Journal of Machine Learning Research 3 (2002) 921-950 Submitted 12/01; Published 12/02

Policy Search using Paired Comparisons

Malcolm J. A. Strens MJSTRENS@QINETIQ.COM
Guidance & Imaging Solutions
QinetiQ
Ively Road, Farnborough
Hampshire GU14 0LX, UK

Andrew W. Moore AWM@CS.CMU.EDU
School of Computer Sciences
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA

Editors: Carla E. Brodley and Andrea Danyluk

Abstract

Direct policy search is a practical way to solve reinforcement learning (RL) 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 and Jordan, 2000). We evaluate Pegasus, and new paired comparison methods, using the mountain car problem, and a difficult pursuer-evader problem. We conclude that: (i) paired tests can improve performance of optimization procedures; (ii) several methods are available to reduce the 'overfitting' effect found with Pegasus; (iii) adapting the number of trials used for each comparison yields faster learning; (iv) pairing also helps *stochastic* search methods such as differential evolution.

Keywords: Reinforcement Learning, Policy Search, Experiment Design

1. Introduction

"Reinforcement Learning" is a problem description rather than a specific solution method (Sutton and 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 using 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 many methods that come part way between these two extremes. Q-Learning (Watkins, 1989) estimates state-action values which summarize (rather than estimate) the 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 simul-

©2002 QinetiQ Ltd and Andrew W. Moore.

taneously. Ng et al. (1999) introduced another hybrid method that uses time-slice state-occupancy distributions (model information) to aid direct search in the space of policies.

The model-free approach calls upon a simplicity argument: the aim of learning is to solve the learning problem (find the best policy), irrespective of the deep structure in the observed data (i.e. the system model). If the learning process can directly yield a solution without performing an intermediate system identification process, then the results may be robust even when the underlying system is too complex to estimate. There is also much current research into policy gradient methods (Baxter and Bartlett, 1999, Sutton et al., 2000), which aims to find local optima in the parameter space of a stochastic policy. We concentrate on *direct* search methods that do not require a gradient estimate and aim to find a global optimum policy.

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 model-free policy search methods will be successful if they are closely integrated with the simulation; the learner should be able to access the simulation in exactly the same way that it would use a learnt, internal model of the world.

This paper concentrates 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. Learning with a genuine physical system, rather than a simulation, would not allow the control of stochasticity through the specification of a particular random number sequence. However, the other methods we describe based on paired or fixed start states for episodic learning are applicable to learning in a physical system.

In policy search the critical dilemma is how much computation time to assign to each policy evaluation. In continuing tasks this is measured as a number of interaction steps and in episodic tasks it is a number of trials. The trade-off is between accuracy of the empirical estimate of the policy's value, and the number of different policies that can be considered during learning. The correct solution is unclear except in very simple problem domains (*e.g.* deterministic, fully observable, finite environments), and will depend on the particular policy search method. In many cases an adaptive computation time is appropriate. Moore and Lee (1994) used adaptive numbers of function evaluations for model selection in their RACE algorithm, and Teller and Andre (1998) used a similar method ('rational allocation of trials') for tournament selection in evolutionary search. In both algorithms, 'blocking' (Box et al., 1978) was used to increase the reliability of decision-making. Blocking uses similar 'cases' to perform each function evaluation in order to eliminate bias. A case is some aspect of the function evaluation that is under control of the search procedure, for example the initial position of a robot in a reinforcement learning trial. When comparing two policies, blocking corresponds to a paired statistical test. The main emphasis of this paper is to address the accuracy/computation-time trade-off problem in the context of direct search for parameterized policies. Technical detail of the algorithms and application problems is given to make the experiments repeatable.

Section 2 introduces policy search using paired statistical tests. Section 3 then describes three search procedures that can make use of these tests to search for a good policy. Section 4 describes two application problems of differing complexity, which are used to perform an evaluation of paired policy search methods in Section 5. Section 6 concludes with a summary of the main findings and suggests directions for further research. Complete summary tables of the experimental results are given in Appendix A.

2. Policy Search using Paired Comparisons

We assume learning is organized in trials of variable, finite duration and a scalar return signal is available after each simulation trial. The policy $\pi_w(x,a)$ is a stochastic function parameterized by w for choosing action a given observed state x. In general, x, a and w are continuous vector-valued quantities. The value $V(w)$ is the expected return when π_w is used to control the simulation, given a prior distribution for starting states. In a partially observable setting, this prior is defined over hidden states, rather than observation states. The RL problem is to find w^* that maximizes $V(w)$.

Let random variable F_w denote the return from a simulation trial (with starting 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 and policy, or different starting states. Let $\{f_i\}$ be the returns from N simulation trials (a sample of size N from F_w). The value of a policy is the limit of the empirical average return as N approaches infinity:

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

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 $V(w)$. 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 w such that $V(w^*) - V(w)$ 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, modelling inaccuracy in the sum as noise. Both of these 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 when there is a direct relationship between members of one sample and members of the other. For example, in determining whether a medical treatment has worked, some measurement (e.g. blood pressure) is taken before and after treatment of a group of individuals. For every measurement in the "before" sample, there is a corresponding measurement in the "after" sample. By taking advantage of this *pairing* of measurements, more reliable statistical comparison tests can be made (than if the correspondence is ignored and only whole-group statistics are used).

For example, Table 1 shows a set of $N = 8$ paired measurements. Each column in the table corresponds to the same individual (or *case*, or *scenario*). Let the null hypothesis be that there is no difference between the expected values of measurements A and B ($\mu_A = \mu_B$). A *t test* is used to determine whether $\mu_B > \mu_A$. Applying the unpaired version of the test yields a one-tailed probability 0.29 (71% confidence), but the paired version yields 0.025 (97.5% confidence). Hence it is only possible to be confident that $\mu_B > \mu_A$ by exploiting the pairing of the measurements. (If one row of measurements are randomly reordered, then the pairing is lost and the paired test usually yields a less confident answer than the unpaired one.)

Individual	1	2	3	4	5	6	7	8
Measurement A	84	65	54	90	38	58	62	-30
Measurement B	90	71	64	86	35	69	84	0

Table 1: Example of paired measurements.

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 a smaller number of trials (N) than is needed to accurately estimate $V(w)$. The difference in return from any two simulation trials has 3 causes: (i) different policies; (ii) different start states; (iii) stochasticity in the environment (or policy). 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" this variability and be confident in which policy is better. If, when comparing two policies, the same, fixed starting states are used, then the variability in the outcome will be reduced. This means that paired statistical tests are applicable.

Some optimization procedures (*e.g.* downhill simplex method, differential evolution, and discrete grid methods) require only comparisons between policies. We will call these methods "procedural pairwise comparison" to distinguish them from the optimization methods that require the numerical function value (*e.g.* quadratic approximation methods, simulated annealing, and some direction-set or conjugate gradient methods). Some optimization procedures are also robust when the comparison operation is unreliable. Powell (1998) describes many optimization procedures.

Let the random variable $F_{w,x}$ denote the result of a simulation trial with starting state x (also called a *scenario*). If x is drawn from the known prior, a single draw from $F_{x,w}$ is then an unbiased estimator for $V(w)$. Paired statistical tests model the difference between two policies evaluated with the same starting state: $D_x(w_1, w_2) \sim F_{w_1,x} - F_{w_2,x}$. When comparing two policies, N such differences are obtained, each with a different x. Paired statistics can indicate whether the differences are significant at some confidence level. 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 the differences are Normally distributed. The Wilcoxon test uses the ranks of the difference magnitudes to test whether the median difference is non-zero. If the difference distribution is symmetric, 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 symmetric, 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 starting states (scenarios) to eliminate (ii) and using fixed random number sequences to eliminate (iii). We write $f_{w,x,y}$ for the deterministic return of simulating policy w from starting state x using random number seed y (for a deterministic random number generator). Given N starting states x_i (sampled from the known prior) and N unique seeds y_i, the return from simulation becomes a sum of deterministic functions:

$$V_{PEG}(w|\{(x_i, y_i)\}) = \frac{1}{N}\sum_{1=1}^{N} f_{w,x_i,y_i}$$

A deterministic optimization method can then be used to find the maximum of $V_{PEG}(w)$. A perceived advantage of the Pegasus method is that the gradient of this function 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 thousands of simulation runs.) $V_{PEG}(w)$ is likely to be much less smooth than $V(w)$, because $V(w)$ averages over all random number sequences and starting conditions. This could seriously affect convergence of gradient-descent methods. The effectiveness of Pegasus also strongly depends on the way the simulation uses the random number sequence. For example, if a small change in policy can affect the way the fixed sequence of random numbers are interpreted, then $V_{PEG}(w)$ has discontinuities.

The number of scenarios required for $V_{PEG}(w)$ to be a good estimate for $V(w)$ is a polynomial in horizon time, and in various measures of the complexity of states, actions, rewards, system dynamics, and policies (Ng and Jordan, 2000). 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 $\{f_{w_1,x_i,y_i} - f_{w_2,x_i,y_i}\}$. This provides a search algorithm with additional information that can be related to the true optimization objective - maximizing $V(w)$, not $V_{PEG}(w)$. (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 $\{(x_i, y_i)\}$ 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, possibly affecting convergence properties of the optimization procedure. Nevertheless, we will show that this "changing scenarios" approach is highly effective.

3. Optimization Procedures

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

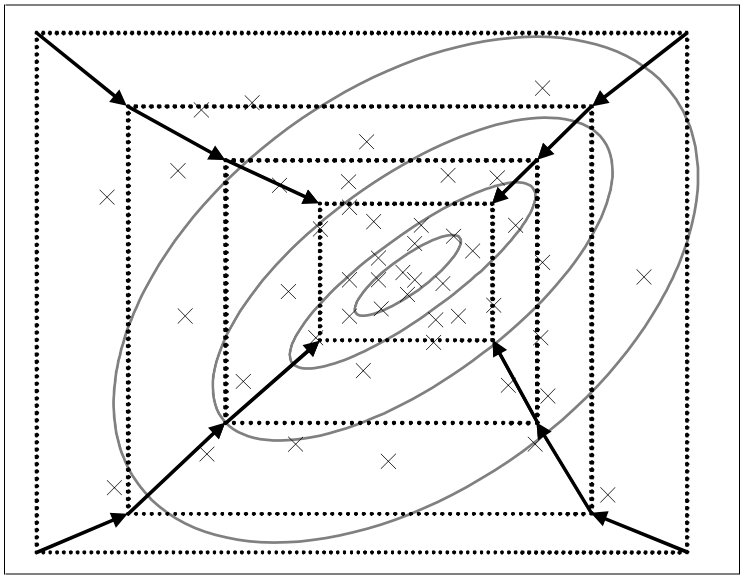


Figure 1: Optimization using random search with a shrinking region of interest.

3.1 Dynamic 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 w are chosen uniformly from a region of interest (ROI), set to the whole parameter space in the simplest implementation. Each chosen policy is individually compared, using a fixed number (N) of trials, with the best policy found so far, which is replaced accordingly. We also show that random search with a dynamic ROI can be made into an effective policy search algorithm using paired statistical tests with asymmetric confidence intervals to adapt N. The bounds of the dynamic ROI are shrunk towards the current best point at regular intervals. Each dimension of the ROI reduces by a factor of $\sqrt{2}$, and this happens 16 times during learning. Figure 1 illustrates how this process works. Successive ROI bounds are indicated by boxes, contours of the function to be optimized by ellipses, and locations at which the function is evaluated by crosses.

Operation of the method using paired statistical tests is straightforward. When using the changing scenarios approach, start states (or Pegasus scenarios) are changed each time the best point changes. This enables the benefits of changing scenarios to be achieved, without excessively increasing computational costs.

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 and 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. The algorithm has the property that the simplex adapts its size and shape according to the local geometry of the search space. For example, in very smooth regions, the simplex will expand (and take large steps), and in narrow valleys it will become elongated. Likas and Lagaris (1999) have used a similar algorithm for policy search.

The algorithm is normally described in terms of atomic operations (reflect, expand, contract outside, contract, super-contract), some of which require multiple function evaluations. These are essentially internal states; state transitions depend on comparisons between the evaluation of a proposed point and an existing vertex:

> *REFLECT*. The proposal is the reflection of the worst vertex through the centroid of the remainder. This is the most common operation, and allows the simplex to 'crawl' along smooth functions, in approximately the steepest direction. A function evaluation takes place at the proposal. If the result is better than the best existing vertex, an EXPAND operation is attempted. Otherwise, if the proposal is better than the second-worst vertex, then it is accepted and another REFLECT operation takes place. Otherwise the proposal is rejected and the state becomes CONTRACT.
>
> *EXPAND*. The proposal and its evaluation from REFLECT is stored, but not yet accepted. In an attempt to accelerate the optimization, a new proposal is generated, in the same direction, but twice as far as the first from the centroid of the remaining vertices. If the function evaluation at this vertex is greater than the stored proposal, then the expansion is accepted. Otherwise the stored proposal, a reflection, is accepted. The next operation is REFLECT.
>
> *CONTRACT*. This operation attempts to reduce the step size after a failed reflection. A new proposal is generated at the midpoint between the rejected proposal and the centroid calculated in the REFLECT operation. This proposal is accepted if it is better than the worst vertex, and the state becomes REFLECT. Otherwise the proposal is rejected and the state becomes CONTRACT INSIDE.
>
> *CONTRACT INSIDE*. Both proposals (REFLECT and CONTRACT) have failed, so an attempt is made to improve the worst vertex by generating a proposal on the same side of the centroid (of the remaining vertices), instead of the opposite side. The proposal is located at the midpoint between the worst vertex and that centroid. If the function evaluation is better than the worst vertex, then the proposal is accepted (replacing the worst vertex) and the state becomes REFLECT. Otherwise the state becomes SUPER CONTRACT.
>
> *SUPER CONTRACT*. No progress has been made from the REFLECT, CONTRACT and CONTRACT INSIDE operations (i.e. the worst vertex cannot be improved). Each vertex is now moved to the midpoint between it and the best vertex. The vertices are not re-sorted until all m function evaluations have taken place. This is an expensive operation in high-dimensional search spaces. The vertices are then re-sorted and the new state is REFLECT.

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 (Wright, 1996) and oriented restarts (Kelley, 1997). DSM can be implemented using only pairwise comparisons of policies, rather than reference to the numerical function value.

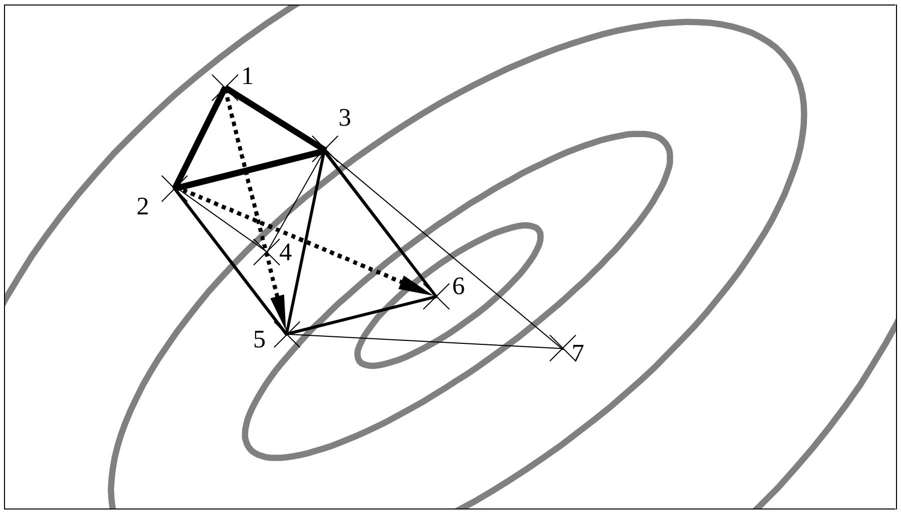


Figure 2: Operation of the downhill simplex method.

Figure 2 shows the operation of the method. The simplex has 3 vertices (1,2,3) for a 2-dimensional search space. The first proposal (4) is a reflection of vertex 1 (the worst) through the centroid of 2 and 3. The proposal is accepted and the simplex attempts to expand in the same direction (5). This expansion is accepted, and another reflection (of the new worst vertex) generates a proposal (6) which is also accepted, after a tentative expansion (7) fails.

In our implementation, the simplex is always initialized with one vertex at the origin, and the remaining vertices placed on each of the axes through the origin, at a displacement of $R_i/5$ where R_i is the size of the search space in dimension i. The DSM method can become stuck, or 'stagnate', particularly in high-dimensional optimization problems. This not only occurs when a local minimum of the objective function is found; it also happens when the simplex becomes too elongated. To reduce this problem, the simplex is restarted at regular intervals during learning (*e.g.* after every 256 function evaluations). At each restart, the best vertex (which we call the *anchor*) is retained, but a small, uniformly distributed "jitter" is added in each direction. All other vertices are placed so that the simplex has its original size and set of orthogonal displacement vectors. However, vertices are reflected through the anchor if necessary, so as to be on the same side of the anchor as the origin. (This has a regularization effect.) The jitter is always in a direction so as to ensure that the resulting simplex volume contains the best vertex from before the restart. The jitter in direction i is uniformly distributed in magnitude according to $U[0, R_i/80]$, but clipped as necessary to the bounds of the search space.

DSM is the only method we consider that makes (implicit) use of gradient information to select proposals. This has the advantage that a local optimum can be found very rapidly, and the disadvantage that its performance is sensitive to its starting position which largely determines whether a local or global optimum will be found. However, when optimizing a stochastic function, the unreliable decision-making resulting from empirical policy evaluations may actually *help* the simplex to move out of the "basin of attraction" of a local optimum, increasing the chances of finding a global optimum. Another way of interpreting the effect of noise is that it acts as a 'regularizer' - essentially causing the search space to be smoother or easier to traverse. An obvious practical approach for exploiting this property would be to slowly decrease stochasticity during learning by increasing the number of trials used for each policy evaluation. If the noise is sufficiently large initially, and reduced at a sufficiently small rate, it might be expected that the DSM would find the global optimum. However, unlike some Markov Chain Monte Carlo sampling/optimization methods, there are no theoretical guarantees that the stochastic behavior of the simplex will allow it to find, with high probability, a global optimum of the function.

Using a fixed set of scenarios with the DSM has no affect on the implementation of the algorithm. However, there is a subtle aspect to the implementation which greatly affects performance for optimizing stochastic functions. Re-evaluating all the vertices (including the jittered best vertex) at each restart allows the function values stored at the simplex vertices to decrease. Therefore any optimistic values resulting from noise in the function (*i.e.* stochastic empirical returns) is likely to decrease to realistic values.

When using DSM with changing scenarios, the change can be made whenever the DSM performs a SUPER-CONTRACT or reset. At these times, most of the vertices would be moved and require re-evaluation anyway, so the number of additional simulation trials is minimized. When using the Wilcoxon test for policy comparison, there is no guarantee that a total ordering will exist on the simplex vertices.[1] (i.e if vertex *A* beats vertex *B*, and *B* beats *C*, there is no guarantee that *A* beats *C* using Wilcoxon tests in each case.) Therefore a voting method was used in which every vertex was compared with every other vertex. The votes for each vertex were counted to place an ordering on them. To break ties, the empirical average return was used.

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 and Price, 1995). It has been shown to be effective for global optimization of reasonably complex functions, and is very simple to implement. DE's proposals are much more random than DSM, but more sophisticated than the dynamic random search method. DE keeps a larger population of points than DSM (at least $2m$, typically 64 in our experiments).

Initially, the population is chosen randomly, uniformly distributed within the search space. The process of generating a proposal makes use of at least four mutually exclusive population members. Firstly, one candidate in the population is chosen systematically (in some predefined order) for improvement. Then the vector difference between two randomly chosen members, weighted by a scalar parameter, F, is added to a third randomly chosen 'parent'. Some implementations add two or more such vector differences. Crossover (see below) takes place between this and the candidate, to obtain a proposal point. A function evaluation takes place at the proposal, and a test is then performed for whether this is better than the candidate. According to the outcome *either* the proposal replaces the candidate *or* the population remains unchanged. In either case, a new candidate for replacement will be selected on the next iteration. Crossover is implemented here by selecting each element of the proposal vector from either the candidate or the new vector according to some pre-specified probability ρ. For each proposal, F was chosen stochastically from the set $\{1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8\}$ with probabilities $\{81, 432, 972, 1200, 886, 400, 108, 16, 1\}$ as fractions of 4096. Proposals outside the bounds of the search space were ignored, allowing a valid proposal to be obtained for each candidate.

1. Total ordering would be guaranteed only if each vertex had been evaluated on the same set of scenarios.

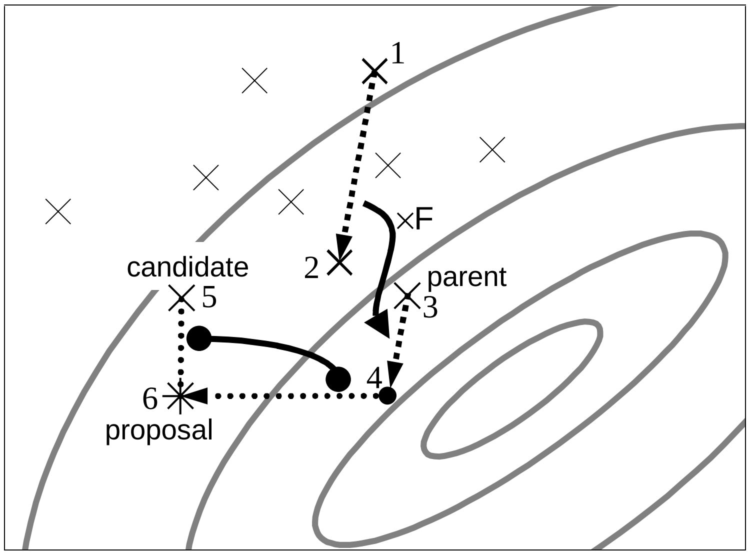


Figure 3: Obtaining a new proposal in differential evolution.

Figure 3 illustrates this process in more detail: the weighted difference between two population members (1,2) is added to a third population point (3). The result (4) is subject to crossover with the candidate for replacement (5) to obtain a proposal (6). The proposal is evaluated (using a number of trials) and replaces the candidate if it is found to be better. Note that the proposal could be identical to (4) or (5), depending on the outcome of crossover.

The role of the vector addition is to provide proposals that are within, or near the boundaries of, the existing population. The *F* parameter affects the rate at which the variance of the population is likely to decrease. Crossover prevents the population from become trapped in a subspace and may exploit the relative independence between different directions in the policy space. This latter property is particularly important for finding high-dimensional policies that have one parameter for each state, or map *localized* state features to actions: crossover would allow two policies which each perform well in particular regions of state space to be combined into one which works well in both regions.

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 with probability only slightly better than chance (0.5) in large populations. This means that it should be possible to use many fewer trials (or even $N = 1$) for each comparison. It is again possible to use paired statistical tests to make the individual comparisons more reliable.

When implementing differential evolution with paired comparisons and changing scenarios, the simplest approach (used in our evaluations) is to re-evaluate every candidate with a new set of scenarios before evaluating each proposal with the same new scenarios. This doubles the number of trials performed for each sweep of the population, but ensures that the bias associated with using scenarios is independent for each decision.

4. Evaluation Tasks

Two tasks, of different complexity, are described. These have very different properties; the mountain car task has a small, fully observable state space and is near-deterministic. The pursuer-evader task is partially observable, requires multi-agent learning, and contains a ‘hostile’ agent (the evader) with a stochastic policy and hidden internal state.

4.1 Multi-Pursuer Evader Task Description

The main application problem on which the methods will be evaluated requires two pursuers to co-operate in order to catch a highly maneuverable reacting evader, in a two-dimensional plane. This yields a 12-dimensional policy search problem (6 dimensions for each pursuer). The evader uses a stochastic policy that is hand-designed to give effective evasion behavior.

4.1.1 SIMULATION

The pursuers start close to the origin, always with the same initial direction of motion (see Table 2). The initial location of the evader is chosen randomly,[2] between 400 and 800 meters from the origin. At each time step (0.002 seconds), the evader and pursuer positions are updated analytically, and each (simultaneously) chooses an angular velocity as an action. Pursuer angular velocities are limited to 2π rad s^{-1}, whereas the evader can turn at 6π rad s^{-1}. In combination with the effect of different speeds, this means that the ”turning circle” of the evader is much smaller. In order to allow faster simulation, an adaptive time scale was used. Each agent (pursuer or evader) was able to specify a number of primitive time steps for which its chosen action would be valid. This implementation detail did not affect behavior of the system and so will not be described in detail here.

Each trial ends when 3072 steps (approximately 6 seconds) have elapsed (failure), or one of the pursuers has come within one “miss distance” (D meters) of the evader (success). In our experiments, $D = 2$ throughout. We assume that the pursuers have perfect information about the position of the evader and *vice versa*. Latency in the measurement process is ignored, and it is also assumed that velocities (and derived quantities) can be measured instantaneously. The goal is to find optimal policies for the pursuers, in order to maximize expected return.

The evader stochastically switches between 3 internal states,[3] based on the distance and approach speed of the closest pursuer. Representing position by the co-ordinate system (x,y) and velocity by (u,v) we use subscript e for the evader and p for the closest pursuer. The evader’s rate of turn, ω rad s^{-1}, is calculated (according to its internal state) as follows:

CONTINUE. The evader flies away from the origin.
$\omega = 2\pi \operatorname{sign}(y_e u_e - x_e v_e)$

EVADE. The evader flies away from the closest pursuer.
$\omega = 2\pi \operatorname{sign}((y_e - y_p)u_e - (x_e - x_p)v_e)$

MANEUVER. The evader turns sharply to throw the closest pursuer off course.
$\omega = 6\pi \operatorname{sign}((x_e - x_p)(v_e - v_p) - (y_e - y_p)(u_e - u_p))$

2. For x_i, s_i, z_e and s_e let $U[A,B]$ return a natural number between A and B (inclusive). For angles, the usual meaning of $U[\,]$ is ascribed.
3. The pursuers are unable to observe the evader’s hidden state.

State	Symbol and initial distribution
First pursuer position	$x_1 \sim U[0,9], y_1 = -2.5$
Second pursuer position	$x_2 \sim U[0,9], y_2 = 2.5$
Pursuer *i* velocity: (motion direction and magnitude are chosen from uniform distributions)	$u_i = s_i \sin\theta_i$, $v_i = s_i \cos\theta_i$ where $\theta_i \sim U[-\frac{\pi}{2}, \frac{\pi}{2}]$ and $s_i \sim U[475, 525]$
Evader position: (direction and distance from origin are chosen from uniform distributions)	$x_e = z_e \cos\theta_e$, $y_e = z_e \sin\theta_e$ where $\theta_e \sim U[-\frac{\pi}{2}, \frac{\pi}{2}]$ and $z_e \sim U[400, 800]$
Evader velocity: (motion direction and magnitude are chosen from uniform distributions)	$u_e = s_e \sin\theta_e$, $v_e = s_e \cos\theta_e$ where $\theta_e \sim U[-\pi, \pi]$ and $s_e \sim U[225, 275]$

Table 2: States and initial distributions for the pursuer-evader task.

The sign function returns ± 1 according to the sign of its argument. State transitions are determined according to the following rules, using the range (z_{min}) and approach rate ($-\dot{z}_{min}$) of the closest pursuer at the current time:

From EVADE, if $z_{min} \geq 400$ the state becomes CONTINUE.

From CONTINUE, if $z_{min} < 100$ the state becomes EVADE.

If $400 \geq z_{min} \geq 100$ the state switches stochastically between CONTINUE and EVADE with (symmetric) transition probability $1/32$ at each time step.

From EVADE, if $12z < -\dot{z}_{min}$ the state becomes MANEUVER.

From MANEUVER, if $\dot{z}_{min} > 0$ the state becomes EVADE.

4.1.2 RETURNS

The return from a simulation trial is given by $-(z_{end} - D)/(z_{end} + D)$ where z_{end} is the distance of the closest pursuer from the evader at the end of the trial, but a bonus of 1.0 is received for success ($z_{end} = D$). Hence the return is in the range $[-1, +1]$. In trials ending with failure, the return will always be less than zero; hence a positive empirical average return of R (for some policy) implies that the proportion of trials ending in success is at least R.

4.1.3 PURSUER POLICY PARAMETERIZATION

The two pursuers must co-operate to defeat the evader by learning a strategy that hedges against possible maneuvers by taking different paths. The observed state for each pursuer (indexed by i) is a vector of 5 features, derived from measurements:

a. Relative angle of the target: $\phi/(2\pi)$

b. A function of evader range: $\exp(-z_i/100)$

c. A function of sight-line rate: $\tanh(\dot{\phi}/(2\pi))$

d. Range rate: $\dot{z}_i/500$

e. Relative range of pursuers: $(z_i - \overline{z})/(z_i + \overline{z} + D)$

Only (e) conveys information about one pursuer to the other, through the mean pursuer distance[4] $\overline{z}$. We construct a 6-element vector f from these features and a constant (equal to 1) in the sixth element. The angular velocity for a pursuer is then determined as $\omega = 2\pi \tanh(2\alpha^T f)$ in parameters α. This policy deterministically maps the observed features through a linear function, then a sigmoid squashing function (tanh), to yield an action for the pursuer. Concatenating the values of α for the two pursuers yields the target for optimization, a 12-element vector w. Each element of the policy parameter vector w is constrained to the range $[-10, +10]$.

4.1.4 EXAMPLE

Figure 4(a) shows the paths taken by pursuers and evader in a simulation trial during early learning. A 100 meter grid is shown at the origin, where the two pursuers (dashed lines) start. The evader (solid line) starts at about 600m from the origin. In this example, one pursuer has learnt useful behavior, while the other flies in a circle for the whole duration of the trial. The evader is seen to fly directly away from the origin for the first part of the trial (the CONTINUE state). When the pursuer is within 400m range, the evader starts to switch stochastically to and from the EVADE state (flying away from the pursuer). This is seen as a series of perceptible changes of direction. At 100m range the evader remains in the EVADE state, until the criterion is met for it to enter the MANEUVER state (when a sharp turn is apparent). The pursuer cannot turn so fast and so it continues on where the evader has turned. When the pursuer is seen to be receding (negative approach speed) the evader switches back into the EVADE state. Soon after, the trial ends because no time remains, and a negative return is received by the pursuers. If the trial were to continue further, the evader would continue to be successful in out-maneuvering the single pursuer that is following it.

Figure 4(b) shows a successful trial after learning. Both pursuers take an active part in defeating the evader, cutting off escape routes to either side by approaching the pursuer simultaneously from very different directions. The trial ends when the evader is less than 2 meters from one of the pursuers. A return of 1 is received.

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 and 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, and then allowing further learning in the full joint policy space.

4.2 The Mountain Car Problem

We also tested our methods on the mountain car problem (Figure 5). 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.). The car's state consists of its one-dimensional position x and horizontal velocity $\dot{x}$. We used a formulation similar to that of Schoknecht (2001) which accounts properly for the centripetal force[5] acting on the car, in addition

4. The average distance at the current time of the pursuers from the evader.
5. Previous formulations of this problem have ignored the centripetal force, breaking the energy conservation law.

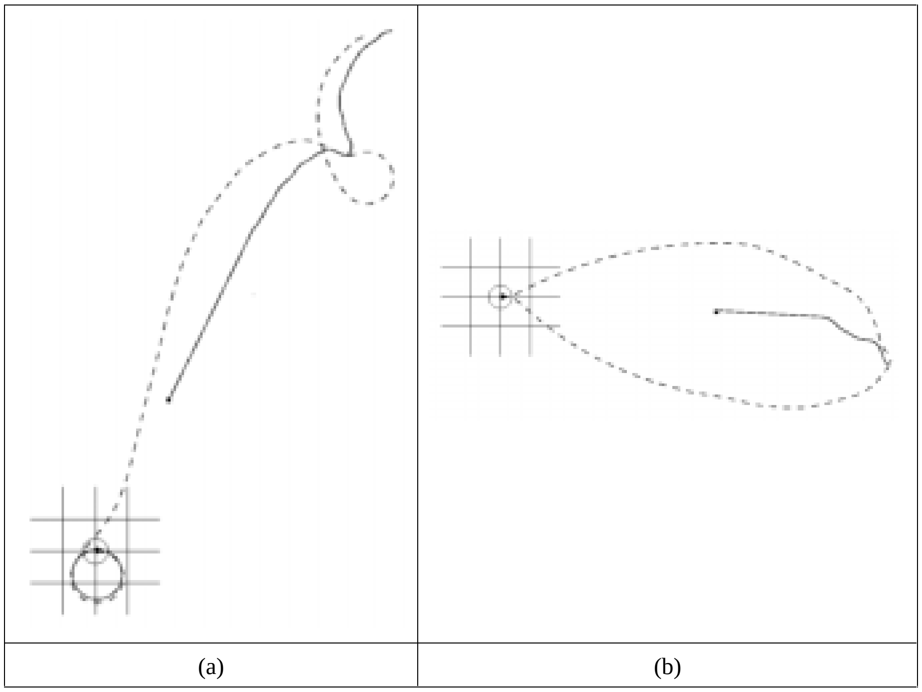


Figure 4: Pursuer-evader simulation trials.

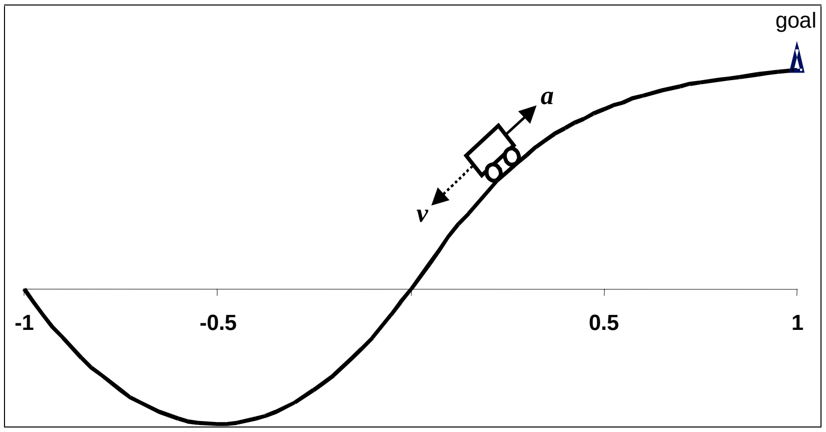


Figure 5: The mountain car problem.

to gravity and its own acceleration. A 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 and Moore, 1999). Uniform noise was also added to the acceleration representing the effect of wind turbulence.[6] The height of the hill is given by:

$$H(x) = \begin{cases} x < 0 & : \quad x(x+1) \\ x \geq 0 & : \quad x/\sqrt{1+5x^2} \end{cases}$$

4.2.1 SIMULATION

The dynamics are given by:

$$\ddot{x} = c(a + zc)/m - H'(g + \dot{x}^2 H'')c^2 \tag{1}$$

where H' and H'' are the first and second derivatives of H, with respect to x. The mass of the car, m, is chosen to be 1. The force of the wind, z, is uniformly distributed in the range $[-2, 2]$ Newtons at every time step. c is the cosine of the slope angle given by:

$$c = \frac{1}{\sqrt{1+(H')^2}}$$

Note that the first term in (1) represents acceleration due to external forces applied to the system. When this is zero, (1) becomes the only solution for conservation of energy in the presence of gravity. First order Runge Kutta integration (the midpoint method) was used to simulate the system, with a (total) time step of 0.025s. The main difference between the formulation used here and that of Schoknecht is that we have limited speed *parallel* to the slope to ± 3.0 ms^{-1}, and reduced the mountain car's maximum acceleration force from 4N to 3N. Also, an episodic formulation is used here, affecting the regime of returns.

Figure 6 shows the maximum effective acceleration *resolved in the horizontal direction* from each of the forces acting on the car.[7] Note that the *resolved* maximum wind and engine forces are not of constant magnitude because they depend on the slope; they can also reverse instantaneously. For a given horizontal position, the resolved centripetal acceleration always has the same sign; its magnitude depends on the speed of the car, and the slope *and* curvature of the hill. The resolved acceleration due to gravity varies only according to the slope of the hill.

4.2.2 RETURNS

The return received at the end of a trial is:

$$R = \begin{cases} -1 & \text{for exits to the left} \\ (x_{max} - 1)/2 & \text{for running out of time (proportional to maximum position reached)} \\ 1 - |\dot{x}_{end}|/3 & \text{for reaching the goal (dependent on terminal velocity } \dot{x}_{end}\text{)} \end{cases}$$

The maximum duration of a trial is 1024 time steps (25.6 seconds).

6. Adding too much noise to the acceleration can make the task less interesting; constant acceleration to the right will be a near-optimal policy under strong wind conditions.
7. The accelerations measured in ms^{-2} are plotted against the car's horizontal position (meters).

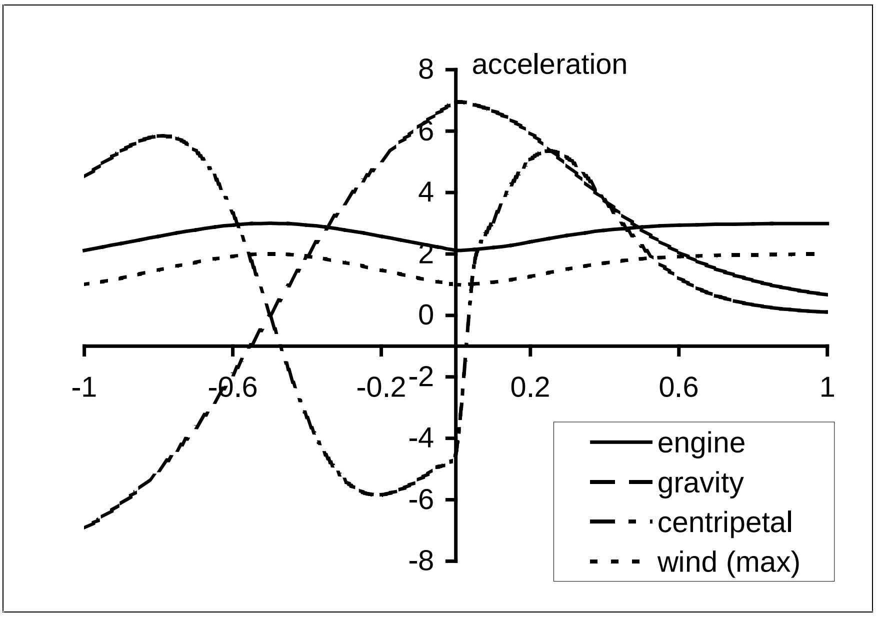


Figure 6: Accelerations acting on the mountain car.

4.3 Policy Parameterization

The deterministic policy, parameterized by w, is given by a thresholded polynomial in x and scaled velocity $u \equiv \dot{x}/3$:

$$a = 3\,\mathrm{sign}((1, x, x^2, x^3, ux, ux^2, ux^3, u^2) \times w)$$

where the sign function returns ± 1 according to the sign of its argument and $\times$ represents matrix multiplication. Each element of w is constrained to the range $[-1, +1]$.

4.3.1 EXAMPLE

Figure 7 shows the position and height of the mountain car during a successful run. The two graphs align with the hill profile (top-left); the time axis is horizontal for the car's vertical position and *vice versa*. The horizontal and vertical position of the car go through six oscillatory cycles, as it builds up energy. The increase in height each time is unreliable due to the additive noise (wind turbulence). When the height of the car is 0.2 meters, the slope of the hill is small enough that it can accelerate to the goal. In the remaining period, the car decelerates to stop exactly at the goal. Note that the horizontal position never falls below -1; in that case the trial would end immediately with a negative return.

5. Experimental Comparison

A large number of experiments have been performed to gain an insight into the effectiveness of paired statistical tests and to evaluate which policy search algorithms are most practical. This evaluation uses both test problems (mountain car and pursuer-evader) and three optimization procedures (dynamic random search, downhill simplex method, differential evolution).

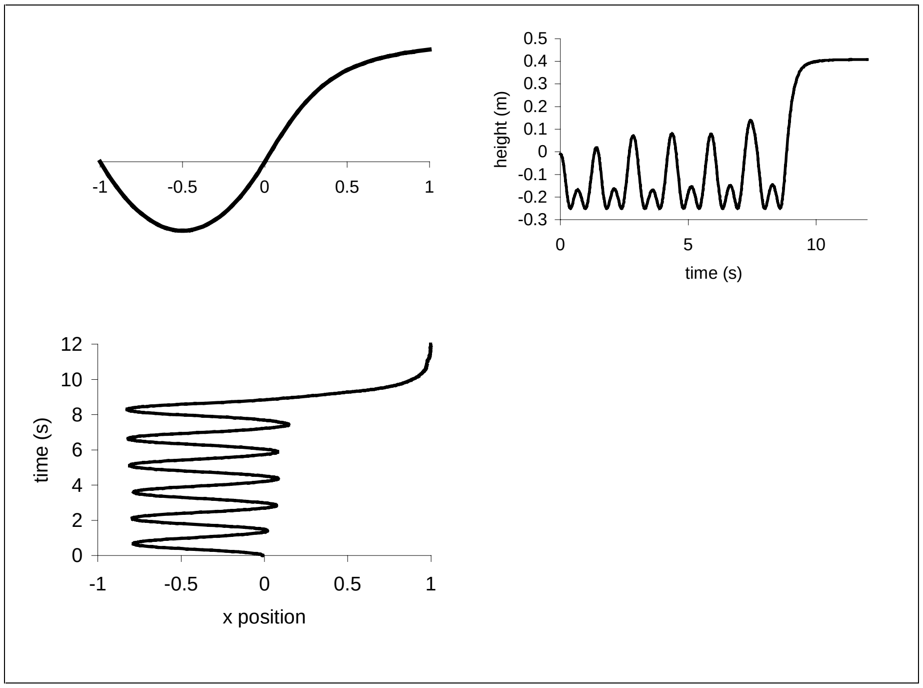


Figure 7: Position and height of the mountain car during a successful trial.

We will first evaluate whether there is a benefit to using pairing of scenarios and of random number seeds (as in Pegasus). Then the level of overfitting (bias towards training scenarios) will be quantified for Pegasus, in order to identify a major difficulty with paired methods. Proposed methods for ameliorating this overfitting problem will then be evaluated. These include (i) using a nonparametric statistical test for policy comparisons; (ii) changing Pegasus scenarios during learning; and (iii) exploiting the population averaging effect of differential evolution. Finally, methods that make use of confidence intervals in the paired tests, to adapt the number of trials per policy, will be evaluated.

In all experiments, an independent test set (size 512) was used to evaluate the policies learnt by each of the methods. Each test example was a Pegasus-style scenario (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.

Test performance is measured using the policy currently believed best by the learning algorithm, except for differential evolution. In measuring the test performance of differential evolution, it is never known which is the best member of the population (because policy evaluations are assumed to be stochastic). To obtain a reasonable measure of test performance for DE we evaluate the *most recently replaced* member of the population, but maximize this quantity over all earlier evaluations within the same learning run. This makes the test performance broadly comparable with the other methods, but in strict terms the maximization process should make use of a validation set, independent of the test set, and the validation runs should be included in the total for learning. Alternatively, tournament selection could be used to predict the best example in the population.

On-line returns (during learning) in the subsequent results will usually be lower than test performance because they include the cost of exploration trials, except where stated. The 'cost of exploration' is particularly high in dynamic random search, where most proposed policies are rejected after being evaluated. Therefore we expect on-line performance to be worst for that method, and best for DSM in which proposals are highly localized. It is possible to measure on-line performance for only the proposals that are successful ("ON-LINE MAX"). This measure can be higher than test performance because the current best policy may be "tuned" to perform well for the scenarios encountered during learning, rather than the full test set. Therefore ON-LINE MAX will be a useful measure for investigating overfitting.

21 runs were performed for each experiment, except where more (41) were required to obtain tighter error bounds. Standard errors were calculated for the mean performance measures (on-line and test). The standard error is indicative of the *typical error* if the underlying populations are Gaussian. When comparing two measures with about the same standard error s a difference of about $2\sqrt{2}s$ indicates confidence at 97.5% (one-tailed) for rejecting a null hypothesis that the two measures are equal. If the two standard errors are very different, then a difference of $2s$, where s is the larger, gives the same level of confidence.[8] Median measures are useful in experiments where some of the learning runs do not converge and hence the returns are very non-Gaussian. If most ($>$75%) of learning runs converge, then the median is a good indicator of typical performance in the runs which do converge. However, while the median is a useful indicator, error bounds are not easily calculated. Therefore, in some cases, we also measure the mean test performance of those learning runs that converged; convergence is indicated be exceedence of some threshold for *on-line* performance.

Full summarized experimental results and learning algorithm parameter settings are given in Appendix A; the most important and apparent trends are discussed here.

5.1 Evaluation of the Benefits of Pairing

The aim of the first experiment is to evaluate the two levels of pairing when comparing policies with a small, fixed, number of cases. Each *case* in the paired methods is either a simple scenario (start state) or Pegasus scenario (start state and random seed). All three optimization procedures were used on both tasks. The results from the control experiments (using simple random search) show that both tasks are non-trivial. Even after 65536 trials, mean test performance was 0.6 ($N = 8$) for the mountain car, and -0.36 ($N = 8$) for the pursuer evader problem. Good performance would be 0.9 and 0.8 respectively.

Figure 8 compares performance of the unpaired and the simplest paired method ('scenarios') which fixes the N (8) trial start states for the whole duration of learning. The latter is a paired method because policies are compared using identical sets of start states. Policy evaluations are stochastic (in both methods) because random number sequences are not controlled by the learner. For dynamic random search, there was no clear difference between the two methods. Neither method performed well for the pursuer evader task. Good performance was obtained after 65536 trials in the mountain car task, and the improvement for using pairing was significant (0.84 ± 0.04 compared with 0.68 ± 0.07). This suggests that using unpaired comparisons in dynamic random search is less reliable for optimizing the true objective (measured by test set performance).

8. These statements are valid (using a t test) for our smallest sample size (21).

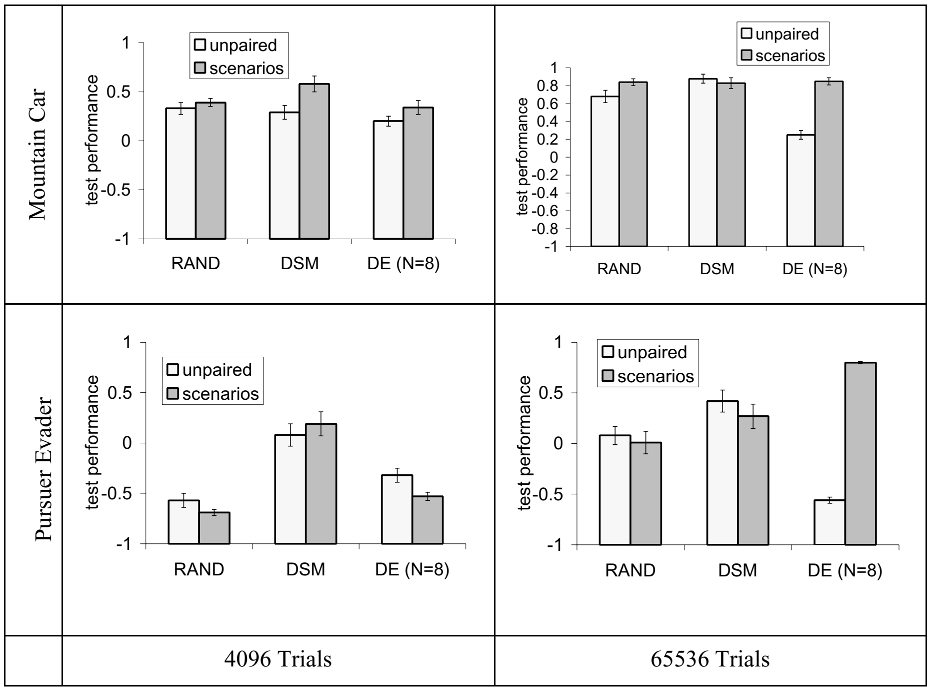
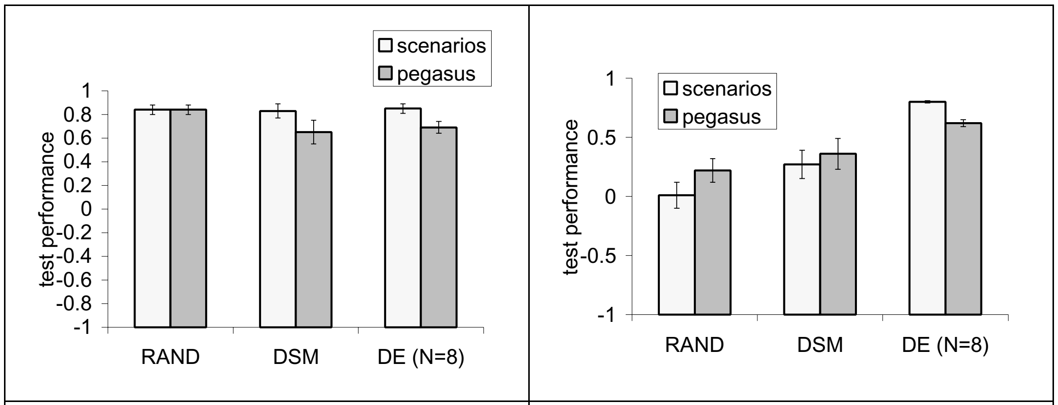


Figure 8: Evaluation of paired and unpaired policy comparison approaches.

The DSM optimization method showed a different trend. In both tasks, the paired method performed better after 4096 trials but was no better (and probably worse) by 65536 trials. This suggests that more reliable decisions are made early on but this is outweighed later by ‘overfitting’ to the set of scenarios. Differential evolution showed the greatest benefits from the paired comparisons. After 65536 trials, DE with fixed scenarios found good solutions for both tasks, but failed badly without. (The improvement was significant for both tasks.) In summary, there was evidence that paired comparisons improved performance of 2 out of 3 search methods. Note that the mountain car problem seems readily solvable with $N = 8$ whereas the pursuer evader problem is more challenging.

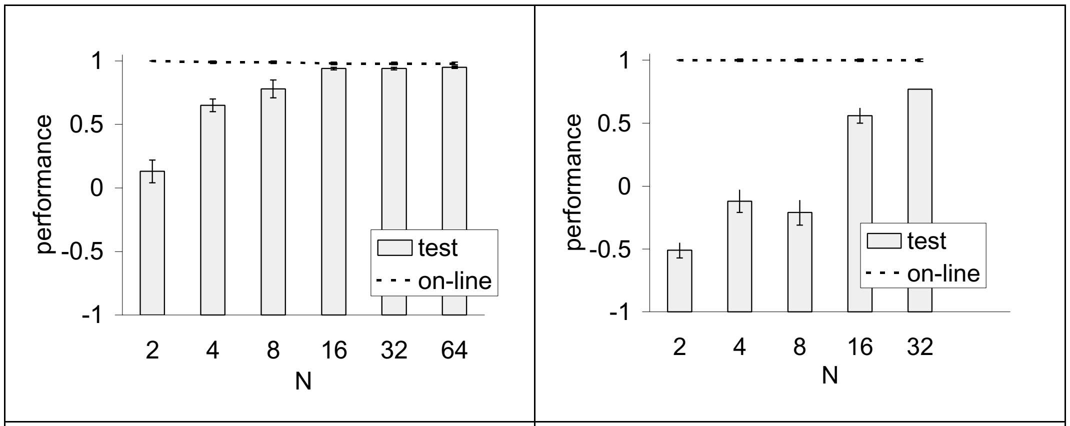
Figure 9 shows a comparison between the ‘scenarios’ method and Pegasus for each of the tasks. Recall that Pegasus fixes start states and random number seeds, making the result of each policy evaluation deterministic. It could be expected that Pegasus would perform better, because it makes use of this stronger form of pairing in the policy comparisons. However, there is evidence for a reduction in performance. The decrease is significant for DE in both tasks. There are two obvious hypotheses that could explain this result. Either (i) Pegasus reduces the chance of convergence to a good solution, or (ii) the converged Pegasus solution is poorer at generalizing to unseen test data.

It is reasonable to suspect that both hypotheses will be true to some extent. In support of hypothesis (i), recall that stochastic function evaluations can reduce the risk that an optimization method will become stuck in a local minimum, and that the deterministic Pegasus objective function will be less smooth than the true objective function and may have discontinuities. In support of (ii), note that the Pegasus learning method can exploit the particular random number sequences specified in the set of N scenarios, rather than finding policies that work well for true stochastic behavior. The



Mountain Car 65536 Trials	Pursuer Evader 65536 Trials

Figure 9: Comparison between fixed scenario method and Pegasus.



Mountain Car 65536 Trials	Pursuer Evader 65536 Trials

Figure 10: Overfitting in Pegasus.

full results show no evidence to support (i): on-line performance for Pegasus is usually better than for fixed scenarios. The next experiment will attempt to find evidence to support hypothesis (ii).

A comparison is now made between different values of N (the number of simulation runs for each policy evaluation), when using the dynamic random search method and fixed Pegasus scenarios. Figure 10 shows the mean test performance (at 65536 trials) for learning runs in which on-line performance was greater than 0.97; i.e. for those runs in which learning was convergent. It is apparent that test performance increases with N. The difference between on-line performance and test performance is a measure of overfitting (*i.e.* the degree to which the policy has become overspecialized to the particular Pegasus scenarios used). Overfitting is apparent for $N < 16$ with the mountain car problem, and for all N with the pursuer evader problem. (It was not possible to obtain a standard error for $N = 32$, or a measurement at $N = 64$, in the pursuer evader problem, because not enough learning runs had converged by 65536 trials.) *From these results it is apparent that pairing of random number sequences in Pegasus can cause excessive overfitting and hence poor generalization performance.*

The problem may be most severe in the pursuer evader problem because there are more policy parameters (12) to be fitted, providing more opportunities for the learner to exploit specific random number sequences. Alternatively, it may be a result of the way random numbers are used in the pursuer evader simulation: a slight change in policy can affect the way the sequence is interpreted (for controlling evader behavior). This is in contrast to the mountain car problem where the random numbers are used only to determine the additive noise (the wind) at each time step, which is unaffected by changes in policy.

5.2 Ameliorating Overfitting

Note that Pegasus exacerbates an overfitting problem that was already present when using fixed scenarios, and so both approaches can be improved if a means can be found to eliminate the overfitting. Two such possibilities are investigated here: an alternative paired statistical test, and changing the scenarios during learning.

5.2.1 NON-PARAMETRIC POLICY COMPARISON

The statistic used in policy comparisons with Pegasus is the empirical average return over the N trials used for each policy evaluation. Note that a policy change which improves performance on one scenario at the cost of performance in others will be accepted under this criterion if the average increases. However, in many application problems we expect the optimal policy to be *optimal for all scenarios*. This is certainly true in fully observable environments if the policy is expressive of all possible behaviors. In this case, a path to the optimal policy may be found by accepting policy changes which improve the *median* of returns over the population of possible scenarios. Maximizing the median is sufficient to yield an optimal policy in the fully-observable situation mentioned above, and may be useful in many other settings. The Wilcoxon signed rank test is used here to test for an increase in the population median on each policy comparison. (Note that this is not the same as comparing *sample* medians.)

In the mountain car task, which is fully observable, generalization performance was increased for all three methods at 4096 and 65536 trials. The average improvement at 65536 trials was 0.06 ± 0.03, and was attributable to a decrease in overfitting, because the *on-line* performance was not changed. The improvement as a whole was significant, but not for any one of the 6 individual comparisons. On the pursuer evader task, there was no evidence for a difference between the two types of policy comparison. In summary, using a non-parametric test in policy comparisons can reduce overfitting.

5.2.2 CHANGING SCENARIOS DURING LEARNING

In order to remove the bias associated with any particular set of scenarios, the Pegasus scenarios can be changed during learning. This works differently for each of the search methods. In dynamic random search, scenarios are changed whenever a new winner is found. In DSM, scenarios are

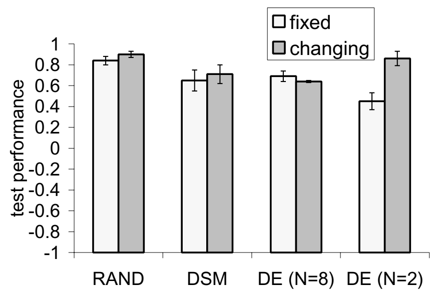
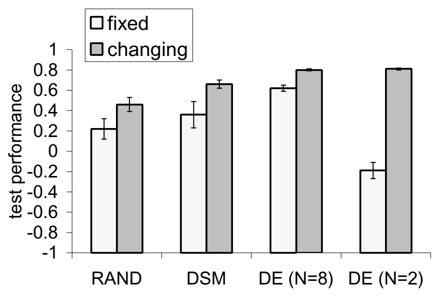
	
Mountain Car 65536 Trials	Pursuer Evader 65536 Trials

Figure 11: The benefit of changing Pegasus scenarios during learning.

changed on SUPER CONTRACT operations and restarts. In DE a new set of scenarios is used for every policy comparison. Figure 11 compares the standard Pegasus approach (with a fixed set of scenarios throughout each learning run) with the 'changing scenarios' variants. In the mountain car task, there was no significant benefit with $N = 8$. For the pursuer-evader task, all three optimization methods showed an improvement in performance from changing scenarios, and this was significant for DSM ($.36 \pm .13$ improved to $.64 \pm .04$) and DE ($.62 \pm .03$ improved to $.80 \pm .01$). This latter result was the first experiment in which useful co-operative behavior was obtained in the pursuer-evader task.

Given that changing the scenarios had eliminated overfitting for $N = 8$, without any loss of convergence, the experiments were repeated for DE with $N = 2$. Of the 3 optimization methods, only DE has the potential to robustly exploit a large number of unreliable policy comparisons, because there will be multiple copies of effective sub-policies[9] within the population at any one time, so replacing any individual member of the population does not significantly affect fitness of the population as a whole. Using $N = 2$ cannot be expected to yield a policy that maximizes expected return; instead it will find a policy that is *most likely*, over 2 scenarios, to perform better than other policies encountered during learning. The expectation is that the latter will often be nearly as good a policy as the former. The benefit of using $N = 2$ is also apparent: four times as many policy comparisons can be performed in the budget of 65536 trials. The results in Figure 11 show that this method does indeed work well, but only if the 'changing scenarios' approach is used. Without this, the same two Pegasus scenarios are used throughout the learning run leading to severe overfitting, as expected.

The success of DE with $N = 2$ is very encouraging because it shows that a large value of N is not necessarily important for finding near-optimal policies. The method successfully eliminates noise (variance) by using Pegasus-style policy comparisons, and avoids long-term bias by choosing new scenarios for every comparison. N could be increased during learning to ensure that the final solution is a local maximum in terms of expected return (the true objective).

9. A sub-policy assigns values to only a subset of the elements in the policy parameter vector.

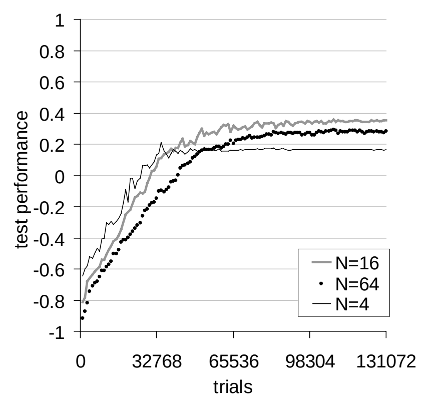
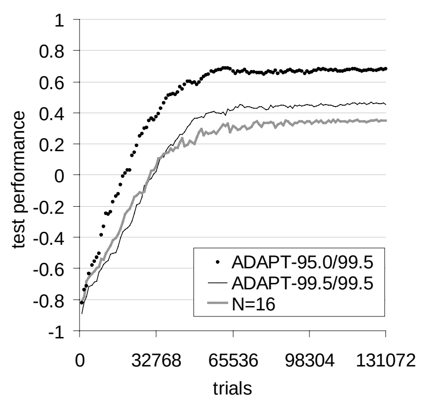
	
Fixed *N*	Adaptive *N* vs Fixed *N*

Figure 12: Adapting *N* in dynamic random search (pursuer-evader).

5.3 Adapting the Number of Scenarios

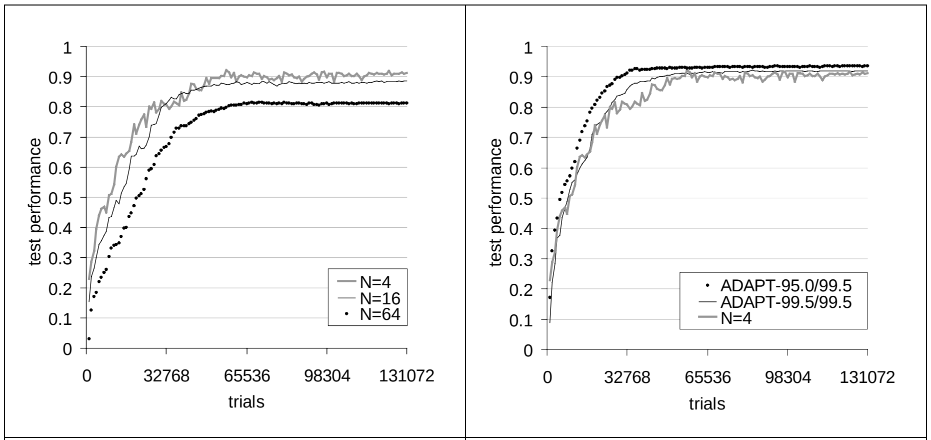
Identifying the fact that we are performing statistical tests on samples (of size *N*) rather than just comparing an average is potentially very useful: variance and other information about the population distribution can be obtained (Strens and Moore, 2001). In particular, the level of significance can be calculated for each policy comparison.

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, to achieve a given level of confidence in each decision. The dynamic 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 95% (one-tailed) significance for rejecting a (worse) proposal, but 99.5% (one-tailed) significance for accepting a (better) proposal (subject to $4 < N < 64$). This is 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 12 compares the adaptive method against fixed-*N* (4, 16 or 64 changing Pegasus scenarios) for the pursuer evader task. Fixed $N = 4$ learns fastest initially, but its final test performance is poor because dynamic random search does not converge to a good solution with the unreliable decision-making. Fixing $N = 64$ learns more slowly, and would eventually obtain the best test performance if it were not for the shrinking region of interest used to obtain convergence within the budget of 131072 trials. Fixed $N = 16$ offers the best compromise, achieving the best final test performance for the three fixed-*N* experiments.

The adaptive method with asymmetric confidence intervals (CIs) learns as fast as $N = 16$, but obtains a much higher final test performance (0.68 ± 0.05 compared with 0.35 ± 0.07). Error bars are not shown in the figure for clarity, but the improvement was significant from 16384 trials onward. Using asymmetric CIs was also significantly better than symmetric CIs from 8192 trials onward.

Results for the mountain car task were similar (Figure 13), except $N = 4$ was the best of the fixed methods, and differences between the methods were only significant at 16384 trials. From



Fixed *N* | Adaptive *N* vs Fixed *N*

Figure 13: Adapting *N* in dynamic random search (mountain car).

2048 trials onward, the asymmetric CI method has higher test performance than all other methods, suggesting that the benefits of adaptation are having an effect very early in learning. These results show that *confidence information available from paired statistical tests can be exploited to adapt the number of trials needed for each policy evaluation* and hence obtain better performance in a given budget of trials.

Schmidhuber and Zhao (1998) proposed a ‘success-story algorithm’ for direct policy search in which each policy change is evaluated indefinitely, or until the change is ‘rolled-back’ when there is insufficient evidence that it has accelerated returns. The algorithm relies upon the choice of a ‘trial length’ parameter, and might also be improved by estimating variance (or confidence) information before making decisions, allowing the trial length to be adapted.

6. Conclusions and Future Work

We have shown that paired statistical tests provide a significant advantage over naive comparison of policies. Although Pegasus is based on pairing of both start states and random number seeds, there is a risk of overfitting which means that the method should be avoided when using a fixed set of scenarios. Using the Wilcoxon signed rank sum test to compare policies reduced this overfitting, but not consistently over both tasks. The simplest and best approach for eliminating overfitting was to change the scenarios used for the paired policy comparisons during learning. This improved the performance for all search methods and for both tasks. With changing scenarios it was possible to obtain good performance on a 12-dimensional policy search problem with hidden state (pursuer-evader) with only 8 trials per policy.

By exploiting confidence information available from paired statistical tests, we also showed that the number of trials for each policy could be adapted to speed-up learning. The appropriate confidence interval depends on the nature of the search algorithm, but can be asymmetric. For dynamic random search, the asymmetric confidence interval ensured that very high confidence was needed before the best policy was replaced by a new proposal, but rejection decisions were made on much less evidence.

Most importantly, we showed that paired statistical tests were very effective at improving the performance of *stochastic* optimization methods such as dynamic random search and DE. 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.

Although we have shown that paired methods lead to very effective search procedures, there are several outstanding issues:

(i) The best paired methods (those with changing scenarios) can only be used in procedural pairwise search. This restricts the designer's choice of optimization method.

(ii) Paired methods do not exploit all the structure in the learning problem. In particular, while they can take account of the start state, they do not pay attention to the state trajectory observed during each trial. It is reasonable to expect that the state trajectory contains useful information that could be exploited by the learner.

(iii) Paired methods do not generalize across policy space (except where the optimization procedure does this) or across state space. Exploiting the smoothness of returns in each should provide useful information for the learner. Furthermore, extrapolation over the policy space could help the learner in its experiment design.

Boyan and Moore (2000) used learnt evaluation functions to predict the performance of a local search procedure as a function of starting state, enabling predictions about the likely payoff for starting the search at different locations. Similarly, *baseline* functions can be used in reinforcement learning to reduce variance of a stochastic policy gradient (Sutton et al., 2000). It is possible that similar functions, estimated over the joint space of states and policy parameters, could be exploited within direct policy search to give faster learning.

Acknowledgments

We would like to acknowledge support for this project from the UK Ministry of Defence Corporate Research Programme.

Appendix A. Experimental results and parameter settings

Category	**Values taken and description**
Performance statistic	**Mean** The mean of a performance measure over a number of runs (21 or 41), together with the *standard error* interval which defines typical errors on the estimate, assuming the population is Normally distributed. **Median** The median of a performance measure. This is a very useful indicator of performance if some trials do not converge, but has no confidence interval.
Performance measure	**On-line** The actual returns received by the agent while learning. This is normally lower than test performance because many learning trials are used for exploration. **On-line (max)** The performance estimate that the optimization procedure has stored with its best policy example (e.g. the best simplex vertex in DSM). **Test** The returns received by the agent's 'current' best policy using a standard test set of 512 scenarios (start states and random number seeds). For DE, the best population member is not known. Instead, the best test performance found (so far) during learning with DE is given. This may be greater from the current test performance because the agent may have become worse.
Comparison method	**Unpaired** Policy evaluations use N random starting states. **Scenarios** Policy evaluations use identical sets of start states. **Pegasus** Policy comparisons use identical sets of start states and of random seeds. **Pegasus(WX)** The Wilcoxon statistic is used for making policy comparisons.
Number of trials	$\{1024,\ldots,131072\}$ The number of trials performed during a single learning run.
Trials per policy N	$\{2,4,\ldots,64\}$ The number of trials used to evaluate each policy.

Table 3: Information for interpreting the results tables.

Context	**Parameters**
General	Repeats: 21 Trials per policy (N): 8 Total number of trials: 65536; except adaptation experiment (131072) Test set size: 512 Threshold for determining convergence of on-line (max) performance in overfitting experiment: 0.97
Downhill Simplex Method	Interval between resets: 128 policy evaluations (mountain car); 256 (pursuer evader).
Differential Evolution	Population size: 64 F: stochastic (see section 3.3) Crossover (ρ): 0.975 (mountain car); 0.5 (pursuer evader)
Random Search	Lower limit on number of trials per policy: 4 Upper limit on number of trials per policy: 64

Table 4: Default parameter settings.

	4096 Trials				**65536 Trials**			
	On-line	**Test**	**On-line median**	**Test median**	**On-line**	**Test**	**On-line median**	**Test median**
Control (N=8)	.51±.03	.19±.06	0.52	0.15	.83±.02	.60±.08	.85	.78
Control (N=64)	.11±.02	.06±.02	0.11	0.09	.45+.03	.42±.03	.42	.40
RANDOM								
Unpaired	.57±.03	.33±.06	.52	.26	.94±.03	.68±.07	.99	.91
Scenarios	.49±.03	.39±.04	.50	.39	.96±.02	.84±.04	.99	.93
Pegasus	.54±.02	.34±.06	.56	.37	.95±.03	.84±.04	.99	.92
Pegasus(WX)	.56±.05	.46±.06	.59	.43	.97±.01	.87±.03	.99	.94
DSM								
Unpaired	.65±.04	.29±.07	.56	.13	.95±.03	.88±.05	.99	.95
Scenarios	.76±.05	.58±.08	.90	.72	.92±.03	.83±.06	.99	.94
Pegasus	.67±.07	.56±.08	.81	.58	.80±.07	.65±.10	.99	.92
Pegasus(WX)	.66±.08	.59±.07	.75	.53	.78±.08	.72±.09	.99	.94
DE								
Unpaired	-.15±.02	.20±.05	-.15	.12	.00±.03	.25±.05	-.07	.14
Scenarios	-.11±.02	.34±.07	-.07	.46	.69±.03	.85±.04	.76	.98
Pegasus	-.03±.03	.31±.04	-.04	.39	.77±.03	.69±.05	.80	.93
Pegasus(WX)	-.02±.03	.36±.05	-.02	.38	.80±.03	.78±.05	.86	.94

Table 5: Comparison between unpaired and paired methods on the mountain car task.
(All on-line figures for Random Search and DSM are calculated using the *on-line max* criterion.)

	4096 Trials				**65536 Trials**			
	On-line	**Test**	**On-line median**	**Test median**	**On-line**	**Test**	**On-line median**	**Test median**
Control (N=8)	-.99±.00	-.57±.04	-.99	-.58	-.99±.00	-.36±.07	-.99	-.30
Control (N=64)	-.99±.00	-.76±.05	-.99	-.83	-.99±.00	-.46±.05	-.99	-.50
RANDOM								
Unpaired	-.33±.05	-.57±.07	-.38	-.67	.87±.04	.08±.09	1.0	.15
Scenarios	-.42±.04	-.69±.03	-.46	-.70	.84±.04	.01±.11	1.0	-.08
Pegasus	-.29+.06	-.49±.06	-.32	-.58	.91±.04	.22±.10	1.0	.35
Pegasus(WX)	-.22±.07	-.49±.08	-.28	-.58	.85±.06	.19±.10	1.0	.36
DSM								
Unpaired	.09±.11	.08±.11	.25	.21	.50±.05	.42±.11	.57	.56
Scenarios	.35±.11	.19±.12	.50	.45	.61±.07	.27±.12	.71	.44
Pegasus	.48±.09	.25±.10	.64	.39	.85±.08	.36±.13	1.0	.58
Pegasus(WX)	.33±.10	.14±.11	.42	.17	.93±.05	.43±.09	1.0	.58
DE								
Unpaired	-.96±.01	-.32±.07	-.96	-.40	-.16±.02	.56±.03	-.16	.54
Scenarios	-.97±.00	-.53±.04	-.97	-.55	.19±.02	.80±.01	.18	.81
Pegasus	-.97±.00	-.53±.06	-.97	-.55	.14±.03	.62±.03	.17	.62
Pegasus(WX)	-.97±.00	-.60±.05	-.97	-.60	.22±.02	.60±.03	.21	.60

Table 6: Comparison between unpaired and paired methods on the pursuer-evader task.
(All on-line figures for Random Search and DSM are calculated using the *on-line max* criterion.)

	MOUNTAIN CAR			PURSUER EVADER		
Number of Scenarios (*N*)	**On-line mean >.97**	**Test mean (on-line >.97)**	**Overfit**	**On-line mean >.97**	**Test mean (on-line >.97)**	**Overfit**
2	1.00±.00	.13±.09	.87±.09	1.00±.00	*-.51±.06*	1.51±.06
4	.99±.01	.65±.05	.35±.05	1.00±.00	*-.12±.09*	1.12±.09
8	.99±.01	.78±.07	.21±.07	1.00±.00	*-.21±.10*	1.21±.10
16	.98±.01	.94±.01	.04±.02	1.00±.00	.56±.06	.44±.06
32	.98±.01	.94±.01	.04±.02	1.00±.00	.77±inf	.23±inf
64	.98±.01	.95±.01	.03±.02	1.00±.00	N/A	N/A

Table 7: Overfitting in Pegasus.

	FIXED SCENARIOS				CHANGING SCENARIOS			
	On-line	**Test**	**On-line median**	**Test median**	**On-line**	**Test**	**On-line median**	**Test median**
MOUNTAIN CAR								
RANDOM	.95±.03	.84±.04	.99	.92	.93±.02	.90±.03	.97	.96
DSM	.80±.07	.65±.10	.99	.92	.75±.08	.71±.09	.96	.90
DE (N=8)	.77±.03	.69±.05	.80	.93	.56±.07	.64±.07	.64	.71
DE (N=2)	.92±.04	.45±.08	.98	.45	.78±.05	.86±.06	.89	.99
PURSUER EVADER								
RANDOM	.91±.04	.22±.10	1.0	.35	.77±.10	.34±.07	1.0	.46
DSM	.85±.08	.36±.13	1.0	.58	.64±.04	.69±.04	.64	.69
DE (N=8)	.14±.03	.62±.03	.17	.62	.15±.02	.80±.01	.15	.80
DE (N=2)	-.45±.02	-.19±.08	-.46	-.30	.27±.02	.82±.01	.27	.81

Table 8: Changing the Pegasus scenarios during learning.

	1024 TRIALS	**2048 TRIALS**	**4096 TRIALS**	**8192 TRIALS**	**16384 TRIALS**	**32768 TRIALS**	**65536 TRIALS**	**131072 TRIALS**
MOUNTAIN CAR								
N=4	**.22±.03**	.29±.05	.40±.05	.45±.05	.64±.04	.81±.03	.90±.02	.91±.01
N=16	.15±.03	.24±.02	.30±.02	.39±.02	.55±.03	.81±.02	.88±.02	.80±.02
N=64	.03±.03	.13±.02	.19±.02	.26±.02	.40±.02	.67±.03	.81±.03	.81±.03
ADAPTIVE N 95.0/99.5	.17±.02	**.32±.03**	**.43±.02**	**.56±.03**	**.76±.03**	**.91±.02**	**.93±.02**	**.94±.02**
ADAPTIVE N 99.5/99.5	.09±.03	.22±.03	.37±.03	.49±.03	.64±.03	.86±.02	.92±.02	.92±.02
PURSUER EVADER								
N=4	**-.64±.05**	**-.60±.05**	**-.52±.06**	**-.49±.06**	-.27±.06	.13±.06	.16±.06	.16±.06
N=16	-.81±.02	-.78±.02	-.66±.04	-.59±.04	-.38±.05	.06±.06	.32±.07	.35±.07
N=64	-.91±.02	-.87±.02	-.74±.04	-.65±.03	-.48±.03	-.14±.05	.21±.07	.28±.06
ADAPTIVE N 95.0/99.5	-.82±.03	-.74±.03	-.63±.03	-.50±.04	**-.12±.06**	**.37±.06**	**.67±.05**	**.68±.05**
ADAPTIVE N 99.5/99.5	-.89±.01	-.82±.03	-72±0.04	-.62±.04	-.46±.04	.02±.07	.42±.07	.45±.07

Table 9: Adapting the number of trials per policy with random search.

References

L. C. Baird. *Reinforcement learning through gradient descent.* PhD thesis, Carnegie Mellon University, Pittsburgh, 1999.

J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning. Technical report, Research School of Information Sciences and Engineering, Australian National University, 1999.

G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building.* Wiley, New York, 1978.

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

M. E. Harmon and L. C. Baird. Multiplayer residual advantage learning with general function approximation. Technical report, Wright Laboratory, 1996.

C. T. Kelley. Detection and remediation of stagnation in the nelder-mead algorithm using a sufficient decrease condition. Technical report, Department of Mathematics, North Carolina State University, 1997.

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

A. W. Moore and M. S. Lee. Efficient algorithms for minimizing cross validation error. In *Proceedings of the 11th International Conference on Machine Learning*, pages 190–198. Morgan Kaufmann, 1994.

R. Munos and A. W. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1348–1355, S.F., 1999. Morgan Kaufmann Publishers.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7 (4):308–313, 1965.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, San Francisco, CA, 1999.

A. Y. Ng and M. Jordan. PEGASUS:A policy search method for large MDPs and POMDPs. In C. Boutilier and M. Goldszmidt, editors, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 406–415, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.

J. Schmidhuber and J. Zhao. Direct policy search and uncertain policy evaluation. Technical Report IDSIA-50-98, IDSIA, 1998.

R. Schoknecht. System dynamics of the mountain car. Personal communication. Institute of Logic, Complexity and Decduction Systems, Kalsruhe, Germany, 2001.

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

M. J. A. Strens and A. W. Moore. Direct policy search using paired statistical tests. In *Proceedings of the 18th International Conference on Machine Learning*, pages 545–552. Morgan Kaufmann, San Francisco, CA, 2001.

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

R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

A. Teller and D. Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA, 1998. Morgan Kaufmann.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989. (To be reprinted by MIT Press.).

M. H. Wright. Direct search methods: once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)*, pages 191–208. Addison Wesley Longman, 1996.