CEC-2005, Edinburgh, 2-5 September 2005, Vol 1, pp81–88, IEEE press $Revision$: 1.20

Evolving Problems to Learn about Particle Swarm and other Optimisers

W. B. Langdon **Riccardo Poli**

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK

Abstract- We use evolutionary computation (EC) to automatically find problems which demonstrate the strength and weaknesses of modern search heuristics. In particular we analyse Particle Swarm Optimization (PSO) and Differential Evolution (DE). Both evolutionary algorithms are contrasted with a robust deterministic gradient based searcher (based on Newton-Raphson). The fitness landscapes made by genetic programming (GP) are used to illustrate difficulties in GAs and PSOs thereby explaining how they work and allowing us to devise better extended particle swarm systems (XPS).

1 Introduction

Particle Swarm Optimisation (PSO) [Kennedy and Eberhart, 2001] is 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 attracts it with random magnitude to the best location so far encountered by the particle. The other attracts 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 (possibly with the maximum velocity being bounded to maintain stability, cf. Sections 2.2 and 3.3) until the swarm as a whole converges to an optimum.

Following Kennedy's graphical examinations of the trajectories of individual particles and their responses to variations in the key parameters [Kennedy, 1998] the first real attempt at providing a theoretical understanding of PSO was the "surfing the waves" model presented by Ozcan and Mohan [Ozcan and Mohan, 1999]. Shortly afterwards, Clerc and Kennedy [Clerc and Kennedy, 2002] 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.

Differential Evolution (DE) is a very popular population-based parameter optimisation technique [Storn and Price, 1995; Price, 1999; Storn, 1999b]. In DE new individuals are generated by mutation and DE's crossover, which cunningly uses the variance within the population to guide the choice of new search points. Although DE is very powerful [Lampinen and Zelinka, 1999], there is very limited theoretical understanding of how it works and why it performs well [Zaharie, 2003].

In spite of some theoretical contributions [van den Bergh, 2001], 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. The same also holds for DE. The conventional approach to this situation, which is common to other families of optimisers, is to study the performance of various algorithms on a subset of a standard suite of problems, attempting to find 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*. Thus the underlying strengths and weaknesses of each optimiser are exaggerated and thereby revealed.

The next section explains how we use genetic programming to evolve fitness landscapes and gives details of three optimisers, which are tested against each other in Section 3. Sections 4 and 5 summarise our results and describe the conclusions that they lead us to.

2 Method

The method we use is similar to [Langdon *et al.*, 2005], where we mostly used it to investigate particle swarm optimisation, whilst here we are concerned to compare PSO with other optimisers. As before we use the standard form of genetic programming (GP) [Koza, 1992; Langdon, 1998; Langdon and Poli, 2002] 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 problem landscape that can be searched by each of the two techniques. In each generation, the fitness of an individual is established by taking the difference between the search performances of the two techniques on the function represented by the 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 as a tool, it is the landscapes that it produces that are important. These are the product of single GP runs. However, we consider in detail the performance of PSO etc. on them and we use multiple runs of the optimisers to show statistical significance of the difference in their performance on the automatically produced landscapes.

To ensure the fitness landscapes are easy to understand, we restrict ourselves to two dimensional problems (cover-

Table 1: tinyGP Parameters

Function set:	$+ - \times$
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 binary tournaments for both parent selection and who to remove from the population.
Initial pop:	Trees randomly grown with max depth of 4 (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. Depending on experiment, $-1 \ldots +1$ or $-10 \ldots +10$.
Termination:	generation 6 or 10

ing the square -10..10) and values $10^{-5} \ldots 1$. Outside the square fitness is defined to be exactly zero. That is, if $f(x, y)$ is the function evolved by GP, the fitness function it represents is $\delta(-10 \leq x \leq 10) \times \delta(-10 \leq y \leq 10)$ $max(10^{-5}, min(f(x, 0, 1))$. (The inclusion of values up to ± 10 should readily allow extension to discrete integer problems.) For simplicity the -10..10 range is divided into 2001 points at which the objective function is defined. So, on a microscopic level, the search problem is composed of 2001×2001 horizontal tiles, each 0.01×0.01. This is 4 004 001 points, so it is easy to find the global optimum by enumeration.

The optimisers being compared are run on each landscape until either they find a global optimum or they use up all the fitness evaluations they are allowed. To avoid problems with floating point arithmetic, finding a fitness value within 10^{-5} of the highest value in the landscape is regarded as having found a solution. Note this applies to all optimisers pairs, e.g. when PSO$\gg$DE and when DE$\gg$PSO. So it is neither an advantage nor a disadvantage for any optimiser.

2.1 Details of GP parameter settings

We used a simple steady state [Syswerda, 1990] genetic programming system, tinyGP, implemented in Java [Poli, 2004]. Details are given in Table 1.

2.2 Details of PSO parameter settings

We used a Java implementation of PSO. The swarm contained 30 particles and was run for up to 1000 generations. To highlight strengths and weaknesses of the core components of PSOs, constriction and friction were not used. (We will investigate them in future studies.) As with other optimisers, the initial random starting points were chosen for both techniques being compared. In most cases they were chosen uniformly at random from $-1 \ldots +1$. (The range $-10 \ldots +10$ was used in later experiments, Section 3.4.) Similarly the initial velocities were chosen from from $-1 \ldots +1$ ($-10 \ldots +10$ in Section 3.4).

It is well known that unless constrained, PSO swarms are unstable and tend over time to oscillate widely. Two techniques are commonly used to control this: 1) velocity limiting and 2) addition of a "constriction" or damping term. Initially we took advantage of our knowledge of the landscapes to be evolved by GP and used position (rather than velocity) clamping, to limit the PSO to the $-10 \ldots +10$ box known to contain the optima. However this might be felt to be giving the PSO an unfair advantage. So, in later experiments (Section 3.3 onwards) we reverted to the more usual velocity clamping and did not allow particle speeds to exceed 10 in any dimension.

2.3 Details of Newton-Raphson parameter settings

Newton-Raphson is an intelligent hill-climber. If the initial point is an optimum, it stops. Otherwise it takes two steps. 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 *assumes* that the global maximum will have a value of 1. (Remember the GP is constrained to generate values no bigger than 1). From the current value it calculates how much more is needed to reach an optimal value. From its estimate of the local gradient, it calculates how far it needs to move and in what direction. It then jumps to this new point. If the new point is an optimum, it stops.

It has several strategies to make it more robust. Firstly the initial step used to estimate the local gradient is large (1.0). If N-R fails, the step size is halved, to get a better estimate of the local gradient. Similarly instead of trying to jump all the way to an optimal value, on later attempts it tries only to jump a fraction of the way. (On the second attempt 1/2 way, third 1/4 and so on.) In this way N-R is able to cope with non-linear problems, but at the expense of testing the landscape at more points.

Should the step side fall to 0.01, our Newton-Raphson optimiser gives up and tries another random initial start point. (E.g. the starting position of the second PSO particle in the swarm.) N-R continues until either it finds an optimum or it has used the same number of fitness evaluations as maximum allowed to the other optimiser. (I.e. N-R cannot exceed (PSO or DE) population size $\times$ maximum number of generations.) This gives a robust optimiser.

2.4 Details of DE parameter settings

Unlike the other optimisers, we did not code our own implementation of DE. Instead we used Rainer Storn's Java implementation of Differential Evolution and followed his recommendations for parameter settings [Storn, 1999a; Storn, 2005]. The population was 20, i.e. $10\times$number of dimensions. We ran DE for up to 1000 generations. The crossover rate was 90% and the F factor was 0.8. We also used Storn's "DEBest2Bin" strategy.

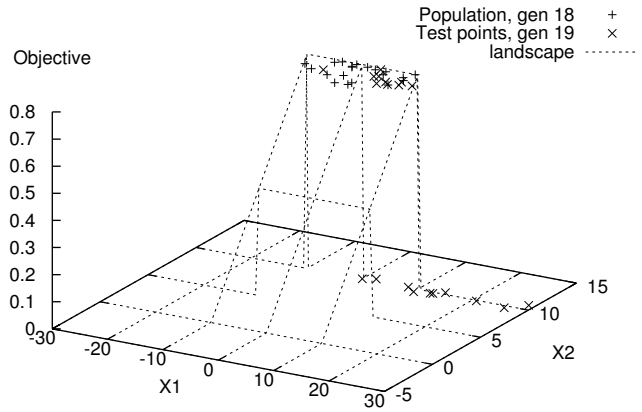

Figure 1: Cliff edge landscape where PSO outperforms DE. The gradient leads the DE population ($+$) to jump lemming like past the optima at the ridge. So many test points ($\times$) have zero fitness.

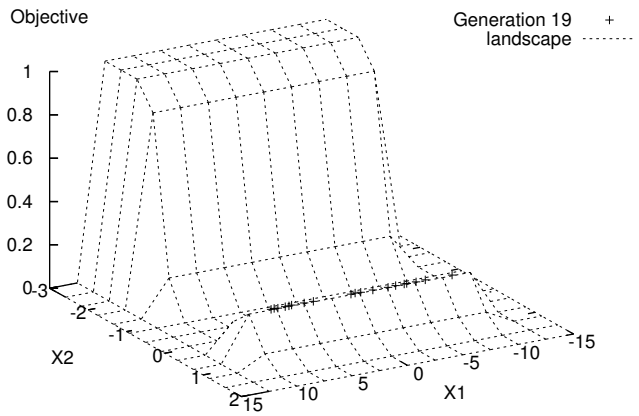

Figure 2: Smooth landscape where PSO outperforms DE. Both PSO and DE populations start near the origin but DE population always find local optima at the small ridge and ignores the huge global optima in the opposite direction. The PSO always finds one of the best points.

3 Experiments

3.1 PSO v. Unmodified DE

Genetic programming was easily able to find a landscape where a simple PSO could find a global optimum using fewer fitness evaluations than Rainer Storn's Differential Evolution java code and vice-versa. The evolved landscape is flat. Every point is an optimum value. Trivially the PSO code performs better because it performs one fitness evaluation and stops. Storn's code completes the evaluation of the initial population (twenty points). Thus PSO has a 20:1 advantage.

3.2 PSO v. immediate stopping DE

The DE code was altered, so that like the PSO code, it stopped immediately on finding a global optimum, rather than completing the evaluation of the current population.

Again GP finds a landscape which the PSO solves on average by generation 5 but DE never solves it. See Figure 1. DE finds this type of "cliff edge" landscape hard because the gradient on one side continously leads it to over shoot the global optimum. However we can regard this as GP having found an unfair comparison because it has placed the optima exactly at the PSO boundary. When a swarm member overshoots it is forced back inside the $-10 \ldots +10$ box. I.e. it is placed at an optimum, which solves the problem. Nevertheless the example is informative since it indicates *DE has a problem with "cliff edge" landscapes*.

While this example may appear artificial, [Schoenauer and Michalewicz, 1996] have suggested that global optima to constrained problems are often at the boundary between a smoothly varying feasible region and an infeasible region (where a constraint is violated). Depending upon how the constraint is handled, fitness in the infeasible region may be dramatically lower than in the feasible region. I.e. cliff edges may be common in constrained problems and so our results suggest that *DE might not perform well on constrained optimisation problems*.

3.3 Velocity Limited PSO

If we remove any PSO advantage, and use velocity clamping rather than position clamping (see Section 2.2), GP finds it harder to evolve a landscape suited to PSO. However increasing the GP population size to 1000 (see Table 1), enables GP to find a landscape (see Figure 2) which deceives DE into going to a local optimum. On average PSO finds the global optimum in 3 generations. DE never finds it.

This result is important because it shows that *Differential Evolution sometimes has a limited ability to move its population large distances across the search space if the population is clustered in a limited portion of it.* Indeed in other experiments (not reported) we noted that DE has problems with the spiral "long path problem" [Langdon and Poli, 2002, page 20]. This may be why Storn's WWW pages recommend the initial population should be spread across the whole problem domain ([Price, 1999, page 85] agrees)[1].

3.4 Wide Initial Population

Up to this point we have used the same initial points as for our PSO, i.e. scattered uniformly at random across the square $-1 \ldots +1$ centred on the origin. In the experiments described in Sections 3.4.1–3.4.6 we extended the initial population to the whole $-10 \ldots +10$ region. (We still use the same initial points for the PSO, DE and the other optimisers.) Again GP needed a larger population (1000) but managed to evolve a landscape where PSO outperforms DE.

3.4.1 PSO beats DE

The landscape evolved by GP is given in Figure 3. Using starting positions and seeds neither PSO nor DE had seen

[1] The reasons for DE getting stuck may be due to lack of movement opportunities. [Lampinen and Zelinka, 2000] calls this "stagnation". However they say "stagnation is more likely to occur" with "small population size ($\ll 20$)", while we have observed slow movement with larger populations as well.

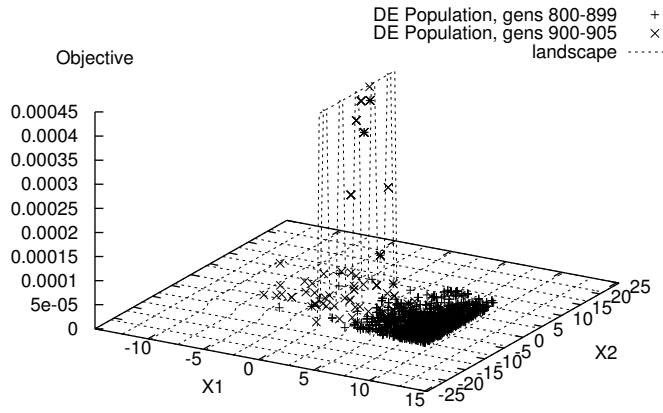

Figure 3: Landscape where PSO outperforms DE. Both PSO and DE initial populations are widely scattered. The central parabolic spike is given by $0.009x(0.44 - x)$. In this run the DE population converges toward one edge of the legal region (generations 800-899). Later it finds the spike by chance and then climbs its peak. PSO, being more stochastic, seems to have fewer problems stumbling into it by chance.

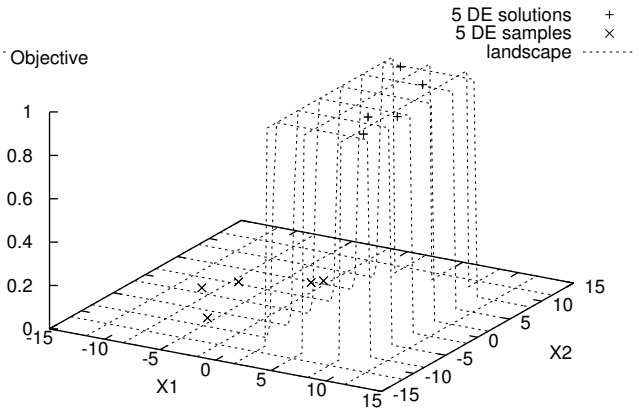

Figure 4: Evolved landscape $x^3 - x^2$. PSO and DE readily outperform Newton-Raphson.

during GP's evolution, the PSO was approximately three times more likely to solve the evolved problem using fewer fitness evaluations than DE. (In 50 runs, PSO won 36, DE 13 and they exactly tied once, p (two tailed sign test) = 0.0026. Mean 324 v. 424 evaluations.)

Again this landscape (Figure 3) is very instructive. *In the absence of landscape structure* (the landscape is largely flat), *DE tends to expand its search. This reduces its ability to find isolated optima in the original search region.* Note Storn's DE java implementation follows [Price, 1999, page 86]'s recommendation and, after initialisation, does not limit the search. Instead Figure 3 shows the fitness function is effectively bounding DE's search to the legal region.

3.4.2 PSO beats Newton-Raphson

GP readily evolves a landscape where our particle swarm optimiser beats our Newton-Raphson optimiser, see Figure 4. In fact it is the same landscape as it finds when asked to evolve a problem where DE beat Newton-Raphson (see Section 3.4.4). In 50 runs (with new starting positions) PSO and N-R always solved the problem but our PSO significantly outperformed our Newton-Raphson, on average evaluating 13.7 v. 75.9 points in the $x^3 - x^2$ landscape. This happens because approximately half the search space has low fitness and is flat. Newton-Raphson wastes many fitness evaluations where there is no gradient before giving up and restarting. In contrast the problem is easily solved by PSO and Differential Evolution initial random populations. *So again, our method gives useful information emphasising a weakness of gradient search.*

3.4.3 DE beats PSO

With a population of 1000, GP evolved a landscape (see Figure 5) on which Differential Evolution does consistently

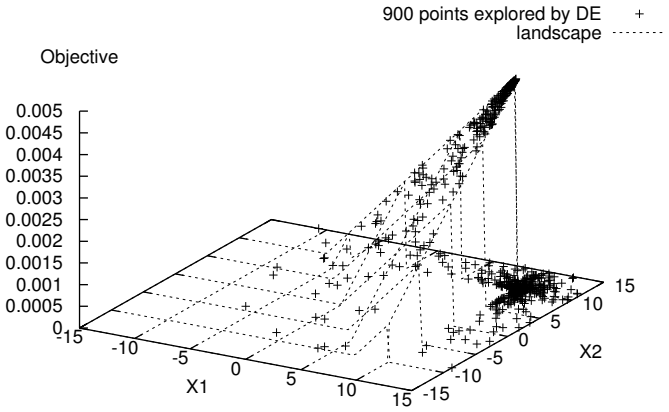

Figure 5: Evolved landscape $0.0003(0.11+x+0.7y)$ showing points explored by Differential Evolution.

(and significantly) better than our PSO. In 50 runs DE always solved it, taking on average 4 400 evaluations. Whilst PSO solved it only 6 times (within 1000 generations). The mean of the successful runs was 633 generations (19 000 evaluations).

It is clear that DE suffers again from the "cliff edge" problem, cf. Section 3.2, taking on average 221 generations to find the optimum. However, unlike Figures 1 and 3, there is a unique global optimum (occupying $2.5\ 10^{-7}$ of the feasible search space). This target proves to be too small for our PSO, which seldom finds it exactly. *This shows a weakness of the* standard *PSO: the particles are unable to home in on "narrow" global optima.*

3.4.4 DE beats Newton-Raphson

GP (with a population of 10) readily finds a landscape where Differential Evolution consistently beats gradient search. In fact it is the same $x^3 - x^2$ landscape as it used when PSO defeated N-R (see Section 3.4.2). This suggests that the $x^3 - x^2$ landscape in Figure 4 is generally a problem where most population based algorithms would beat N-R.

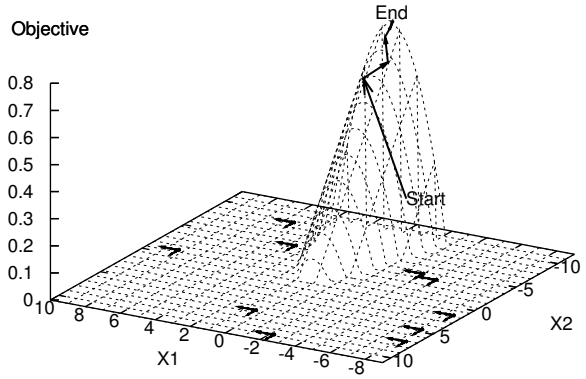

Figure 6: Evolved landscape $.011 + .077x(1-x) - .075y$ showing points explored by Newton-Raphson. Note arrows in the plane $z = 0$ where gradient search fails. However eventually N-R is restarted within the parabolic region with a gradient and (on the occasion shown) climbs it to one of the three global optima ("End"). To avoid clutter, in the path "Start"–"End" only successful moves are shown. The target occupies $7.5\ 10^{-7}$ of the feasible search space and proves to be too small for our PSO, cf. Figure 8.

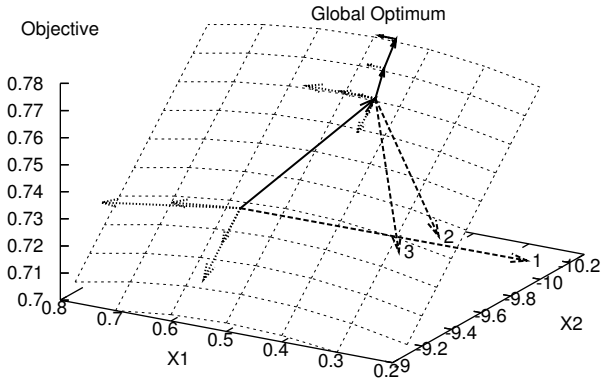

Figure 7: Last four successful steps in gradient based optimiser Newton-Raphson. (Same example as Figure 6.) Dashed arrows at right angles to each other indicate N-R's sampling of the problem landscape in order to estimate the local gradient. Using this, it then guesses where the global optimum is. Arrows 1, 2, and 3 show cases where it over estimated the distance to be travelled and passed the $y = -10$ boundary. Following each unsuccessful jump, N-R halves its step size and re-estimates the local gradient. Successful jumps are shown with solid arrows.

3.4.5 Newton-Raphson beats PSO

With a population of 1000, GP evolved the landscape shown in Figures 6–8. In 50 runs, using starting points chosen independently from those used by the GP, our gradient based searcher did significantly better than our PSO. In every run N-R found the global optimum ($x = 0.5, y = -10$), while the PSO was never able to find it.

Due to the flat top of the parabola there are three 0.01×0.01 tiles within 10^{-5} of the maximum value. Reaching any of them is regarded as solving the problem. The PSO samples a point very near the optimum but the particles' energy continues to increase. So, as time progresses, our PSO searches ever wider on this landscape, cf. Figure 8. I.e. the swarm samples points further and further from the optimum. This is interesting (although not unknown): *a PSO without constriction or friction can focus its search for only a limited number of iterations.* If the optimum is not found in that time, the PSO is unlikely to find it later. This is the opposite of a GA, which tends to focus its search in later generations, rather than expand it.

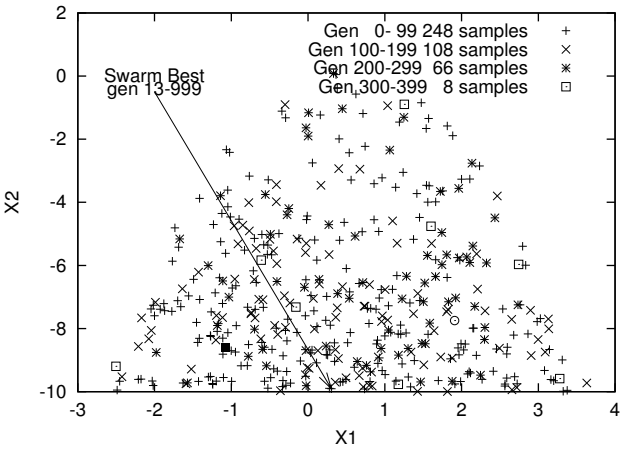

Figure 8: All points with fitness $> 10^{-5}$ sampled by PSO in first run (Same landscape as Figures 6 and 7.) By generation 13 the Swarm best lies within 0.001 of the global best. However it does not improve. Of the first 30×100 particle locations, only 248 lie in the $> 10^{-5}$ region, 108 from generations 100 to 199 and 75 after generation 200. In fact, the swarm (remember we are not using either position clamping or constriction) becomes increasingly energetic and more and more dispersed.

3.4.6 Newton-Raphson beats DE

With a population of 1000, GP evolved the parabolic problem shown in Figures 9 and 10. In 50 runs, using start positions chosen independently from those used by the GP, N-R significantly outperformed DE. N-R took on average 164 evaluations and always solved the problem, whilst DE required on average 3 200 but only solved it 43 times out of 50.

The bi-modal nature of the landscape means both optimisers are quite likely to head towards the lower ridge line (at $x = -10$, $z = 0.43$). However N-R wins over population based approaches because: 1) it ascends the gradient faster and 2) it stops when it reaches the lower hill top and

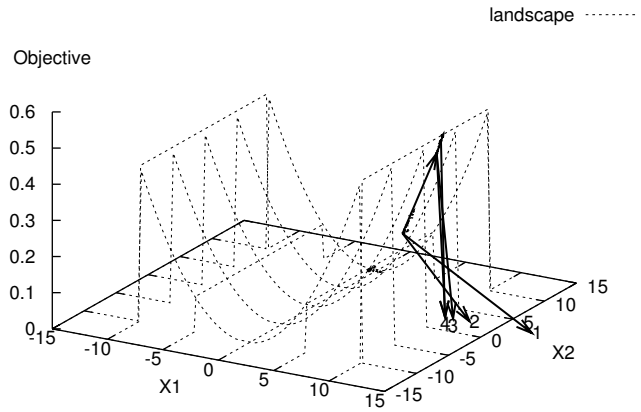

Figure 9: A run of gradient based optimiser. Dashed arrows at right angles to each other indicate N-R's sampling of the problem landscape, $0.0047(x+1)(x-0.38)$, in order to estimate the local gradient. After 15 fitness evaluations, N-R abandons its first start position (near (0,10)). However using only another 30 fitness evaluations it successfully climbs to an optimum from its second starting point. Arrows 1, 2, 3, and 4 show cases where it over estimated the distance to be travelled and passed the $x = 10$ boundary, cf. Figure 7.

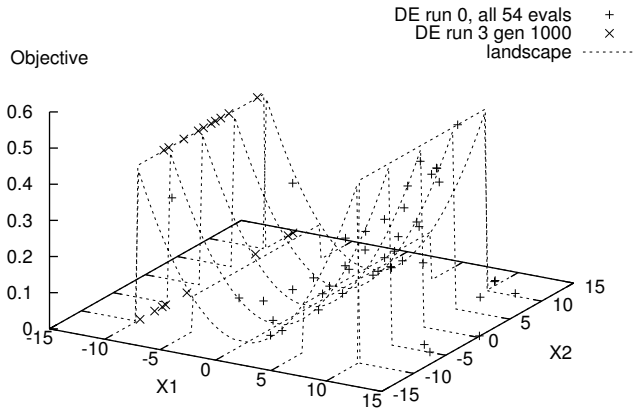

Figure 10: A successful ($+$) and a non successful ($\times$) DE run on same landscape as Figure 9. In the successful run DE only evaluated 54 points. In contrast in the other run DE ascends the slightly lower hill and gets stuck at the top of it. $\times$ show the points evaluated during the last generation.

restarts from another random position. Restarting virtually guarantees N-R will find the right ridge ($x = 10$, $z = 0.5$).

Again this landscape, Figures 9 and 10, is very interesting. It emphasises the differences in the strategy used to deal with local optima by N-R and DE. A hill-climber with restarts, deals with them by finding them and restarting. A population based algorithm (such as DE) deals with non-global optima by assuming they will have smaller basins of attraction than the global optimum. When this is true, most members of the population are more likely to sample the neighbourhood of the global optimum and so they can pull the whole population towards it. If the basins of attraction of local and global optima have almost identical sizes (like the landscape evolved by GP) this strategy may fail. In fact the problems where population based algorithms perform the worst, deceptive problems, non-global optima have have much bigger basins of attraction than that of the global optimum. This example shows *GP has automatically discovered the notion of deception for Differential Evolution*.

4 Discussion

A summary of our results (based on Sections 3.4.1–3.4.6) is given in Table 2.

It is a strength of our method that it is has not been necessary to set up a "fair" comparison *inside the GP fitness function*. The evolved fitness landscapes very clearly show the different strengths of the optimisers. Where one optimiser becomes stuck on an evolved landscape, increasing the number of generations from 1000 to 1500 or more will not substantially alter the comparison when the other technique takes on average 8 generations and solves it every time.

Rather than using predefined benchmarks it proved easy for genetic programming to find a simple landscape where evolution beats gradient search. (Simply by removing the gradient). Similarly GP with tiny populations (10) was able to find landscapes to fill most of the other niches in Table 2 (optimiser v. optimiser). However examination of several of them showed the difference in performance was not necessarily dramatic. In these cases GP was run again with a more common population size (1000).

Run time depends heavily on a number of factors. These include the computer used and which of the optimisers are being compared, their population sizes (1, 20 or 30), the number of generations they are allowed (up to 1000), the size of the genetic programming population (10 or 1000) and the number of GP generations (10). (Figures in brackets indicate values used in Section 3.) Nevertheless to give an indication of the costs of our technique, we note that the smallest GP run with the fastest heuristic (N-R) took about a minute on a 3GHz PC. The longest run with a population 100 times bigger took 32 hours. Doubtless, if needbe, these times could be greatly reduced by suitable code optimisation and/or parameter tuning.

5 Conclusions

Theoretic analysis of evolutionary algorithms in general, and particle swarm optimisers and differential evolution in particular, is very hard. While we have not abandoned this, it is clear that evolutionary computing itself can help our understanding. We have shown genetic programming by forcing alternative techniques to compete inside a single computer (rather than scattered across the pages of diverse conferences and journals) can readily produce examples which illustrate their comparative strengths and weaknesses, cf. Table 2.

Table 2: Summary of Landscapes evolved in Section 3.4

	Particle Swarm Optimisation (PSO)	Differential Evolution (DE)	Newton-Raphson (N-R)
PSO	–	GP population 1000 Section 3.4.1 Figure 3. PSO's more expansive search is more likely to find the central spike (occupying $\approx$ 1% of the feasible region). While DE tends to be less expansive than a PSO. Figure 2, of Section 3.3, gives an example where DE's more directed search is deceived by a local optima, whereas the PSO readily picks up on the bigger signal issued by the large global optimum. Possible problems with DE (such as "stagnation" [Lampinen and Zelinka, 2000] etc see Section 3.3) many not have arisen since the DE population is not small (we us a DE population of 20) and the initial population covers the whole feasible search space. However Section 3.3 notes concerns that sometimes DE populations after converging may have limited ability to move a distance.	GP population 10 Section 3.4.2 Figure 4. Gradient based search fails when there is no gradient.
DE	GP population 1000 Section 3.4.3 Figure 5. Here we did not use "constriction" in the PSO, this allows it to search widely. However it makes it less able to focus its search and home in a small high fitness target. In Section 3.4.3 GP exploits this and creates a problem with a tiny target area. DE is able to use the gradient information to locate the target. However, due to the "cliff edge" (cf. Section 3.2) DE takes a long time. Generally DE tends to fall over "cliff edges" and so may find difficulties optimising heavily constrained problems.	–	GP population 10 Section 3.4.4 Figure 4. Gradient based search fails when there is no gradient.
N-R	GP population 1000 Section 3.4.5 Figures 6–8. Both N-R and PSO are hampered by the large area without a gradient. PSO can come close to the optima quickly but, without constriction, friction or position limiting, the swarm becomes increasingly erratic and the stable swarm best is not sufficient to keep the swarm near the optimal region.	GP population 1000 Section 3.4.6 Figures 9 and 10. The evolved landscape is smooth, allowing gradient based search to reach the optima more quickly than population search. However GP reinforces this advantage by making the population multi-modal. This benefits our N-R, since it rapidly restarts on reaching a local optimum. While DE may be deceived into heading in the wrong direction.	–

Acknowledgements

We would like to thank Jim Kennedy, Chris Stephens, Thiemo Krink, Owen Holland and Cecilia Di Chio for helpful discussions. We would like to thank Rainer Storn for the use of his Differential Evolution java code. This work was funded by EPSRC grant GR/T11234/01.

Bibliography

[Clerc and Kennedy, 2002] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[Kennedy and Eberhart, 2001] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

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

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[Lampinen and Zelinka, 1999] Jouni Lampinen and Ivan Zelinka. Mechanical engineering design optimization by differential evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, chapter 8, pages 127–146. McGraw-Hill, Maidenhead, Berkshire, England, 1999.

[Lampinen and Zelinka, 2000] Jouni Lampinen and Ivan Zelinka. On stagnation of the differential evolution algorithm. In Pavel Osmera, editor, *Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing,*, pages 76–83, Brno, Czech Republic, 7-9 June 2000.

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

[Langdon *et al.*, 2005] W. B. Langdon, Riccardo Poli, Owen Holland, and Thiemo Krink. Understanding particle swarm optimisation by evolving problem landscapes. In Luca Maria Gambardella, Payman Arabshahi, and Alcherio Martinoli, editors, *2005 IEEE Swarm Intelligence Symposium, SIS 05*, Pasadena, California, USA, 8-10 June 2005. IEEE.

[Langdon, 1998] William B. Langdon. *Genetic Programming and Data Structures*. Kluwer, Boston, 1998.

[Ozcan and Mohan, 1999] Ender Ozcan and Chilukuri K. Mohan. Particle swarm optimization: Surfing the waves. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali 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.

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

[Price, 1999] Kenneth V. Price. An introduction to differential evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, chapter 6, pages 79–108. McGraw-Hill, Maidenhead, Berkshire, England, 1999.

[Schoenauer and Michalewicz, 1996] Marc Schoenauer and Zbigniew Michalewicz. Evolutionary computation at the edge of feasibility. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, volume 1141 of *LNCS*, pages 245–254, Berlin, September 22-27 1996. Springer.

[Storn and Price, 1995] Rainer Storn and Kenneth 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, March 1995.

[Storn, 1999a] Rainer Storn. DeApp - an application in java for the usage of differential evolution. http://http.icsi.berkeley.edu/ storn/code.html, 1999.

[Storn, 1999b] Rainer Storn. Designing digital filters with differential evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, chapter 7, pages 109–125. McGraw-Hill, Maidenhead, Berkshire, England, 1999.

[Storn, 2005] Rainer Storn. Differential evolution. http://www.icsi.berkeley.edu/ storn/code.html. Accessed 15 Feb 2005.

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

[van den Bergh, 2001] Frans van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, November 2001.

[Zaharie, 2003] Daniela Zaharie. Control of population diversity and adaptation in differential evolution algorithms. In R. Matousek and P. Osmera, editors, *9th International Conference on Soft Computing MENDEL, 2003*.