pursuers early in the trial and starts to fly away from them. As the pursuers get closer, the evader then makes two successful maneuvers, each throwing the pursuers off course. It is eventually defeated when the evaders approach simultaneously from very different directions. Arrows show final velocity vectors.

There are many interesting variations on this system: the evader can learn new behavior (a competitive game) or the pursuers could learn one at a time (a co-operative game). Harmon & Baird (1996) address some of these issues, and apply residual advantage learning to a similar pursuer-evader game. Initial learning can also be made faster by searching only for one pursuer's policy, giving the other pursuer a "left-handed" version of it.

5. The Mountain Car Problem

We also tested our methods the mountain car problem (Figure 2). The goal is to cause an under-powered car to climb a hill. The car must gain kinetic energy by accelerating in alternating directions, each time reaching higher positions on the valley walls. We started with the formulation given by Sutton & Singh (2000). The car's state consists of its one-dimensional position *p* and velocity *v*. It moves under gravity and its own acceleration a_t on a hill with height $\sin(3p)$. However, using their formulation, a near-optimal policy was obtained trivially, by setting a_{t+1} to have the same sign as v_t, maximizing the car's energy. (The reason for this is that when it reaches the left bound for *p*, velocity is reset, so useful energy is automatically maximized.)

A more difficult control problem is obtained if we cause the trial to *end* with failure at the left bound, and also give lower return if the car reaches its goal state with excess energy (Munos & Moore, 1999). Uniform noise was also added to the acceleration representing the effect of wind turbulence. Even with these changes the policy search

Table 1. Pursuer-evader results using DSM and random search.

RETURN (%)	2048 TRIALS		65536 TRIALS	
$N = 64$	ON-LINE	TEST	ON-LINE	TEST
RANDOM SEARCH	1.5 ± 0.2	7.2 ± 2.0	13 ± 0	28 ± 2
PEGASUS	28 ± 1	33 ± 1	60 ± 3	34 ± 2
PEGASUS (WX)	5.3 ± 0.1	34 ± 3	46 ± 2	42 ± 1
SCENARIOS (WX)	4.3 ± 0.2	17 ± 0	44 ± 1	41 ± 1
UNPAIRED	4.6 ± 0.2	20 ± 1	40 ± 2	40 ± 2

problem was solved easily with a few Pegasus scenarios. Therefore we also introduced hidden state, in the form of the car's mass *M*, and the steepness of the slope, *S*. The dynamics are given by:

$$v_{t+1} = bound\left[v_t + \left\{a_t + 10U[-1,1] - 2.5S\cos(3p_t)\right\}\partial t/M\right]$$
$$p_{t+1} = bound\left[p_t + v_{t+1}\right]$$

The *bound* operator ensures $-0.07 \le v \le 0.07$ and $-1.2 \le p \le 0.5$; $a_t \in \{-1,1\}$; $\partial t = 0.001$. Initially $v = 0$, p has $U[-0.7,0]$; log *M* and log *S* are drawn from $U[-1,1]$. Trials end when $p \in \{-1.2, 0.5\}$ or $t = 1024$. The return from each trial is 0 for an immediate failure, and 1 for stopping perfectly at the hilltop. Shaping returns are also used[1]. The deterministic policy, parameterized by $?$, is given by a thresholded polynomial in $\hat{p}$ and $\hat{v}$ (re-scaled versions of *p* and *v* in the range [-1,1]):

$$a = sign\left(\left(1, \hat{p}, \hat{p}^2, \hat{p}^3, \hat{v}, \hat{v}\hat{p}, \hat{v}\hat{p}^2, \hat{v}\hat{p}^3\right)^? \cdot q\right)$$

6. Experimental Comparison

We will first evaluate whether there is a benefit to using pairing of scenarios and/or random number seeds, and compare the options for paired tests (Paired *t* or Wilcoxon). Then we will consider how *N* can be adapted during learning in random search and DSM. Finally, we apply paired testing to speed-up DE.

6.1 Evaluation of Criteria for Comparing Policies

The aim of the first experiment is to compare the different criteria when comparing policies with a small, fixed, number of scenarios. This should show whether there are any benefits in using paired testing, and which paired test should be used. The DSM optimization procedure was used except in the control experiments (random search).

The methods compared were (i) random search (using Pegasus-style comparisons), (ii) Pegasus, (iii) Pegasus modified to use the Wilcoxon statistic, (iii) fixed scenarios, but no pairing of random number seed, and (v) unpaired comparison (random scenario for every trial). An independent test set was used to evaluate the policies learnt by each of the methods. Each test example was a Pegasus-style configuration (start state and random number seed), and the same test set was used for all the methods. Therefore test set performance for a given policy was deterministic. On-line returns (during learning) in the subsequent results will usually be lower than test performance because they includes the cost of exploration. For each RL method, test performance is always measured using the best policy found so far (i.e. currently *believed* best based on performance during learning).

Table 1 shows results for the pursuer-evader problem, expressed as a % of the return which would be achieved if every trial led to capture of the evader[2]. Each is the average of 16 runs (with standard error), and the test set size is 1024 (minimizing test bias). Results are shown after 2048 and 65536 trials (equivalent to 32 or 1024 policy evaluations). The best test set performance is obtained by using the Wilcoxon test when comparing policies. This is significantly better than the standard Pegasus algorithm after 65536 trials. It is also apparent that on-line performance of the standard Pegasus algorithm can be highly misleading; on-line performance reaches 60% but test performance never exceeds 34%. Using the Wilcoxon test, there is much less evidence of this type of overfitting (4%). The results for 2048 trials do indicate that Pegasus-style pairing of random number seeds significantly improves learning rates, but best performance at 65536 trials was obtained if only the start states were paired.

We also verified that overfitting in Pegasus decreases with the number of scenarios: 34% for 16; 30% for 32; 26% for 64; 8% for 128; (not shown in table). (A simple way to reduce overfitting is to change the Pegasus scenarios occasionally during learning, but this can affect convergence of the DSM.)

Table 2 shows results for the mountain car problem (Returns are scaled by 100.0 to express as a percentage.) Many fewer scenarios (8) and trials (about 200) were needed to obtain good results. With this number of scenarios, significant overfitting (11%) was again apparent when using Pegasus. The Wilcoxon test slightly improved test performance, but did not decrease overfitting. The method using paired scenarios performed best (but not significantly better than with unpaired scenarios or modified Pegasus).

[1] Return was $(t/4096)$ for exits at the left (proportional to trial duration); $0.25 + (0.4159\,p_{\max} + 0.4991)^4$ if trial duration was 1024 steps (a function of maximum position reached), and $(1 - 7.142|v_t|)$ for reaching the hilltop (dependent on terminal velocity).

[2] The actual achievable return is less than 100% because some starting states provide an impossible task for the pursuer.

Error! Not a valid link.

Figure 3. Adapting *N* in random search (pursuer-evader).

6.2 Adapting the Number of Scenarios

In the next set of experiments, we try to make better use of computation time by adapting the number of trials used to evaluate each policy. Ideally, we would decide how reliable each policy comparison needs to be, and increase *N* until the corresponding confidence level is reached (in the Paired *t* test). Note that in the context of minimizing expected cross-validation error this kind of adaptive paired test has been used with success (Moore & Lee, 1994).

The random search method is very suited to this form of adaptation because it compares new proposals with the *best* point found so far; usually only a small number of trials will be required for enough statistical significance to reject a proposal. The confidence levels do not need to be the same for acceptance and rejection. We required 90% (two-tailed) significance level for rejecting a (worse) proposal, but 99% (two-tailed) significance for accepting a (better) proposal (subject to $1<N<256$). This is highly efficient because we know that only a very small proportion of proposals will be accepted, so *N* will only be large when the comparison is required to be very reliable.

Figure 3 compares the adaptive method against fixed-*N* (using 16 or 64 Pegasus scenarios). Although fixed N=16 may learn faster initially, the solution it finds generalizes poorly (test set performance is 35%). Fixing N=64 learns more slowly, but eventually obtains better test set performance (36%). The adaptive method averaged only N=25, but gave test set performance of 42%, so it generalized much better than fixed N=64. With unpaired comparisons and N=16, final test performance was only 26%. *This demonstrates that paired statistical tests can be used to convert a very simple optimization procedure into a powerful policy search method.* This adaptive random search also found near-optimal policies for the mountain car (82.5±0.2% after 4096 trials).

6.3 Adapting *N* in the Downhill Simplex Method

DSM is not suited to this type of adaptation of *N* using only confidence levels, because most comparisons are between a new proposal and the *worst* point in the simplex. Many of these comparisons will not yield statistically significant results even with very large numbers of scenarios. However, it is still useful to change *N* during learning. An upper limit on *N* is increased during learning (from 16 to 128), with a confidence level fixed at 95%. This means that we initially accept overfitting in order to speed-up learning. At each

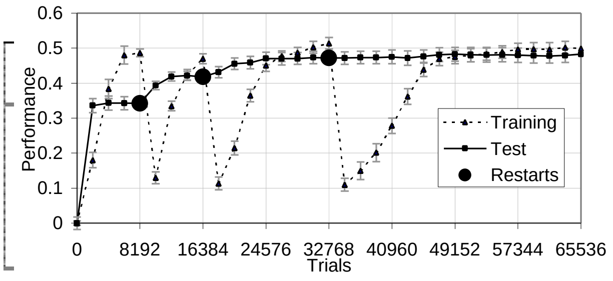

Figure 4. DSM with increasing *N* (pursuer-evader).

doubling of the limit, the DSM is restarted (at the best policy found so far; see Figure 4). Each restart helps the system to find a policy with better generalization performance.

Using Pegasus-style comparisons, final test performance was 48% (Wilcoxon) or 45% (Paired *t*); significantly better than all results we obtained with fixed values of *N*. Pairing only the start states (and not the random number seeds), 50% performance was attained (both tests). This improvement was significant for the Paired *t* test, but not for the Wilcoxon test.

6.4 Differential Evolution Results

We applied differential evolution to the pursuer-evader problem with a population size of 32, r = 0.5, F = 0.2, and using 2 vector differences in the mutation step. Using Pegasus-style paired comparisons with N=2, convergence was achieved. Each comparison used unique scenarios. Test performance after 65536 trials was 47±2%. This performance was certainly a result of using paired comparisons; using unpaired comparisons gave 27±2%

These results are surprising, because we normally expect genetic search methods to learn much more slowly than direct search methods. The reason for the good performance is that DE is able to utilize unreliable comparisons much better than direct search methods. Using N=2 allowed many more policies to be evaluated than with a direct search method (for which *N* is much larger). Even if a large proportion of comparison decisions are wrong, the fitness of the population as a whole will increase as each nearly[3] unbiased comparison is made. The "population-averaging" effect of genetic methods is not only helping to search a large space of solutions, but also to account for bias in the returns obtained during Pegasus-style policy comparisons. *Therefore stochastic search methods can use Pegasus-*

[3] As *N* becomes small, DE will prefer policy *A* to policy *B* if the return of *A* is larger in the majority of scenarios. This does not necessarily mean *A* has higher expected return.

style comparisons with small numbers of trials per policy very effectively.

DE was less reliable on the mountain car problem (with 4096 trials). In a typical configuration, it yielded 73% test performance (standard error 3%). However in 7 of the 16 runs it found near-optimal policies (average 81.8%).

7. Conclusions

Best overall performance on our pursuer-evader application (50% average; 68% maximum) was obtained using DSM, with paired start states, Wilcoxon comparisons and adaptive *N*. DE with *N*=2 also performed very well. The mountain car problem was solved easily by most methods.

We have shown that paired statistical tests provide a significant advantage over naïve comparison of policies. Although Pegasus is based on pairing of both start states and random number seeds, it was usually better to pair only the start states, unless other measures were taken to prevent overfitting. The Wilcoxon signed rank sum test often outperformed the comparison of average returns (i.e. the Paired *t* test statistic) used by Pegasus. The improvement was attributable to a reduction in overfitting.

By fixing the confidence interval used in the paired statistical tests, we also showed that the number of trials for each policy could be adapted to speed-up learning (for random search and DSM). The appropriate confidence interval depends on the nature of the search algorithm.

Most importantly, we showed that paired statistical tests were very effective at improving the performance of *stochastic* optimization methods such as random search and DE. Random search can use asymmetric confidence intervals to obtain an appropriate number of trials for each proposed policy automatically. In differential evolution, the number of scenarios can be very small (if each comparison is made using different scenarios) because the DE population has a powerful averaging effect. This leads us to believe that stochastic optimization methods (but not necessarily differential evolution), combined with paired statistical tests, will be a powerful tool for policy search in many more than 12 dimensions.

Acknowledgements

This research was funded by UK Ministry of Defence (Corporate Research Technology Group 3).

References

Baird, L. C. (1999). *Reinforcement learning through gradient descent.* Doctoral Dissertation, Carnegie Mellon University, Pittsburgh.

Barto, A. G.; Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence 72,* 81-138.

Baxter, J., & Bartlett P. L. (1999). *Direct gradient-based reinforcement learning: I. Gradient estimation algorithms* (Technical report.) Research School of Information Sciences and Engineering, Australian National University.

Dearden, R., Friedman, N., & Andre D. (1999). Model based Bayesian exploration. *Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence.* San Francisco: Morgan Kaufmann.

Harmon, M. E., & Baird III, L. C. (1996). *Multi-player residual advantage learning with general function approximation* (Technical Report WL-TR-1065). Wright Laboratory, Ohio.

Kelley, C. T. (1997). *Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition* (Technical Report CRSC-TR97-2). North Carolina State University.

Likas, A., & Lagaris, I. E. (1999). Training reinforcement neurocontrollers using the polytope algorithm. *Neural Processing Letters, 9*, vol. 9:2, 119-127.

Meuleau, N., & Bourgine, P. (1999). Exploration of multi-state environments: local measures and back-propagation of uncertainty. *Proceedings of the Twelfth International Conference on Machine Learning*. San Francisco: Morgan Kaufmann.

Munos R., & Moore, A. W. (1999). Variable resolution discretization for high-accuracy solutions of optimal control problems. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. San Francisco: Morgan Kaufmann.

Nelder J. A., & Mead R. (1965). A simplex method for function minimization. *Journal of Computing, 7*, 308-313.

Ng A., & Jordan M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*.

Ng A., Parr R., & Koller D. (1999). Policy search via density estimation. *Advances in Neural Information Processing Systems 11 (Proceedings of the 1998 Conference)*. MIT Press.

Powell, M. J. D. (1998). Direct search algorithms for optimization calculations. *Acta Numerica, 7,* 287-336.

Storn, R., & Price, K. (1995). *Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces* (Technical Report TR-95-012), International Computer Science Institute, Berkeley, California.

Strens, M. J. A. (2000). A Bayesian framework for reinforcement learning. *Proceedings of the Sixteenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Sutton, R. S., & Barto, S. (1998). *Reinforcement learning.* Cambridge, MA: MIT Press.

Sutton, R. S., McAllester, D., Singh, S., Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems 12 (Proceedings of the 1999 Conference)*, 1057-1063. MIT Press.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral Dissertation, Department of Psychology, Cambridge University, U.K.

Wright, M. H. (1995) Direct search methods: once scorned, now respectable. In D. Griffiths & G. Watson (Ed.), Numerical Analysis, Pitman Research Notes in Mathematics. London: Addison Wesley Longman.