2005 IEEE Swarm Intelligence Symposium 8-10 June 2005, Pasadena, California, USA pp30–37 $Revision : 1.33$ Corrected crossover rate

UNDERSTANDING PARTICLE SWARM OPTIMISATION BY EVOLVING PROBLEM LANDSCAPES

W. B. Langdon, Riccardo Poli, Owen Holland

Department of Computer Science,
University of Essex, UK

Thiemo Krink

EVALife Group,
University of Aarhus, Denmark

ABSTRACT

Genetic programming (GP) is used to create fitness landscapes which highlight strengths and weaknesses of different types of PSO and to contrast population-based swarm approaches with non stochastic gradient followers (i.e. hill climbers). These automatically generated benchmark problems yield insights into the operation of PSOs, illustrate benefits and drawbacks of different population sizes and constriction (friction) coefficients, and reveal new swarm phenomena such as deception and the exploration/exploitation tradeoff. The method could be applied to any type of optimizer.

1. INTRODUCTION

Particle Swarm Optimisation (PSO) [3] is an optimisation technique based on the collective motion of a flock of particles, the particle swarm. In the simplest (and original) version of PSO, each member of the particle swarm is moved through a problem space by two elastic forces. One attracting it with random magnitude to the best location so far encountered by the particle. The other attracting it with random magnitude to the best location encountered by any member of the swarm. The position and velocity of each particle are updated at each time step (with the maximum velocity being bounded to maintain stability) until the swarm as a whole converges to an optimum. This is usually very quick, in terms of the number of updates required, and the solutions are typically very good for a range of difficult problems.

The update rule for this basic PSO contains only two parameters: 1) the relative importance of the influences on a particle of the particle best and the swarm best solutions and 2) the number of particles in the swarm. The original derivation of PSO was an abstract version of the factors involved in the feeding behaviour of flocks of birds. This may have inspired early progress, which often took the form of adding terms based on biological or physical analogies. One of the most successful of these was the "inertia weight", a friction coefficient added to the velocity update rule.

Following Kennedy's graphical examinations of the trajectories of individual particles, and of their responses to variations in the key parameters [2], the first real attempt at providing a theoretical understanding of PSO was the "surfing the waves" model presented by Ozcan and Mohan [6]. Shortly afterwards, Clerc and Kennedy [1] developed a comprehensive 5-dimensional mathematical analysis of the basic PSO system. A particularly important contribution of that work was the use and analysis of a modified update rule, involving an additional constant, k, the "constriction coefficient". If k is correctly chosen, it guarantees the stability of the PSO without the need to bound velocities.

In spite of these theoretical contributions, we still do not have an adequate understanding of why certain parameter settings, or certain variants of the basic form, perform better or worse than other PSOs (or other optimisers) on problems of a given type. A conventional approach to this situation, which is common to other families of optimisers, would be to study the performance of various PSOs (and other algorithms) on a subset of a standard suite of problems, and attempting to identify the reasons behind relative success or failure. Unfortunately, the observed differences may be small, making it difficult to discern the source and nature of the differences. The technique introduced here turns this idea on its head: instead of studying the performance of two optimisers on a standard problem in the hope of finding an informative degree of difference, *we evolve new problems that maximise the difference in performance between the optimisers*. In this way, the underlying strengths and weaknesses of each optimiser are exaggerated and thereby revealed.

The next section describes our method. This is followed in Section 3 by a description of the GP used to create the new benchmarks, along with details of the PSO and other optimisers studied and results obtained (Section 4). Sections 5 and 6 discuss our results and conclude that simplistic assertions that "convergence" is required are unfounded and must be qualified by consideration of the problem to be solved.

2. APPROACH AND METHOD

We can easily use the standard form of genetic programming (GP) [4, 5] to evolve problems on which one search technique performs radically better or worse than another. We begin with a GP population in which each individual represents a function (a landscape) that can be searched by each of the two techniques. The fitness of an individual is established by taking the (signed) difference between the search performances of the two techniques on the function represented by that individual. With this approach, GP will tend to evolve benchmark problems where one technique outperforms the other.

It is important to note that we are using GP merely as a convenient tool for producing landscapes on which one optimiser (or parameter set) performs better than another. Each such landscape is the output of a single GP run. Once a landscape has been produced, we perform further multiple runs of the relevant optimisers on that landscape to establish the statistical significance of the apparent difference in performance. We then examine in detail the progress of each optimiser to try and understand the relationship between the shape of the landscape and the resultant performance.

To ensure that the landscapes produced are readily comprehensible, we restrict ourselves to two dimensional fitness functions (covering the square $-10 \ldots + 10$) with values $0 \ldots 1$. (The inclusion of values up to ± 10 should readily allow extension to discrete integer problems.) For simplicity the $-10 \ldots + 10$ range is divided into 2001 points at which the objective function is defined. On a microscopic level, this means that the search problem is composed of 2001^2 horizontal tiles, each 0.01 x 0.01. This is 4 004 001 points, so it is easy to find the global optimum by enumeration.

3. OPTIMISERS

3.1. Genetic Programming

We use a Java implementation of tinyGP [7] with 100 constants. Details are given in Table 1.

To reduce the inevitable noise when comparing stochastic techniques, each GP individual (i.e. each fitness landscape) is used five times. Each time we run both optimisers until they either find an optimum or exhaust their quota of trial points (e.g. maximum number of generations $\times$ swarm size). The fitness of the landscape (as far as GP is concerned) is the difference between the sums over the five runs of the number of trial samples taken by the two optimisers being compared. To further reduce the noise in the comparison, the two optimisers start from the same start points.

When running PSOs with different population sizes, the one with the larger population is run first; a subset of its initial positions and velocities is saved, and the smaller PSO is started from these positions and velocities. A similar technique is used when a PSO and a gradient following optimiser (Section 3.3) are being compared: the PSO is run first, a subset of the start positions is saved, and the gradient follower uses starting points from this set of saved positions.

The GP process turned out to be very reliable. In every case examined, a single run produced a landscape with the appropriate performance difference. These results are from single runs of each combination of optimiser type.

The optimisers we used in this initial investigation are described below. Although our primary focus is on the differences between various forms of PSOs, we are also interested in how PSOs perform in comparison to other optimisers, so we also included a representative gradient-following algorithm.

3.2. Partical Swarm Optimiser

We used Java implementations of two standard forms of PSO, one without constriction and one with constriction. Depending upon the experiment, the swarm contained either 10 or 100 particles and was run for up to 1000 generations. The initial random starting points and velocities were chosen uniformly at random from $-1 \ldots + 1$. Unless otherwise stated, the swarms do not use constriction, friction or velocity limiting. However in some cases the swarm is forced to remain in the feasible region. (I.e. particles straying outside the $-10 \ldots + 10$ box are forced back onto its boundary.)

3.3. Newton-Raphson Optimiser

The Newton-Raphson optimiser (N-R) is an intelligent hill-climber. If the initial point is an optimum, it stops. (The optimum value is *assumed* to be 1. Remember the GP is constrained to generate functions with values no larger than 1.) If the point is not at an optimum, N-R takes two steps from its current position. One in the x-direction and the other in the y-direction. From these measurements of the landscape, it calculates the local gradient. It then calculates the difference between the current value and the assumed optimum value of 1. From its estimate of the local gradient, it then calculates how far it would need to move, and in what direction, in order to reach an optimum value. It then jumps to this new point. If the new point is an optimum, it stops, and so on.

We have chosen a variant of N-R which incorporates several additional strategies to make it more robust. (Although the variant is more properly described as a pseudo-Newton-Raphson optimiser with restart, we will continue to refer to it as N-R.) The step used to estimate the local gradient is initially set to a large value, in this instance (1.0). If N-R fails to reach an optimum at the first attempt it tries again with the step size halved, in order to get a better estimate of the local gradient. Similarly, instead of trying to

Table 1. tinyGP Parameters

Function set:	$+ - \times$ DIV[a]
Terminal set:	x, y, 100 constants uniformly randomly chosen in the range $0 \ldots 1$
Fitness:	Landscape points sampled by optimiser A minus those by optimiser B. A and B start from same initial random start points. A and B run 5 times. Individual fitness values are not fixed but each is re-evaluated in each parent selection tournament. (Current fitness is used when selecting who dies).
Selection:	Steady state [8] binary tournaments for both parent selection and who to remove from the population.
Initial pop:	Trees randomly grown with max depth of 6 (root=0).
Parameters:	Population 10 or 1000. 10% crossover, 90% mutation, 2% chance of mutation per tree node. Optimiser initial points are chosen uniformly at random from square centred on origin.
Termination:	generation 10

[a] DIV is protected division I.e. if $|y| <= 0.001$ DIV$(x, y) = x$ else DIV$(x, y) = x/y$.

jump all the way to an optimal value, after a failure it will jump only a fraction of the way. (In our implementation, it will attempt to jump halfway on the second attempt, a quarter of the way on the third attempt, and so on.) These measures enable N-R to cope with non-linear problems, but at the expense of testing the landscape at more points.

Should the step size fall below 0.01 at any time, our Newton-Raphson optimiser will restart at another starting point chosen from the saved subset of initial starting points used by the competing PSO.

4. RESULTS

4.1. PSO v. Gradient Search

Genetic programming can readily find landscapes on which the PSO beats the quasi Newton-Raphson optimiser. The lower diagram of Figure 1 shows such a case. In 250 trials on this landscape, the swarm of ten particles always found one of the two optimal regions, while the N-R technique was successful only 131 times. By tracking the steps of each algorithm, it is possible to see why a landscape is easy or difficult for a particular technique. The PSO and N-R start near the origin where the gradient is low. Usually this causes the gradient follower to restart, or to take steps that are too large. The arrows show an interesting example where the initial step (horizontal arrow) is too big. By reducing its step size, the gradient follower starts to ascend one of the hills, but the progressive reduction in the step size causes it to get stuck just before it can reach the plateau.

The upper diagram of Figure 1 shows an evolved landscape where the Newton-Raphson optimiser does better than a standard PSO with ten particles. Gradient ascent takes on average only about 300 fitness evaluations compared to the PSO's 1 400. Interestingly, the swarm is more reliable: over 3 800 random starts, the N-R optimiser solves it only 98.76% of the time compared to the PSO's 99.95%.

The problem shown in the upper part of Figure 1 is not particularly difficult for PSO, but it is ideally suited for the

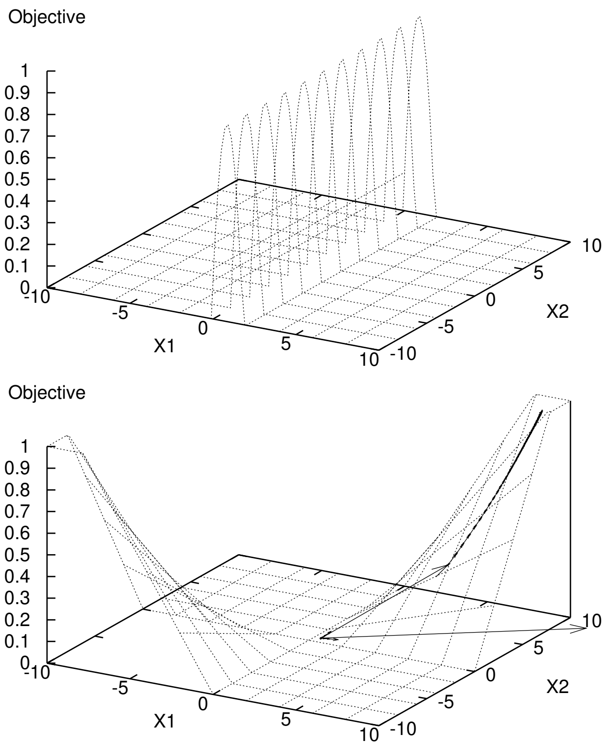

Figure 1. Upper diagram: landscape $0.176 + 1.72x - x^2$ where quasi Newton-Raphson optimiser outperforms PSO. The landscape is ideal for N-R, as it is simple, with optima close to 1 and with convex curvature. Lower: landscape $0.0127xy$ where PSO beats quasi N-R. Both algorithms start near the origin where the gradient is low. This usually causes the gradient follower to take steps that are too large, or to restart. The arrows show a case where N-R tends to overshoot as it climbs one of the hills. Therefore it reduces its step size, but this eventually causes it to get stuck just before it can reach the plateau.

N-R algorithm: the landscape is simple, the optima are close to 1, and the curvature is convex. Taken together, these factors mean that N-R produces good estimates of the location of the peak. These tend to be closer than it actually is, and so jumps tend to be improvements. This means the step size is not reduced. Together this produces very rapid progress from few evaluations.

4.2. Small Swarms beat Big Swarms

We have found that we can also automatically generate problems more suited to one type of PSO than to another. The simple landscape of Figure 2 is not deceptive (i.e. the gradient leads directly to the optima). However all the optima are some distance from the swarms' starting points. The problem is readily solved by both small and large swarms: a swarm of 100 particles usually takes four collective updates to reach the peak, whereas a swarm of 10 particles takes between 5 and 7). This indicates that the increased sampling associated with the larger population is delivering relatively little information. In terms of the number of fitness evaluations required to solve the problem, the smaller swarm is more efficient, needing only between 50 and 70 evaluations, in contrast to the ≈ 400 required by the larger swarm.

Figure 2 also shows the movements of the particles of the smaller swarm over the first seven update cycles, enabling us to see how the swarm is operating on this landscape. In this case it is clear that the dispersion of this small swarm produces at each step a reliable increase in the swarm best solution, yielding coherent motion towards the optimum. A larger and more dispersed swarm would find a better swarm best solution at each iteration, reducing the number of iterations, but the improvement would be sublinear in relation to the increased size of the swarm, and the number of evaluations required. Hence, *on simple landscapes small populations should be used*.

4.3. Big Swarms beat Small Swarms

Figure 3 shows an example where a swarm of 100 particles does better than one of 10. In this landscape, the global peak occupies only 2% of the region across which the swarms are initially distributed, but the local gradient leading to the false peak occupies almost half the search space. In all cases examined, the smaller population is always deceived into following the local gradient and therefore fails. (There is of course a finite probability that this will not happen.) In contrast, the larger swarm usually finds the global peak by chance during initialisation.

Genetic programming has automatically created (and tuned) an example where random search can do relatively well because the gradient information seen by the PSO is *deceptive* and leads to a local optimum from which it will never escape. Obviously the two particular population sizes

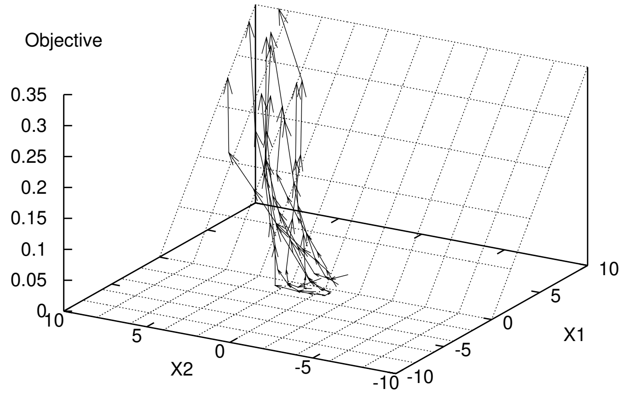

Figure 2. Non-deceptive landscape $0.032x$ evolved by GP, where gradient leads directly to all optima. The arrows show the movement of the ten particles in the swarm over the first seven generations.

are important to this example but we would expect GP to be able to devise other landscapes which would separate other pairs of small and large populations. It is an open question whether all such solutions would involve this mixture of needle-in-a-haystack and deception.

4.4. Constriction Wins

Figure 4 shows the outcome of an experiment in which we wanted to evolve problems where a constriction factor is beneficial to the search. Over 7 010 runs, a 10 particle PSO with constriction evaluated 76.46 (mean) test points, however without constriction it took 300.46.

Often both PSO took a similar amount of time. However in many runs, the unconstrained swarm took much longer. Figures 5–8 show other aspects of the example given in Figure 4. From this starting point with constriction only 11 cycles were needed (Figure 6) and the concentration of the swarm in the promising region near the solution is clear. However the unconstrained swarm oscillates for 116 generations before stumbling into an optimum (Figure 5).

Looking at the kinetic energy of the swarm clearly differentiates the two cases. Figure 7 shows without constriction the energy increases exponentially. Whilst Figure 8 shows, with a constriction factor of 0.7, kinetic energy falls exponentially. We would anticipate the gradient in Figure 8 to be given by the increase seen in Figure 7 less a multiplicative factor associated with the constriction of 0.7.

We conclude: *where an optimum is near the initial positions of the particles and the landscape is simple, constriction can help find it by reducing the energy of the swarm so helping to focus the search.*

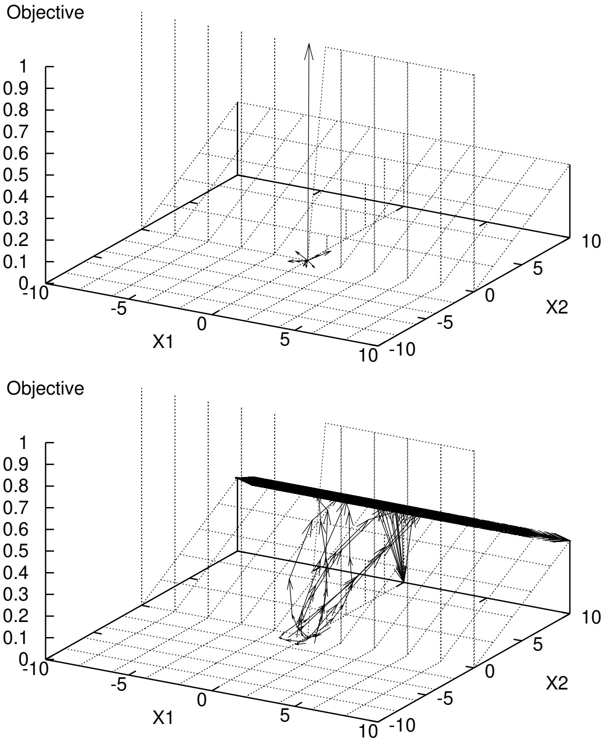

Figure 3. GP uses protected division operation to place optima at $y = 0$ and a valley at $x = 0$. Away from these two lines the landscape is $0.0312y$. Upper diagram: Swarm of 100 particles. Larger swarm stumbles into an optima purely by chance. Lower diagram: Starting from a subset of the initial conditions used by the large swarm, the 10 particle PSO does not find an optimum in the initial random search but instead is subsequently pulled up the gradient and away from the optima.

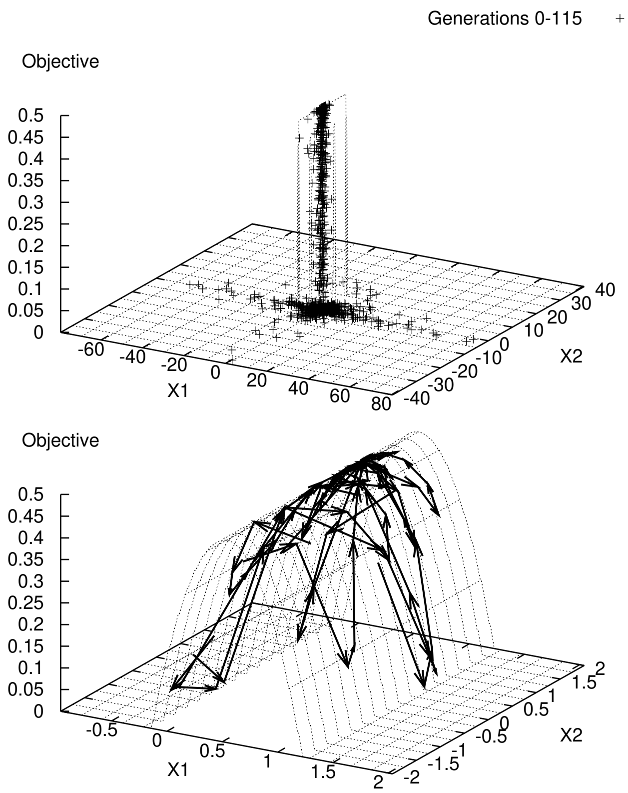

Figure 4. Landscape $0.8 + 1.08x - x^2$ evolved to favour the use of constriction. Upper diagram: The motion of a ten particle PSO without constriction. The search strays widely (note the large scale). In this run 116 generations were needed to reach an optimum. Lower diagram: The motion of the same swarm on the same landscape (drawn on a much smaller scale) with a constriction coefficient of 0.7. Starting from the same conditions only 11 generations where needed for a solution.

4.5. Constriction Fails

Figure 9 shows the opposite case, in which we were interested in finding fitness landscapes on which the use of a constriction factor was deleterious. Again, we used a 10 particle swarm. The upper diagram in Figure 9 shows the typical performance of a swarm without constriction. It finds an optimum in only seven update cycles. In contrast, the lower diagram shows the performance of a swarm with a constriction factor of 0.7. In spite of the strong gradient, the swarm becomes stuck. In fact, it fails to find an optimum even after 1000 update cycles. Figures 10 and 11 show, initially the constriction factor causes the energy to fall exponentially and the whole swarm to converge to a single fitness value. Once the whole swarm is together particle movements continue almost imperceptibly. *Constriction can impede search where the swarm seeks optima some distance from its starting locations.*

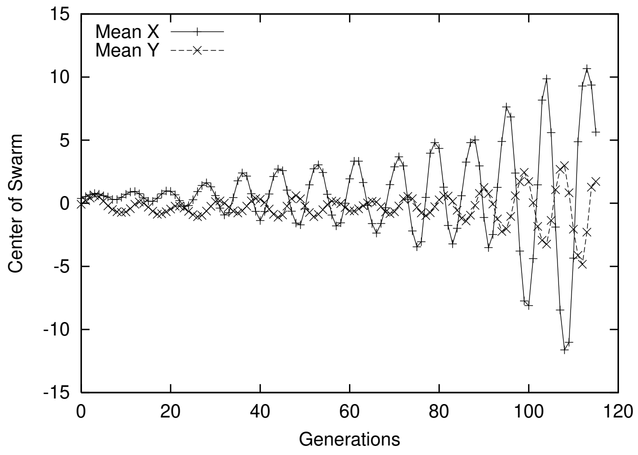

Figure 5. The oscillating and increasing amplitude of the search made by 10 particle PSO without constriction etc. on the landscape $0.8 + 1.08x - x^2$ of Figure 4.

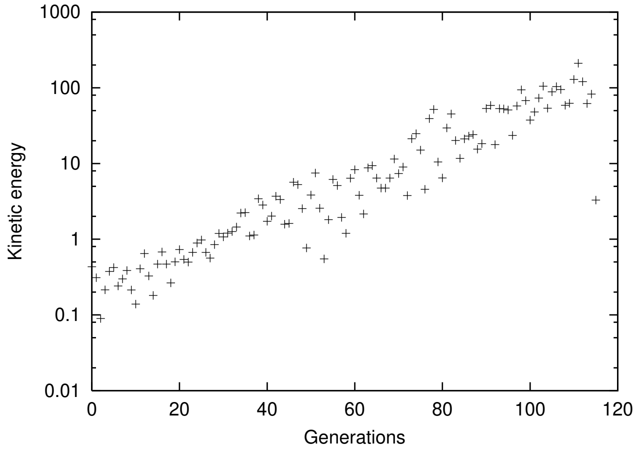

Figure 7. The increasing kinetic energy of PSO swarm of 10 particles without constriction on landscape $0.8+1.08x-x^2$ of Figure 4. (Note the logarithmic scale.)

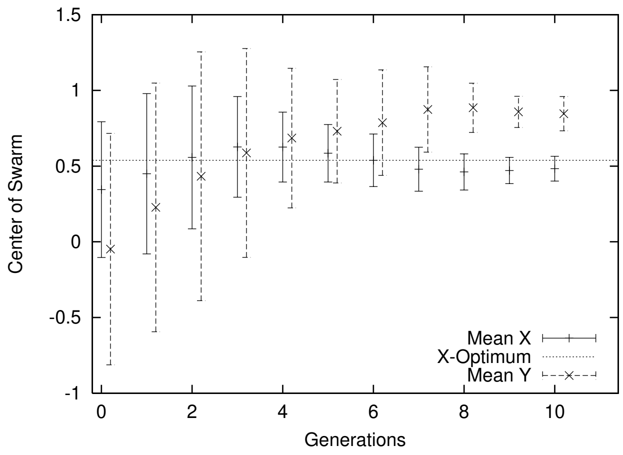

Figure 6. The search progress for the ten particle PSO with a constriction coefficient of 0.7 on the $0.8 + 1.08x - x^2$ landscape of Figure 4 (error bars show swarm spread, i.e. standard deviation of particles' positions). Note how the particles' position in the x dimension converges towards optimum.

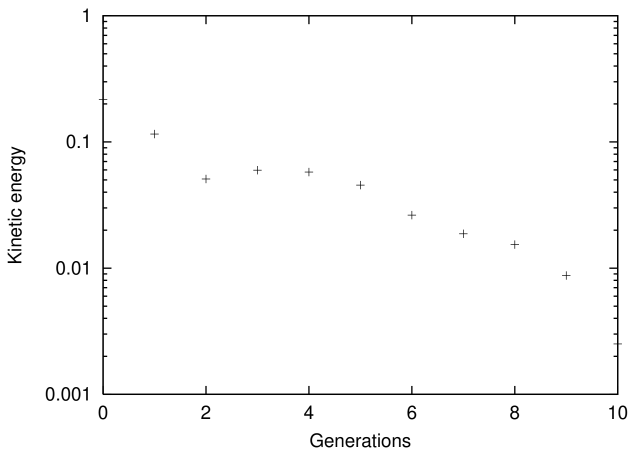

Figure 8. Kinetic energy of PSO swarm of 10 particles with constriction factor 0.7 on landscape $0.8+1.08x-x^2$. Note log scale.

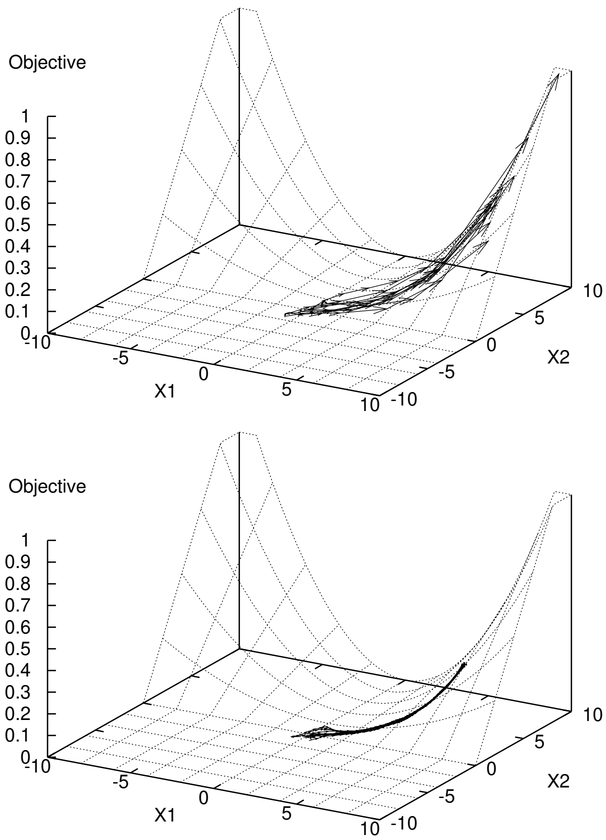

Figure 9. Upper diagram: landscape $0.00124x^2y$ evolved to hinder use of constriction. Showing the movement of a ten particle swarm without constriction or position or speed limits. In this run 7 generations were needed. Lower diagram: the movement on the same landscape when using a construction coefficient starting from the same initial conditions. The swarm becomes stuck despite the strong gradient. Even after 1000 generations the problem is not solved cf. Figures 10 and 11.

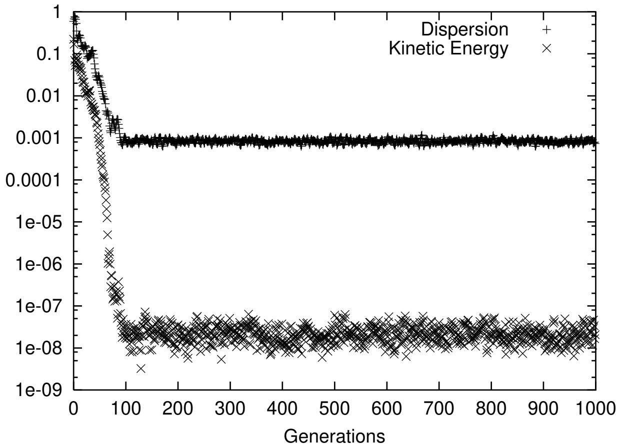

Figure 10. RMS spread of swarm members ($+$) and the mean kinetic energy ($\times$) of swarm members for first run of a ten particle PSO with constriction on the $0.00124x^2y$ landscape of Figure 9. (The swarm best does not improve after generation 63.)

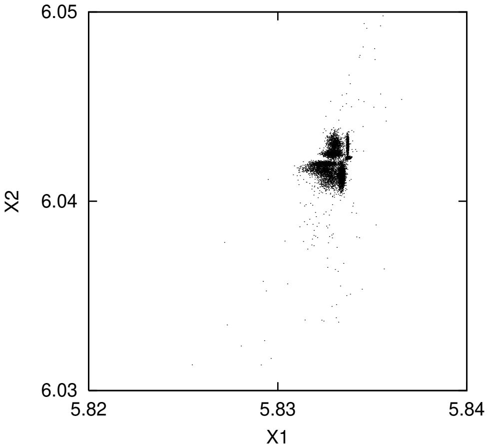

Figure 11. Position of ten swarm members on $0.00124x^2y$ landscape of Figure 9. In first run with constriction, all ten particles occupy the same 0.01×0.01 tile from generation 90 to generation 1000.

5. DISCUSSION

The use of the technique introduced in this paper has so far proved quite informative. Evolving landscapes to differentiate between particle swarm optimisers (PSO) and a gradient ascent technique (Section 4.1, Figure 1) confirms that gradient ascent can easily beat basic PSO on simple landscapes. However it also shows that only a slight increase in problem complexity is needed to reverse this situation.

Evolving landscapes to differentiate between population sizes confirms small populations can move across simple landscapes faster, i.e. with fewer fitness evaluations (cf. Section 4.2, Figure 2). More interestingly, the experiment in Section 4.3 (Figure 3) illustrates GP's ability to find surprising and illuminating solutions. We asked it to find a landscape suitable for a large population, which it did. However GP's solution does not use the swarm nature of the PSO. Instead it generates a "swarm deceptive" problem which drags the smaller swarm in the wrong direction but allows the larger initial population to find an optimum by pure chance.

Section 4.4 (Figures 4–8) shows a nice problem (generated by GP) where constriction narrows the search, concentrating the swarm about the optima. One of which is quickly found. Without constriction the swarm does not converge at all. Instead it becomes increasingly energetic and explorative. This behaviour is the opposite of that shown by many search techniques. For example, both genetic algorithms (GAs) and simulated annealing tend to explore in the initial phases of a run. Whilst, in the later stages they tend more and more to exploit what they have discovered. The evolved landscape helps us see that where an optimum is near the initial positions of the particles and the landscape is simple, constriction can help find it by reducing the energy of the swarm, so helping to focus a search that would otherwise diverge.

In contrast, Section 4.5 (Figures 9–11) shows an example where simple constriction can produce unwanted convergence. In cases like this where the swarm must travel some distance across the landscape to find an optimum, constriction may cause premature convergence to a non-optimal value. Indeed it can do this even when there is a strong gradient. This leads us to suggest that unconstrained PSOs may do well on problems where the optimiser should remain explorative for a length of time, rather than becoming focused early in the search process.

6. CONCLUSIONS

We have shown that it is practicable to use genetic programming to devise benchmark problems tailored to show the relative strengths and weaknesses of different optimisers, and in particular of different particle swarm optimiser (PSO) parameter settings. We have also demonstrated that the examination of the problem landscapes and of the optimisers' success or failure in dealing with them can give a deeper understanding of the ways in which particular optimisers work, and of the problems for which they may be most appropriate. Indeed we hope to extend and generalise this idea to compare and contrast radically different optimisation techniques. These and future results may be useful in increasing our understanding of the capabilities and weaknesses of both PSOs and competing search techniques, leading to improved and extended particle swarms (XPS).

Acknowledgements

This research is funded by EPSRC grant GR/T11234/01 "XPS: Extended Particle Swarms".

7. REFERENCES

[1] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transaction on Evolutionary Computation*, 6(1), February 2002.

[2] J. Kennedy. The behavior of particles. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on evolutionary programming*, pages 581–589, San Diego, USA, 1998.

[3] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[4] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[5] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[6] E. Ozcan and C. K. Mohan. Particle swarm optimization: Surfing the waves. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1939–1944, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[7] R. Poli. TinyGP. See TinyGP GECCO 2004 competition at `http://cswww.essex.ac.uk/staff/sml/gecco/TinyGP.html`, 2004.

[8] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In G. J. E. Rawlings, editor, *Foundations of genetic algorithms*, pages 94–101. Morgan Kaufmann, Indiana University, 15-18 July 1990. Published 1991.