

# Evolving Problems to Learn About Particle Swarm Optimizers and Other Search Algorithms

W. B. Langdon and Riccardo Poli

**Abstract**—We use evolutionary computation (EC) to automatically find problems which demonstrate the strength and weaknesses of modern search heuristics. In particular, we analyze particle swarm optimization (PSO), differential evolution (DE), and covariance matrix adaptation-evolution strategy (CMA-ES). Each evolutionary algorithm is contrasted with the others and with a robust nonstochastic gradient follower (i.e., a hill climber) based on Newton–Raphson. The evolved benchmark problems yield insights into the operation of PSOs, illustrate benefits and drawbacks of different population sizes, velocity limits, and constriction (friction) coefficients. The fitness landscapes made by genetic programming reveal new swarm phenomena, such as deception, thereby explaining how they work and allowing us to devise better extended particle swarm systems. The method could be applied to any type of optimizer.

**Index Terms**—Differential evolution (DE), fitness landscapes, genetic programming (GP), hill-climbers, particle swarms.

## I. INTRODUCTION

**K**NOWING the modes of failure and safe operating limits of a tool (or system) is vital and forms the basis for all good engineering. However, analyzing complex real-world optimization algorithms, particularly those used in evolutionary computation (EC), has proved to be very hard. We highlight a variety of previously unknown ways that optimization algorithms may fail. That is, we do not propose new and improved algorithms but instead a new technique for analyzing industrial strength algorithms which are routinely used to solve real world problems.

Particle swarm optimization (PSO) [1] 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 towards the best location so far encountered by the particle. The other attracts it with random magnitude towards 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) until the swarm as a whole converges to an optimum.

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. Perhaps inspired by the original derivation of PSO (an abstract version of the factors involved in the feeding behavior of flocks of birds), early progress in PSO 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 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 [3]. Shortly afterwards, Clerc and Kennedy [4] developed a comprehensive five-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,  $\kappa$ , the “constriction coefficient.” If  $\kappa$  is correctly chosen, it guarantees the stability of the PSO without the need to bound velocities.

In spite of some theoretical contributions [5], we still do not have an adequate understanding of why certain PSO parameter settings, or certain variants of the basic form perform better or worse than other PSOs (or other optimizers) on problems of a given type.

The conventional approach to this situation, which is common to other families of optimizers, 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 optimizers on a standard problem in the hope of finding an informative degree of difference, we evolve new problems that maximize the difference in performance between the optimizers.* Thus, the underlying strengths and weaknesses of each optimizer are exaggerated and thereby revealed (cf. Fig. 1).

Differential evolution (DE) is a very popular population-based parameter optimization technique [6]–[8]. 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 [9], there is very limited theoretical understanding of how it works and why it performs well [10].

Covariance matrix adaptation (CMA) [11], [12] is a robust evolutionary strategy (ES), in which not only is the step size of the mutation operator adjusted at each generation, but so too is the step direction in the multidimensional problem space, i.e., not only is there a mutation strength per dimension but their

Manuscript received August 30, 2005; revised June 23, 2006 and August 29, 2006. This paper was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant GR/T11234/01.

The authors are with the Department of Computer Science, University of Essex, Colchester CO4 3SQ, U.K. (e-mail: wlangdon@essex.ac.uk; rpoli@essex.ac.uk).

Digital Object Identifier 10.1109/TEVC.2006.886448

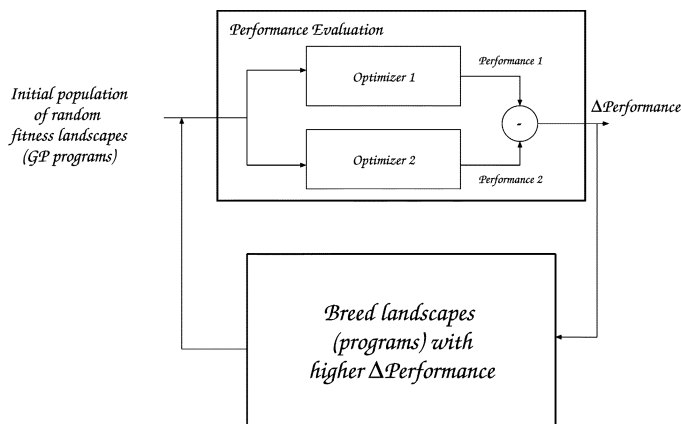


Fig. 1. Diagrammatic representation of the evolution of fitness landscapes using GP. In the GP system, the difference in performance of the two optimizers is used as the fitness measure associated to each program (fitness landscape).

combined update is controlled by a covariance matrix whose elements are updated as the search proceeds. Covariance matrix adaptation-evolution strategy (CMA-ES) also includes powerful heuristics to set search parameters, detect premature convergence, and a restart strategy which doubles the population size on each restart [13], [14]. CMA is a formidable adversary. At the recent Congress of Evolutionary Computation, it was the winner of the Real-Parameter Optimization competition [15], [16].

Since the presentation of the No Free Lunch theorems [17] at ICGA in 1995, it has been known that for an optimization algorithm to solve a problem of interest well, there must be other problems where it does poorly. Often, it is argued that this is not useful since there are a huge number of problems and so the other problems are unlikely to be of interest. Indeed [18] evolves test functions which are so hard that random search does better than a robust  $(1+1)$ -ES evolutionary strategy with a tabu list. Similarly, [19] devises an evolutionary scheme devoted to finding very hard binary constraint satisfaction problems. In the future, we may see coevolutionary systems [20] which both try to evolve difficult problems and evolve hyperheuristic algorithms to solve them [21]. However, we are concerned with today's practical continuous optimization algorithms, in particular, PSOs.

The next section explains how we use genetic programming (GP) to evolve problem landscapes and gives details of the four optimizers. These are tested against each other in Sections III–V. Sections III–V not only describe the experiments but also discuss the numerous lessons that we can learn from them. Sections VI and VII summarize our results and describe the conclusions that they lead us to.

## II. METHOD

The method (Fig. 1) uses the standard form of GP [22]–[25] to evolve problems on which one search technique performs radically better or worse than another. (Preliminary results with this approach have appeared in [26] and [27].) 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. Note the difference between the two notions of fitness used. One is the fitness in the fitness landscapes, and this is what is seen by the two optimizers in Fig. 1. The second notion is the fitness of the programs representing such landscapes. This is measured as the difference in performance between the two optimizers and it is used in the GP system to drive the evolution of landscapes.

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 optimizers to show statistical significance of the difference in their performance on the automatically produced landscapes. All the quoted results have a  $p$  of 1% or better.

To ensure the fitness landscapes are easy to understand, we restrict ourselves to two-dimensional problems covering the square  $-10 \dots 10$  and with values lying between  $10^{-10}$  and 1. Of course, the benchmarks can be generalized to higher dimensions. Outside the square fitness is defined to be exactly zero.

Although this is not a strict requirement for our method, in order to assess the performance of our optimizers, we have used knowledge about the position of the global optimum in a landscape. This needs to be computed for each evolved landscape. Therefore, for simplicity, the  $-10 \dots 10$  range is divided into 2001 points at which the objective function is defined. So, on a microscopic level, the search problem is composed of  $2001 \times 2001$  horizontal tiles, each  $0.01 \times 0.01$ . This is 4 004 001 points, so it is easy to find the global optimum by enumeration.

The optimizers 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  $10^{-10}$  of the highest value in the landscape is regarded as having found a solution. Note that this applies to all optimizer pairs, e.g., when we evolve problems where the PSO does better than DE, as well as when we evolve problems where DE does better than the PSO. So, it is neither an advantage nor a disadvantage for any optimizer.

### A. Details of GP Parameter Settings

We used a simple steady-state [28] GP system, tinyGP, implemented in Java [29]. Details are given in Table I. The GP fitness function uses the difference in performance of the pair of optimizers being compared. To make the comparison as fair as possible, where possible, the two optimizers are started from the same point so neither is disadvantaged by the other starting from a particularly favorable location. With particle swarms, population approaches and optimizers which restart, we keep track of multiple start conditions rather than just a single location, e.g., when comparing PSOs with different population sizes, the starting positions and velocities of the smaller swarm are a subset of those used by the larger. Similarly, our hill-climber starts from the initial location of one of the swarm particles. If it

TABLE I  
TINY GP PARAMETERS

Function set:	$+ - \times \div$
Terminal set:	$x, y, 100$ constants uniformly randomly chosen in the range $0 \dots 1$
Fitness:	Difference in number of evaluations taken by optimisers A and B. Each individual 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.
Parameters:	Population size 10, 100 or 1000. 90% crossover, 10% mutation 2% chance of mutation per tree node. Optimiser initial points are chosen uniformly at random from a square centred on origin of size, depending on experiment, either $-1 \dots +1$ or $-10 \dots +10$ .
Termination:	Generation 10

needs to restart, it uses the same initial location as the next particle. If all the initial conditions are used up, restart points are generated at random.

To minimize the effects of random factors, such as the pseudorandom numbers used, both optimizers are run five times. Finally, the steady-state approach allows us to reevaluate GP individual fitness at each selection tournament. Jin and Branke [30] give a comprehensive survey, which includes other ways of dealing with noisy fitness functions.

#### B. Details of PSO Parameter Settings

We used a Java implementation of PSO (see Table II). In the PSO versus PSO experiments (Section III), the swarm contained either 10 or 100 particles and was run for up to 1000 (or 100) generations (maximum 10 000 fitness evaluations), while in the PSO versus CMA, DE, or hill-climber (N-R) experiments (Section IV), the swarm contained 30 particles and was run for up to 1000 generations (maximum 30 000 fitness evaluations). Unless otherwise stated, the speed of the particles is limited to 10 in each dimension and constriction (friction) was not used. The default value for the coefficients  $c_1$  and  $c_2$  of the forces towards the particle best and swarm best are 0.5. As with other optimizers, the initial random starting points were chosen for both techniques being compared. They were chosen uniformly at random from either  $-1 \dots +1$  or  $-10 \dots +10$ . Similarly, the initial velocities were chosen from either  $-1 \dots +1$  or  $-10 \dots +10$ .

#### C. Details of Newton–Raphson (N-R) Parameter Settings

Newton–Raphson (N-R) 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. Note N-R has access to a small amount of not unreasonable domain knowledge. Typically, this makes little difference, however, Section IV-F gives one example where GP turned this assumption against N-R.) From the current value, it calculates how much more is needed to

reach an optimal value (i.e., the fitness of the best solution to the problem). 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.

Our implementation has several strategies to make it more robust. First, 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, then  $1/4$  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 size fall to 0.01, our N-R optimizer 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 the maximum allowed to the other optimizer (e.g., N-R cannot exceed the optimizer population size  $\times$  maximum number of generations). This gives a robust optimizer. Fig. 2 shows the last few steps where N-R succeeds in finding a unique optimum.

#### D. Details of DE Parameter Settings

Unlike the first two optimizers, we did not code our own implementation of DE. Instead, we used Rainer Storn’s Java implementation of DE. We modified the source slightly so that DE stopped immediately when it found a solution. (The down loaded code fully evaluates each generation [27, Sec. 3.1].) We followed Storn’s recommendations for parameter settings [31], [32]. The population was 20, i.e.,  $10 \times$  number of dimensions. We ran DE for up to 1500 generations (i.e., 30 000 fitness evaluations). The same maximum number of evaluations as the PSO, N-R, and CMA. The crossover rate was 90% and the F factor was 0.8. We also used Storn’s “DEBest2Bin” strategy.

#### E. Details of CMA Parameter Settings

CMA-ES is a sophisticated technique and so we are pleased to have been allowed to use the inventor’s Java implementation without changes.

The initial population is created by mutation. Thus, it is not possible to use exactly the same starting points as the other three search algorithms. This potentially makes the GP fitness function more noisy but, as we shall see, GP was able to cope with the noise.

Since, in the initial generations, CMA’s mutation algorithm is operating blind, it scatters sample points widely. This means about 30% of sample points lie outside the square  $-10 \dots 10$ . This was felt to place CMA at an unfair disadvantage and so, after discussions with Nikolaus Hansen, we used CMA’s boundary option to force all test points into the feasible region. The boundary option means, about 30% of initial test points lie exactly on the boundary of the square  $-10 \dots 10$ .

We used the CMA defaults (which vary according to dimensionality of the problem). The defaults include doubling the population size and restarting the evolution strategy every time stagnation is detected.

TABLE II  
DEFAULT PSO PARAMETERS

Swarm size:	30 particles.
Constriction:	The constriction factor is either not used (default) or 0.7
Swarm positions:	Not limited (i.e. the default is to allow particles to leave $-10 \dots +10$ ) however $\pm 10$ bounds are used with CMA in Section V.
Swarm speed:	Particle velocities bounded in $\pm 10$ . PSO update rule (Java code):  <pre>//pop[i] position of particle i (vector of n dimensions) //popv[i] velocity of particle i (vector of n dimensions) //best[i] best position found by particle i (vector of n dimensions) //clampV() clampV() optionally limit particle position and speed //bparticle i-index of particle which is currently the best in the swarm  for (int i = 0; i &lt; particles; i ++ ) {   for(int j=0; j&lt;pop[i].length; j++) {     double R1 = rd.nextDouble();     double R2 = rd.nextDouble();     popv[i][j] = clampV(Constriction*(popv[i][j] +       c1 * R1 * ( best[i][j]-pop[i][j]) +       c2 * R2 * (best[bparticle][j]-pop[i][j]) ));     pop[i][j] = clampX(pop[i][j]+popv[i][j]);   }//endfor each coefficient   fitness[i] = fitness_function( pop[i] );   updatebest();   if(fitness[i]&gt;=landscape_max) break; } //endfor each particle</pre>
Neighbourhood:	Fully connected. I.e. there is a single global swarm best visible to every member of the swarm.
Initial pop:	Particle positions are chosen uniformly at random from a square centred on the origin of size, depending on experiment, either $-1 \dots +1$ or $-10 \dots +10$ .
Init speed:	Particle velocities are chosen uniformly at random from $\pm 1$ or $\pm 10$ .
Termination:	Generation 1000.

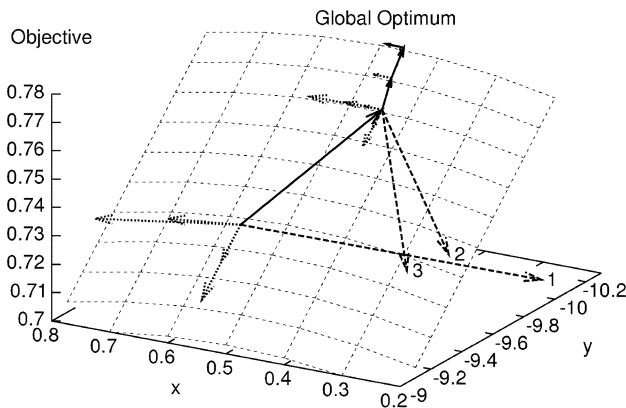


Fig. 2. Last four successful steps in gradient-based optimizer N-R on  $0.11 + 0.77x(1-x) - 0.075y$  landscape [27, Fig. 7]. Dotted arrows at right angles to each other indicate N-R's sampling of the problem landscape in order to estimate the local gradient. Using the gradient, N-R guesses where the global optimum is. Arrows 1, 2, and 3 show cases where it overestimated the distance to be traveled and passed the  $y = -10$  boundary. Following each unsuccessful jump, N-R halves its step size and reestimates the local gradient. Successful jumps are shown with solid arrows.

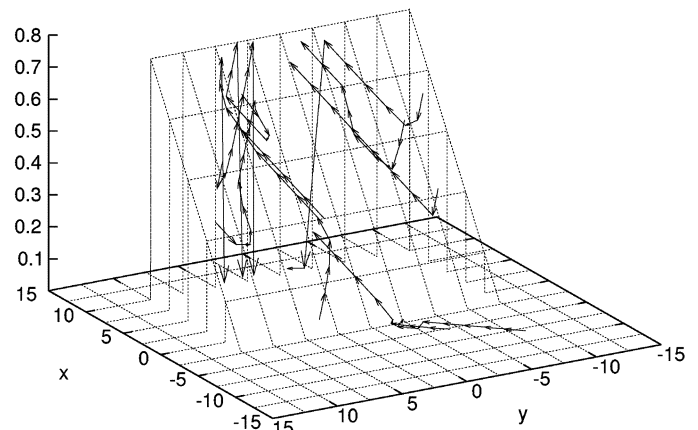


Fig. 3. Nondeceptive landscape  $0.127 + 0.063x$  evolved by GP (population 10), where the gradient leads directly to all optima. The arrows show the movement of the ten particles in the swarm for the first eight generations (maximum speed 1) when a particle is within 0.002 of the optima. To reduce clutter, fluctuations between generation 8 and the end the run in generation 39 are not plotted. On average, a swarm with 100 particles takes 2-1/2 times as many fitness evaluations to find an optimum (i.e., 11 generations versus 42 for the smaller swarm).

### III. EXPERIMENTS—COMPARISON OF DIFFERENT PSOS

#### A. Problems Where Small Swarms Beat Big Swarms

GP can automatically generate problems more suited to one type of PSO than to another. In the simple (nondeceptive) landscape of Fig. 3, the gradient leads directly to the optima. It is readily solved by both small and large swarms: a swarm of 100 particles on average takes 11 collective updates to reach the peak, whereas a swarm of 10 particles takes 42. This indicates that the increased sampling associated with the larger population is delivering relatively little additional information.

In terms of the number of fitness evaluations required to solve the problem, the smaller swarm is more efficient, needing on average only 420 evaluations, in contrast to the 1100 required by the larger swarm. However, both beat random search.

Fig. 3 also shows the movements of the particles of the smaller swarm enabling us to see how the swarm is operating on this landscape. During the first seven updates, 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. Once near the top edge, the swarm oscillates for some time before finding a maximum value. A

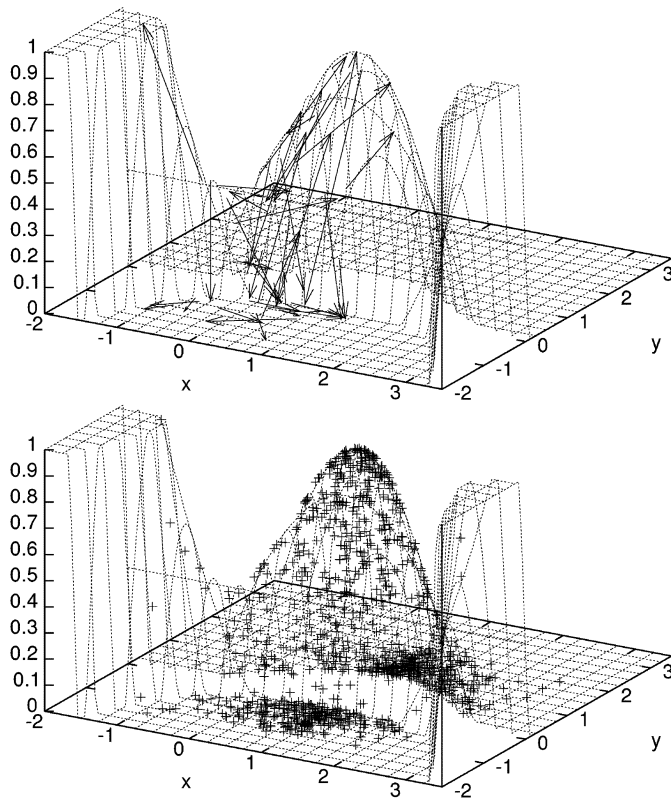


Fig. 4. Evolved landscape  $y(1.32 + 1.78x - x^2 - y) + 0.37$  containing a deceptive peak at 0.9,0.5 with large global optima either side of it. Upper diagram: A particle in a swarm of 100 starts close to the optima at  $x \approx -1.5$  and reaches them in the second generation (119 fitness evaluations). Lower diagram: Starting from a subset of the initial conditions used by the large swarm, all ten particles of the smaller swarm are attracted to a deceptive peak. However, the maximum speed (1) permits the swarm to remain sufficiently energetic to stumble towards the optima, one of which is found in generation 107 (1079 fitness evaluations, plotted with +).

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. This suggests that *on simple landscapes small populations should be used*. By a simple landscape, we mean one, e.g., a smooth unimodal landscape, where in almost every orbit the particles improve their personal best. This in turn means that in almost every generation the swarm improves best. Similarly, we would expect other optimizers to find improved solutions almost continually. Therefore, the microscopic progress rate [33] gives a good prediction of overall (macroscopic) performance. Given this, increasing the population size does not increase the size of improvements or the frequency of finding them, in proportion to the extra effort required by the increased population.

### B. Problems Where Big Swarms Beat Small Swarms

Fig. 4 shows an example where a swarm of 100 particles does better than one of ten. In this deceptive landscape, twin global plateaus lie slightly outside of the region across which the swarms are initially distributed. However, this region contains a false peak. Initially, in most cases, a member of the larger swarm lies sufficiently close to one of the global peaks to be able to move rapidly to it. However, with only ten particles, usually

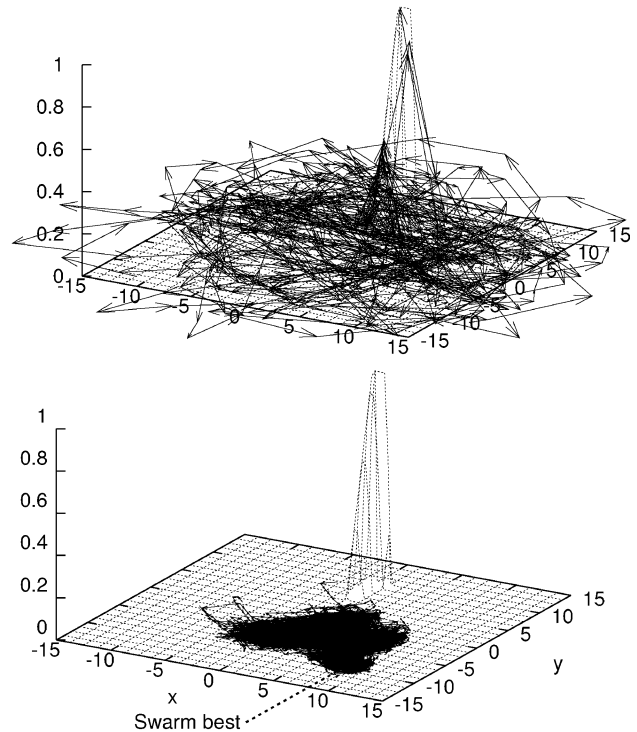


Fig. 5. GP (population 1000) evolves a landscape  $0.54x - x^2 + 0.24y - 1.26$  containing a single peak. Although it is within the initial range of the swarm, in these two runs, no particle initially samples it. Therefore, the swarm initially gets no fitness guidance and moves at random. Upper diagram: The swarm of ten particles with a maximum speed of 10 finds an optimum in generation 65. Lower diagram: Starting from the same initial positions (but speed limited to 1), the lower speed limit impedes search and no solution is found in 1000 generations. Instead, the swarm oscillates about the stationary swarm best.

all of the smaller swarms lie close to the deceptive peak and are attracted to it. Once either swarm is attracted to the central peak, it takes many generations to break free. This leads to a large variation in the number of fitness evaluations, but on average, the smaller swarm takes more than three times as many fitness evaluations.

Using a population of size 1000, GP has automatically created (and tuned) an example where the initial particle positions and velocities are important because most of the gradient information seen by the PSO is *deceptive* and leads to a local optimum from which it takes a long time to escape. Obviously, the two particular population sizes are important to this example but there are other cases where GP has been able to devise a landscape which separates small and large populations [26].

### C. Problems Where Fast Particles Win

Fig. 5 shows a landscape evolved to prefer energetic swarms. In 100 runs with a maximum speed of 10 in both dimensions, a swarm of ten particles always found an optimum, taking on average 67 generations. Whereas when speed is limited to 1, only 73 runs found a solution within 1000 generations. If we exclude the other 27 initial starting conditions, there is little difference in performance. Which of the slower runs fail seems to depend to some extent on the swarm's initial conditions. However, the faster swarm is able to solve the problem even when given unfavorable starting conditions. Note that GP has evolved a problem where the faster swarm is more robust than our slower one.

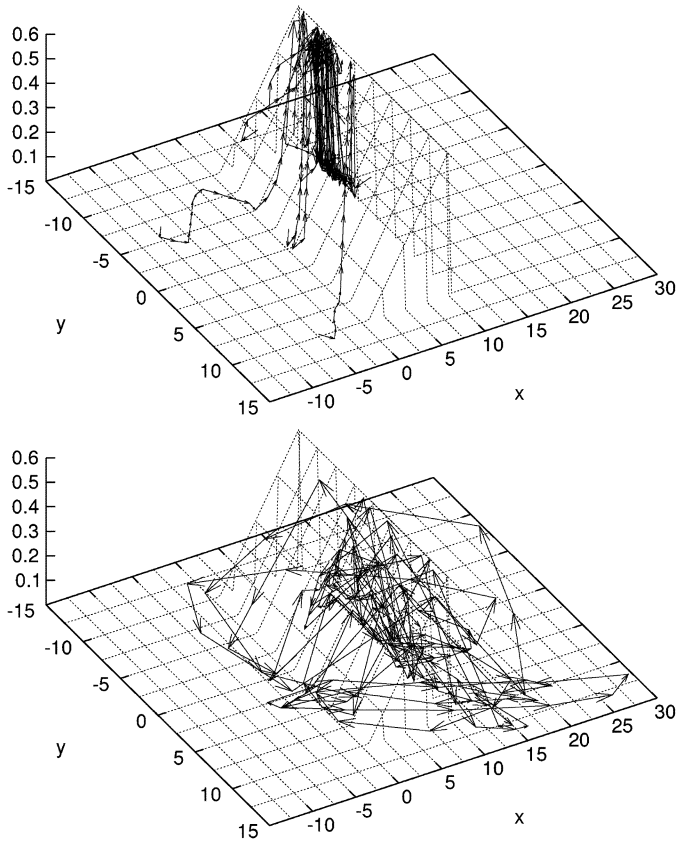


Fig. 6. GP (population 10) evolves a landscape  $0.063x - 0.052$  containing a plane ramp. A slowly moving swarm (arrows, upper diagram) finds an optimum in 25 generations. Lower diagram: Starting from the same initial positions, the fast moving swarm oscillates widely and takes 150 generations. To reduce clutter, only the motion of one of the ten particles (the one which eventually finds a global optimum) is shown.

#### D. Problems Where Fast Particles Lose

Fig. 6 shows a very simple landscape which a speed limited swarm (max 1) is able to solve every time in 50 runs. In contrast, a fast moving swarm searches much more widely, takes more time, and only solves it in 46 of 50 runs. Excluding the four failed runs, the mean fitness evaluations are 380 versus 2700.

#### E. Problems Where Constriction Wins

Fig. 7 shows the outcome of an experiment in which we evolved a problem where a constriction factor of 0.7 is beneficial. In 50 runs, a ten particle PSO with constriction always found a solution, taking on average 159 fitness evaluations. However, without constriction or a speed limit and a limit of 1000 generations, it only found an optimum in 37 of 50 runs. In the successful runs, on average, 900 evaluations were needed.

Often both PSOs took a similar amount of time. However, in many runs, the unconstrained swarm took much longer. Figs. 8–11 show other aspects of the first run where both constricted and unconstricted swarms find a solution. From this starting point with constriction, only 11 generations were needed (Fig. 8) and the concentration of the swarm in the promising region near the solution is clear. However, the

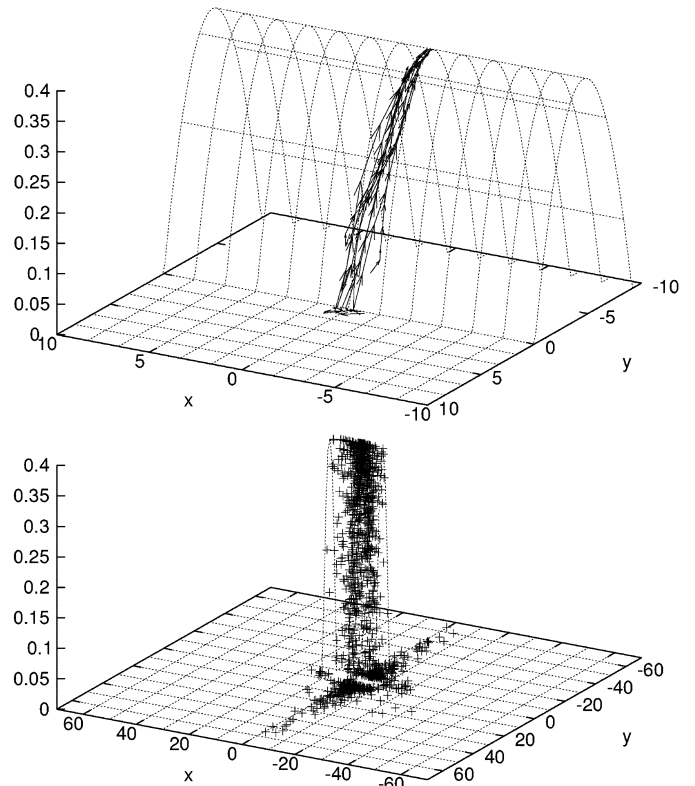


Fig. 7. GP (population 100) evolves a parabolic landscape  $-(0.171 + 0.0188y)y$ . With a constriction factor of 0.7, a ten particle swarm (upper diagram) finds an optimum in 11 generations. Lower diagram: Starting from the same initial conditions, without constriction the swarm explores more and so takes 128 generations. (Note change of scale).

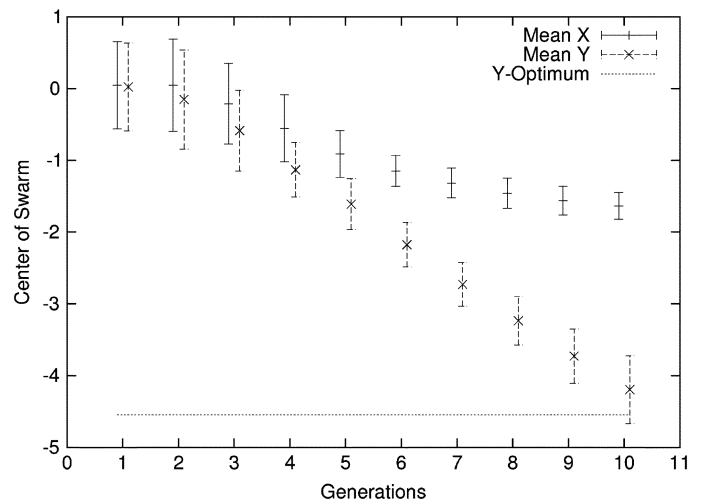


Fig. 8. The search progress for the ten particle PSO with a constriction coefficient of 0.7 on the  $-(0.171 + 0.0188y)y$  landscape of Fig. 7 (error bars show swarm spread, i.e., standard deviation of particles' positions). Note how the particles' position in the  $y$  dimension converges towards optimum.

unconstrained swarm oscillates for 128 generations before stumbling into an optimum (Fig. 9).

Looking at the kinetic energy of the swarm (i.e., sum of  $(1/2)v^2$  for each particle in the swarm) clearly differentiates the two cases. Fig. 11 shows without constriction the energy

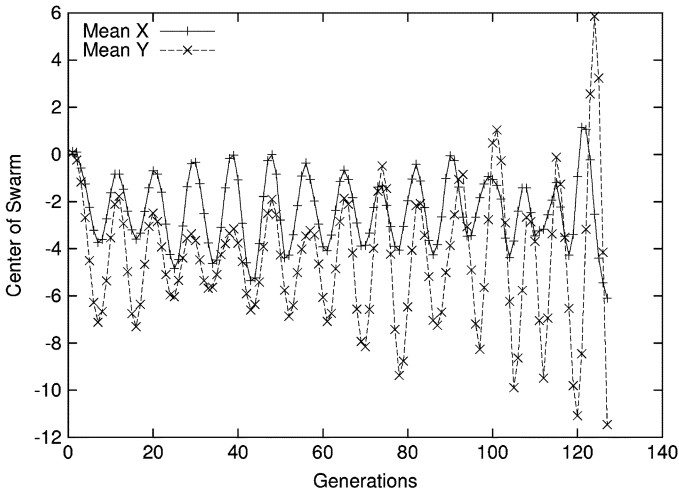


Fig. 9. The oscillating and increasing amplitude of the search made by ten particle PSO without constriction, etc.,  $-(0.171 + 0.0188y)$  of Fig. 7.

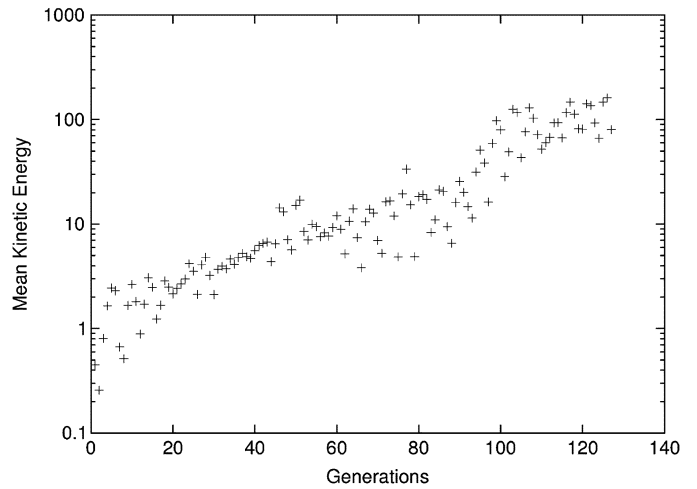


Fig. 11. The increasing kinetic energy of PSO swarm of ten particles without constriction on landscape  $-(0.171 + 0.0188y)$ . (Note log scale.)

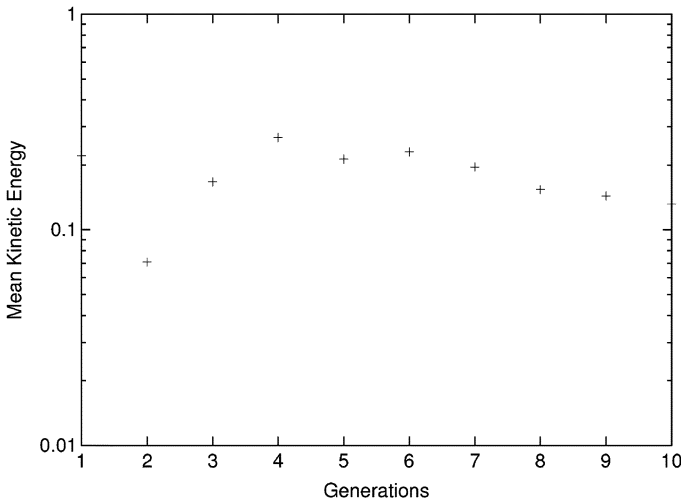


Fig. 10. Kinetic energy of PSO swarm of ten particles with constriction factor 0.7 on landscape  $-(0.171 + 0.0188y)$  of Fig. 7. (Note the logarithmic scale.)

increases exponentially, while Fig. 10 shows that, with a constriction factor  $\kappa$  of 0.7, energy falls exponentially.

This suggests 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 the search.*

**F. Problems Where Constriction Fails**

In runs where we were interested in finding fitness landscapes on which the use of a constriction factor was deleterious, a very successful evolved strategy was to put the optima some distance from where the fully connected swarm starts (see Fig. 12). The more energetic swarm is able to find an optimum, whereas constriction impedes the exploration which is needed. The balance between exploration and constriction is important. We can achieve a more energetic swarm by increasing the forces on the particles, e.g., if coefficients  $c_1$  and  $c_2$  are increased from 0.5 to 1.2, the PSO is able to climb to the peak. This shows that there are circumstances where *constriction can impede the search*

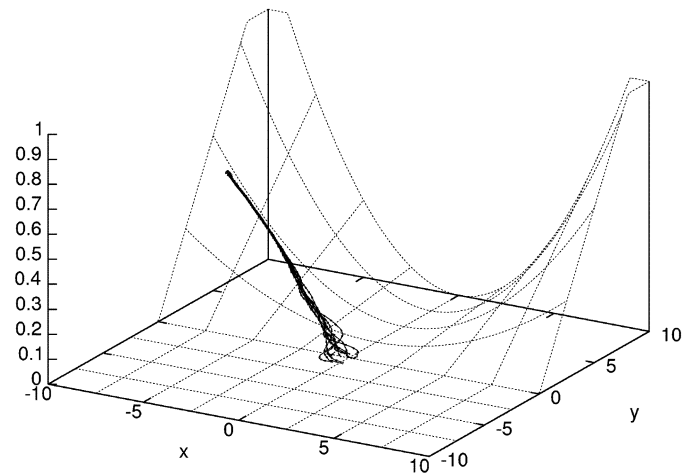
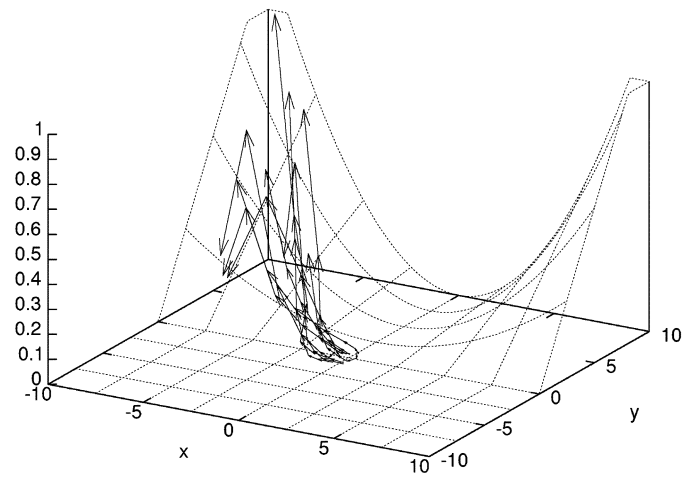


Fig. 12. Landscape  $0.00124x^2y$  evolved by GP. With no constriction factor or speed limit in a new run, a ten particle swarm (upper diagram) finds an optimum in eight generations. Lower diagram: Starting from the same initial conditions, the constricted swarm gets stuck despite the strong gradient. In 50 runs, the constricted PSO was never able to solve this problem (within 1000 generations).

*where the swarm needs to seek optima some distance from its starting locations.*

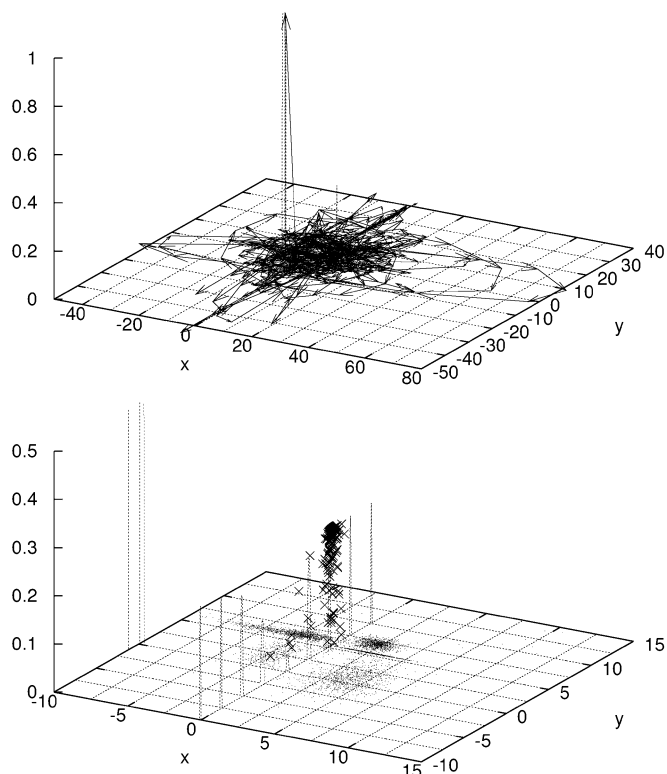


Fig. 13. Deceptive landscape  $x(0.643 + y - x(0.299x + 2.81 + y + y^2))$  evolved by GP. With no constriction factor or speed limit, a ten particle swarm (upper diagram) finds an optimum in 51 generations. Lower diagram: Starting from the same initial conditions (chosen at random in the range  $-10 \dots +10$ ), with constriction the swarm is less expansive (note change of scale). Once it finds a local hill, it explores it and never approaches the global optima.

In these experiments, GP went further and made the whole of the region where the swarms start flat. If all the points seen by the particles have the same fitness, none of them change the location of their “best point seen so far.” This also means the swarm best does not change, so the particles continue their (noisy) orbits around the same points. This is true both with and without constriction, however, with constriction, the orbits continually lose energy preventing the particles searching widely outside their start points. This is not true if there is no constriction and the particles’ orbits tend to expand taking them towards better parts of the search space. Thus, by exploiting the fixed initialization region, GP evolved landscapes which constricted swarms could never solve but were easy for swarms without friction.

The way in which an optimizer is started is obviously an important part of its search algorithm. To see if GP could find other cases where constriction is not helpful, we repeated the experiment but forced the optima to lie in the initialization region.

In these new experiments, GP (with a population of 1000) evolved the deceptive landscape given in Fig. 13. This contains rows of local optima near the origin well separated in both dimensions from a large region containing global optima. In 50 independent runs, without constriction or speed limit a ten particle swarm solves it 27 times, whereas it could only solve it six times with constriction. The upper diagram in Fig. 13 shows typical performance of a swarm without constriction, while the

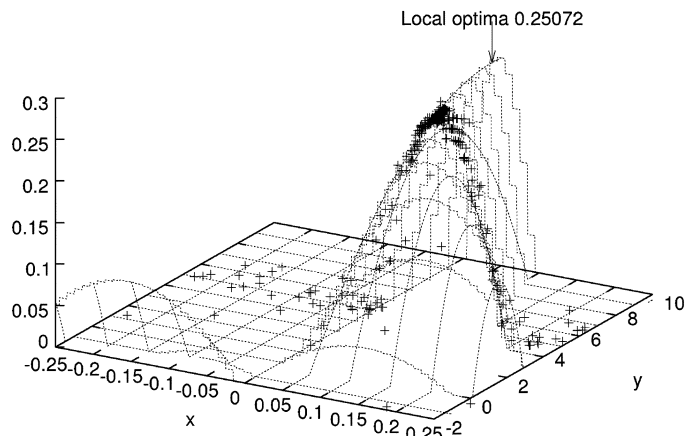


Fig. 14. Close up of lower part of Fig. 13 showing the position of ten swarm members with nontrivial fitness values up to generation 1000. Note how a constriction factor of 0.7 (no speed limit) narrows the search. Eventually, all but one particle are tightly bound together some distance from the local hill top which itself is far from the global optimum. The swarm becomes stuck. Even after 1000 generations, the problem is not solved.

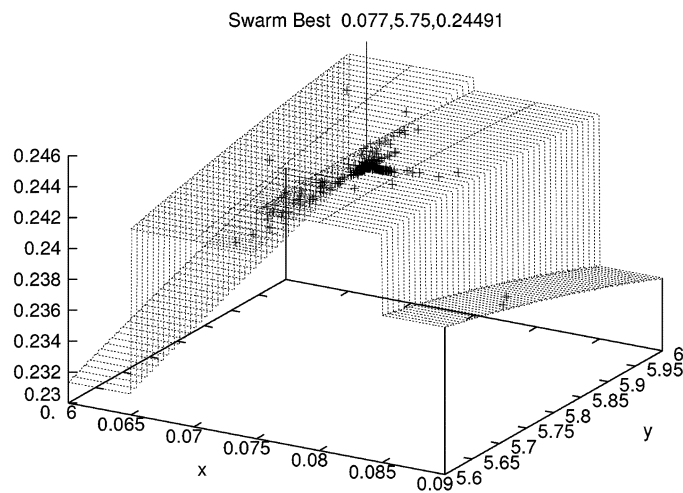


Fig. 15. Close up of lower part of Fig. 14 showing final position of swarm at the top of a parabolic region of fitness landscape. The adjacent parabolic region rises higher, but is lower in the swarm’s immediate neighborhood.

lower part (Figs. 14–19) shows the effects of a constant constriction factor of 0.7.

The unconstricted swarm searches widely and finds an optimum in only 51 update generations. In contrast, the lower diagram shows the performance of a swarm with a constriction factor of 0.7. The swarm becomes stuck and fails to find an optimum even after 1000 generations. In this run, the constricted swarm best improves quickly up to generation 27 but slows down thereafter. The swarm best does not move at all from between generations 98 and 591 when there are slight movements. No further improvements are found during generations 684–999.

Initially, most particles have zero fitness. However, the best is near the suboptimal peak. Each particle is attracted to it as well as its own (unchanged) best position. As the particles oscillate under the influence of both attractors, they tend to tumble into the small peak. The ninth finds the hill in generation 911 and



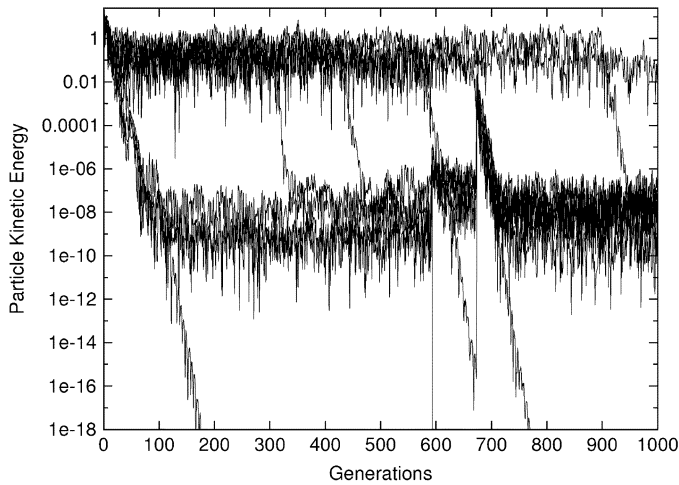


Fig. 16. Kinetic energy of swarm members for first run of a ten particle PSO with constriction on the landscape of Fig. 13. Initially, energy falls rapidly on average from 18 to 0.25 by generation 15. Then, there is a long period where, as individual particles converge towards the swarm best, their energy drops rapidly to a low value ( $\approx 10^{-8}$ ).

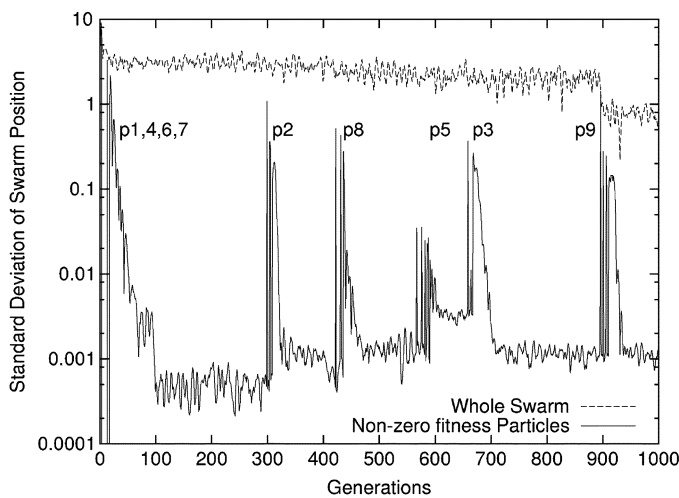


Fig. 17. Convergence of ten particle swarm with constriction on landscape of Fig. 13. Top line refers to whole swarm, while lower one refers to group with nonzero fitness. In this run, four particles (p1, p4, p6, and p7) get stuck on the small fitness hill by generation 28. The sudden increases in the spread of the fit members of the swarm corresponds to five other particles (p2, p8, p5, p3, and p9) finding nonzero fitness values, and joining the fit group. As they do so, they move their own best position seen close to the swarm best and so their oscillations lose energy (cf. Fig. 16). As the energy of the new member falls, it converges towards the other members of the fit cluster, so reducing its spread. The whole swarm (top line) never converges, since one particle never finds the local hill. The fit members of the swarm rapidly draw together so that they occupy a single  $0.01 \times 0.01$  tile.

quickly starts to lose energy, while one particle never finds a nonzero fitness.

Figs. 16 and 17 show that initially, the constriction factor causes the energy to fall rapidly and most of the swarm clusters together. When particles find nontrivial fitness values (near the swarm best), they update their local best and join the rest of the swarm. As they do so, the energy of their search falls dramatically. The bulk of the swarm converges to a single fitness value and most particle movements continue almost imperceptibly.

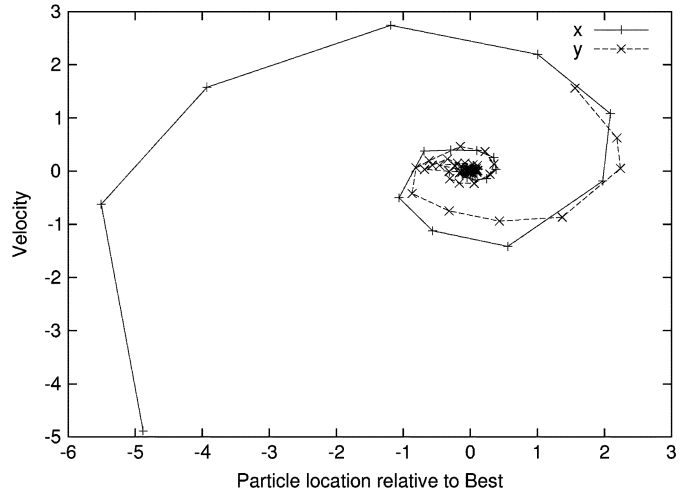


Fig. 18. Velocity (in both  $x$  and  $y$  dimensions) plotted against displacement from best location for particle 1 in a swarm with constriction on landscape of Fig. 13 (generations 2–1000). Note almost all the time this particle's personal best and the swarm best coincide. This gives rise to damped simple harmonic motion which is represented by spiraling inward motion in this graph.

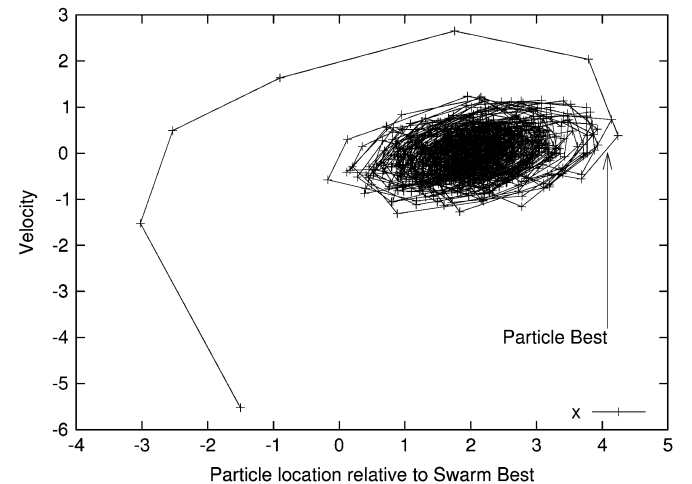


Fig. 19. Velocity against displacement from swarm best location for particle 0 on landscape of Fig. 13 (generations 2–1000). To reduce clutter, only velocity in the  $x$ -dimension is plotted. For the first 8–15 generations, the particle loses energy as it spirals inward. Once the particle spirals in between its own best and the swarm best (the origin), the random oscillations do not decay and the particle is kept in motion continuously despite the constriction factor.

Figs. 18 and 19 show two aspects of the constriction factor. Fig. 18 depicts the motion of particle 1. In almost all generations, it is the best particle in the swarm and its personal best is never very far from the swarm best. This means the random force towards the swarm best and that towards the personal best always point in the same direction. Thus, the PSO velocity update equation can be simplified. While motion of the particle is still noisy, it can now be approximated by assuming both random numbers used in the update are equal to their average value (i.e., a  $1/2$ ). Assuming continuous time, i.e., ignoring the discrete updates used by the PSO and using differential equations rather than differences, the equation can be solved. The solution is a damped simple harmonic motion about the best location. The decay constant is  $((1 - \kappa)/2)$  and the period is  $4\pi(4\kappa - (1 - \kappa)^2)^{-1/2}$  (7.6 for a constriction factor  $\kappa = 0.7$ ),

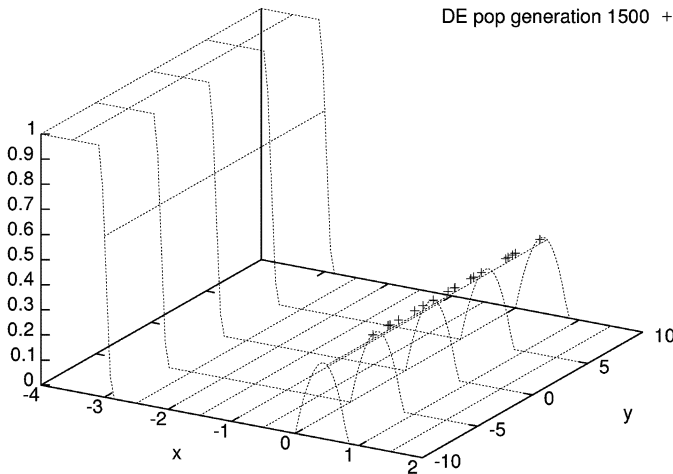


Fig. 20. Cubic  $(1.27 - 1.1x - 0.53x^2)x$  landscape where PSO (speed limit 10) outperforms DE. Both PSO and DE populations start near the origin, but in 40 out of 50 runs DE population find local optima at the small ridge and ignores the huge global optima in the opposite direction. The PSO always finds one of the best points.

while the decay time constant is 6.7. Referring to Fig. 18, we can see that about eight iterations are needed for each cycle of the spiral. After each improvement in the swarm best, both  $x$  and  $y$  dimensions converge exponentially with a decay constant of 0.15, as predicted, i.e., both the observed period and damping are close to predictions. Notice that the period and damping are the same in both  $x$  and  $y$  dimensions, so oscillations in both dimensions remain in step. In Fig. 18, this means the two spirals remain in step with each other.

When the particle best and swarm best do not coincide, the motion can be approximated by replacing the two points by their (randomly) weighted average, and we again get damped simple harmonic motion (with the same decay constant). However, the random weights used to form the average are changed every generation. Thus, this is only a reasonable approximation when the weighted average position does not move much compared with the motion of the particle, i.e., when the particle lies some distance from both the swarm best and the particle's personal best.

If the particle lies near or between both attractors, we can think of it as oscillating with the same period about some randomly chosen point between them for one time unit. However, at the next time unit, another random point will be chosen. This has the effect of keeping the particle in continuous motion. Fig. 19 looks at a particle where the swarm best and particle best are (approximately) the same distance apart through out the whole run. Consequently, the random forces in the PSO update equations keep it moving. Note the particle's kinetic energy depends on the distance between the swarm best and its own best. This explains why the kinetic energy of some particles decays rapidly to zero, while that of others is stable for long periods, cf. Fig. 16.

#### IV. COMPARISON OF DIFFERENT OPTIMIZERS

##### A. Problems Where a Velocity Limited PSO Beats DE

If we use velocity clamping, GP (with population size 1000) finds a landscape (see Fig. 20) which deceives DE. In 40 out of

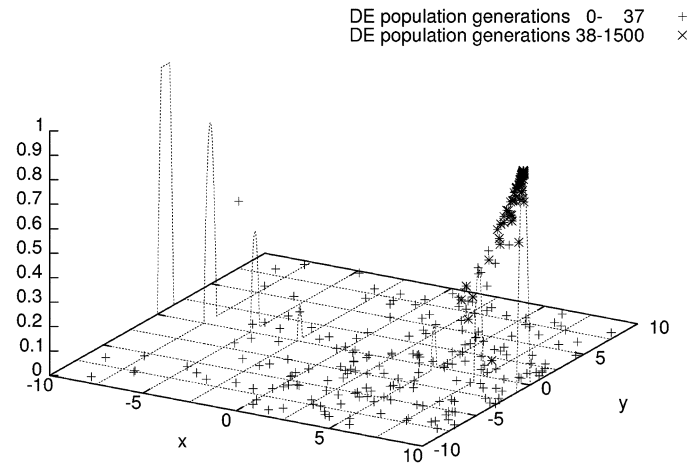


Fig. 21. Landscape  $(0.33 - 0.32x - 2.32y)y$  evolved by GP so that PSO outperforms DE. High fitness values are arranged along an inverted parabolic ridge, centred near the origin at about  $4^\circ$  to the  $y = 0$  line. Note that the end at  $x = -10$  is higher than that at  $x = +10$ . Both PSO and DE (+) initial populations are widely scattered ( $-10 \dots +10$ ). In this run, the DE population converges after generation 38 (x) onto the smaller peak and never finds the larger peak. In contrast, the PSO (maximum speed 10 but no constriction) being more stochastic always finds a global optimum.

50 runs, DE goes to a local optimum, while the PSO (starting from the same points) always finds the global optimum within 52 generations. (When DE succeeds, it is faster than the PSO).

This result is important because it shows that it *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” [25, p. 20]. This may be why Storn's WWW pages recommend that the initial population should be spread across the whole problem domain. This is also recommended in [7, p. 85]. The reasons for DE getting stuck may also be due to lack of movement opportunities. [34] and [35] calls this “stagnation.” However, [34] says “stagnation is more likely to occur” with “small population size ( $\ll 20$ ),” while we have observed slow movement with larger populations as well.

To avoid this bias against DE, in the remaining experiments (i.e., Sections IV-B to V), we extended the initial population to the whole  $-10 \dots +10$  region. (Except for CMA, we still use the same initial points for the PSO, DE, and the other optimizers. Also, our PSO is speed limited,  $|V| \leq 10$ , but constriction is not used in these final comparisons.) In some cases, GP needed a larger population (up to 1000) but in every case it managed to evolve a landscape which suited the desired optimizer in comparison with the other.

##### B. Problems Where PSO Beats DE

In experiments with these new settings, the best of run individual produced by the first GP run (with a population of 1000) had a high fitness on the training examples, but when the PSO and DE were rerun with new pseudorandom number seeds, they performed equally well on the evolved landscape. However, the landscape evolved in the second GP run did generalize. This is given in Fig. 21. Fig. 21 suits the PSO as it always finds a global optimum [peak near  $(-10, 1)$ ]. However, in 23 of 100 runs, the

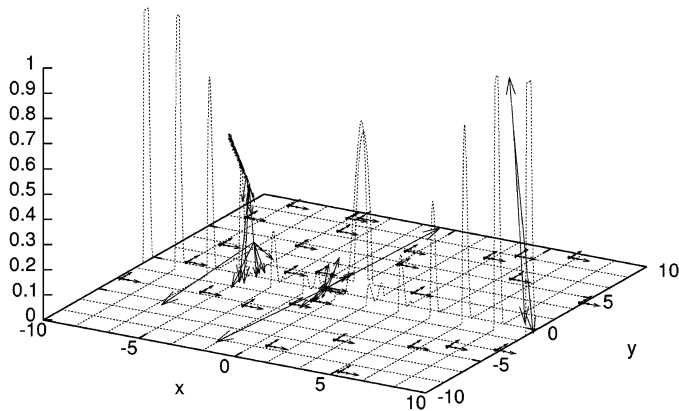


Fig. 22. Evolved landscape (GP population 100)  $y(0.093 + 0.39y + 0.15y^2 - 0.17y^3 - (0.19y^2 + 0.20y^3)x^2)$  showing points explored by N-R in the first run. Note arrows in the plane  $z = 0$ , where gradient search fails. However, after 32 restarts, N-R restarts near one of the global optima and quickly climbs to it (total 582 fitness evaluations). In contrast, the speed limited PSO (max speed = half width of feasible region) takes only 126 fitness evaluations and consistently outperforms N-R.

DE fails to find it. Fig. 21 shows a typical case where the DE population starts widely scattered but by generation 38 has converged on the wrong peak.

Fig. 21 shows PSO's more expansive search is more likely to find the global optima near  $(-10,0)$ , while DE tends to be less expansive than a PSO, its more directed search is deceived by a local optima, whereas (cf. Fig. 20) the PSO readily picks up the bigger signal issued by a large global optimum.

Again, this landscape is very instructive. DE may be deceived into converging on the wrong peak and, once there, it is impossible for it to escape. Note Storn's DE Java implementation follows [7, p. 86]'s recommendation and, after initialization, does not limit the search. Instead, the fitness function is effectively bounding DE's search to the  $-10 \dots +10$  box (the legal region) since its population never leaves it.

### C. Problems Where PSO Beats N-R

GP readily evolves a landscape where our particle swarm optimizer beats our N-R optimizer, see Fig. 22. In 50 runs (with new starting positions), PSO and N-R always solved the problem but our PSO significantly outperformed our N-R, on average evaluating 463 versus 1030 points.

This happens because approximately 95% of the search space has low fitness and is flat. N-R wastes many fitness evaluations where there is no gradient before giving up and restarting. In contrast, the problem is easily solved by PSO. Obviously, gradient search is inefficient where only a relatively small subset of the points in the search space have nonzero gradient.

### D. Problems Where PSO Beats CMA

The normalized landscape evolved on the first run is shown in Fig. 23. CMA does poorly, compared with our velocity limited PSO, because it picks up the low-frequency component of the search space which guides successive generations towards  $x = 10$ . Each time the CMA population gets stuck at  $x = 10$ , CMA restarts its ES with a bigger population. Eventually, the

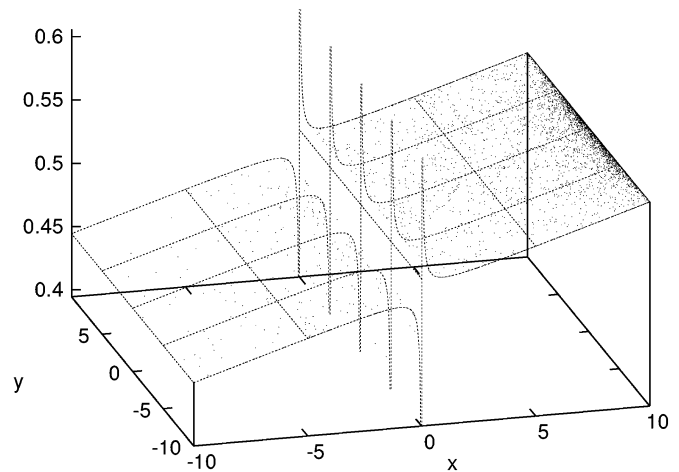


Fig. 23. 8681 points sampled by CMA on its first test run shown on the evolved landscape  $x - (x - 1)/x$  (GP population 10). On average, the velocity limited PSO needs 3000 and CMA 12000 evaluations. With small populations, CMA is lead by the sloping plane up to  $x = 10$  where it gets stuck, it then restarts with a bigger population. In the first run, CMA restarts eight times before the population is big enough to find the central spine by chance. In contrast, the PSO stumbles into it on generation 40 (1228 evaluations).

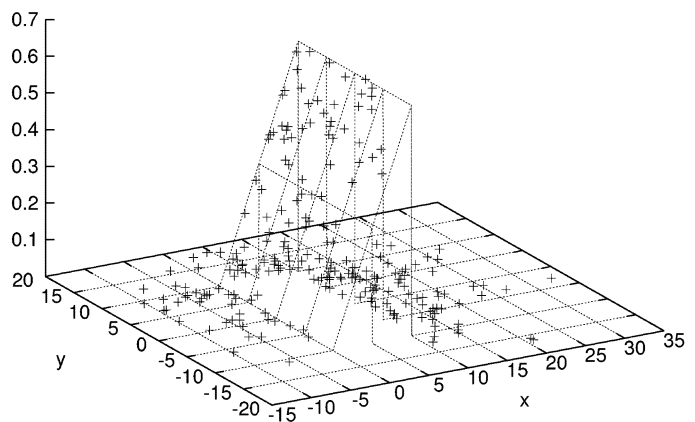


Fig. 24. Landscape  $0.063x$  evolved by GP (population 10), showing 230 points explored by first run of DE. In contrast, a speed limited PSO starting from the same points took 1829.

ES population is big enough to detect the small basin of attraction near the ridge line near  $x = 0$  and it quickly converges on a global value. In contrast, the PSO search is more stochastic. While CMA's restart strategy is robust, it may waste fitness evaluations before it chooses an appropriate population size.

### E. Problems Where DE Beats PSO

With a population of ten, GP evolved a landscape (see Fig. 24) which DE does consistently better than our PSO. In 50 runs, both DE and PSO (with a maximum speed of 10) always solved it, taking on average 400 versus 2100 evaluations.

Both PSO and DE find this type of landscape hard. The speed limited PSO without constriction finds it hard to home in on the narrow region of optima (cf. Section III-E). Notice also that DE finds this type of "cliff edge" landscape hard because the gradient on one side continuously leads it to overshoot the global optimum, cf. also Section IV-I. However, unlike Figs. 20 and 21, there is a narrow global optimum (occupying 0.05%

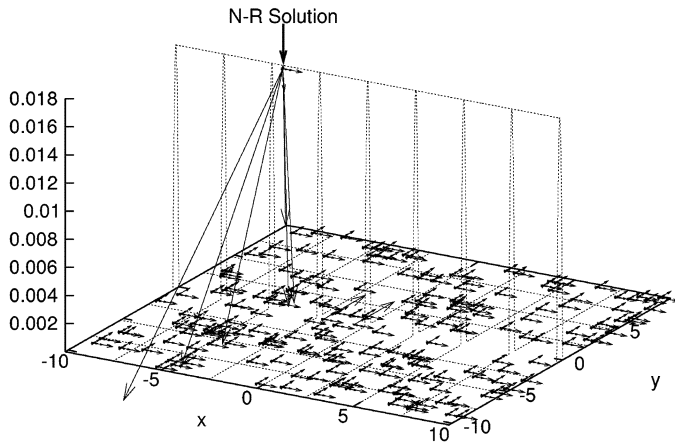


Fig. 25. Landscape  $-(0.13 + 0.24y)y$  evolved by GP (population 10). High fitness points lie in a line at  $y = 0.27$ . Arrows indicate 6733 points explored by first run of N-R gradient follower before it found a solution. In contrast, DE starting from the same points took 1024. N-R is forced to restart 211 times before starting close enough to a solution to find it. Note: 1) large flat area where both do badly and 2) optimum is 0.017, which is very different from 1.0. This causes N-R to overestimate distance to it (long diagonal arrows) but does not affect DE. (To reduce clutter, the 16 cases where N-R started near the high fitness region but was unsuccessful are not plotted).

of the feasible search space). This target proves to be quite small for our PSO, causing it to take on average 2100 fitness evaluations to find it. This shows a weakness of the PSO: the particles are unable to home in on “narrow” global optima. Schoenauer and Michalewicz [36] have suggested that global optima of 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 and PSOs might not perform well on constrained optimization problems*.

#### F. Problems Where DE Beats N-R

GP (with a population of 10) readily finds a parabolic landscape where it consistently beats gradient search, see Fig. 25. In 50 runs, both DE and N-R always find solutions but on average DE takes 770 fitness evaluations versus 3600 for N-R. Remember, the gradient follower assumes the optimum value will be near 1.0, GP has exploited this and set it at only  $1/60$ . When N-R uses the local gradient to estimate the location of the optimum, this causes it to massively overestimate the distance it should move. N-R only solves the problem when it starts very near a solution. Note that if we rescale the landscape so that the optimum is 1.0, then on average N-R needs only 620 samples and will beat DE.

All search heuristics must make assumptions about their search space. GP has turned N-R’s assumption, that the optima are near unity, against it. By rescaling the landscape, GP has further strengthen DE’s advantage. In the experiments with CMA, we prevent GP doing this by normalizing the evolved landscape by linearly rescaling so that the largest value is always one.

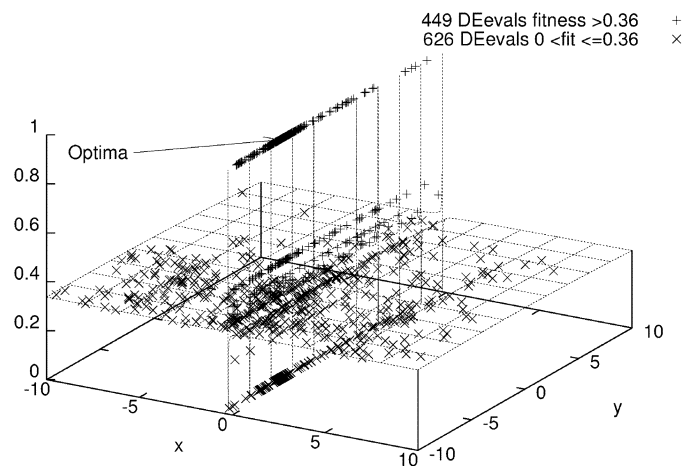


Fig. 26. Landscape evolved by GP (population 100). DE is able to converge on a narrow region near the optimum, e.g., in the first test run, 35% of DE sample points (+) lie near  $x = -0.23$ . However, CMA converges towards  $(-10, -5)$  far from the optimum. (x shows the 49% of points sampled by DE that lie in or below the nearly flat ( $z \approx 0.33$ ) region, while 16% of DE samples lie outside the feasible region and are not shown).

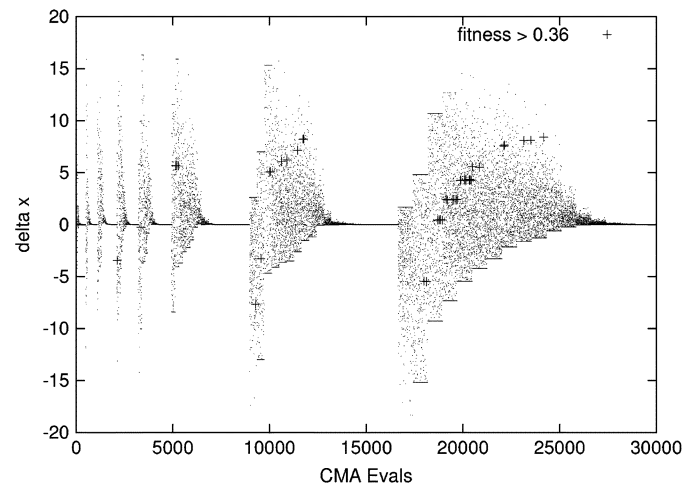


Fig. 27. Strength of mutation in first CMA test on evolved landscape of Fig. 26. Step size falls towards end of each CMA run (CMA restarts eight times). 43 high fitness values are randomly sampled but CMA is unable to converge on them because: 1) single values have little impact on CMA population mean and 2) in the next generation, mutation is still strong and creates offspring some distance away. (Since CMA, but not DE, knows the  $-10 \dots +10$  boundaries, this causes the generation steps, particularly visible on lower right).

#### G. Problems Where DE Beats CMA

DE solved the normalized problem shown in Fig. 26 86 times out of 100, but CMA found the optimum in only 14. Again (cf. Fig. 23), a ridge proves too narrow to attract CMA and most of the population ends near  $x = -10$ .

Fig. 27 shows, in a typical run, even when CMA samples close to the ridge, this is near the start of the run, and therefore the mutation steps in the next generation are large and carry the ES individuals far from the region ( $-2 \lesssim x < -0.23$ ) leading to the optimum. In contrast, DE’s population hugs this region. *CMA’s ability to average over the population makes it robust to noise but less able to respond to outliers*.

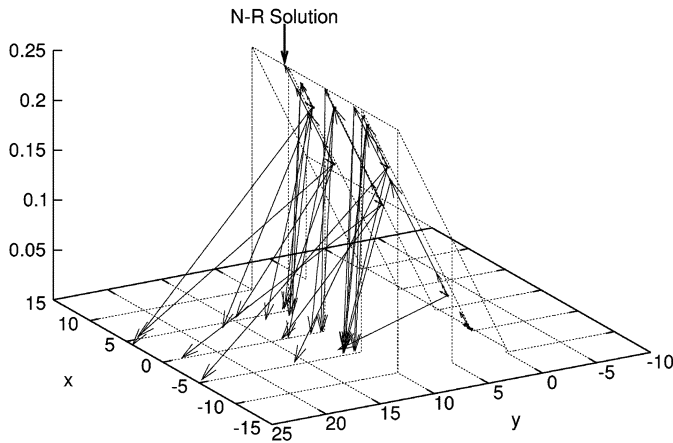


Fig. 28. Evolved landscape  $0.0043 + 0.024y$  showing 146 points explored by N-R in the first test run. The plot has been rotated to show the large number of cases where N-R overshoots (the optima are less than 1.0). However, they are closer to 1.0 than in Fig. 25 and the landscape is simpler, so our gradient follower does better. With a speed limit of 10, our PSO has problems since optima occupy only 0.05% of the feasible region of the search space. In this run, using the same starting positions at the N-R, the PSO samples 412 points before finding an optimum.

H. Problems Where N-R Beats PSO

With a population of 100, GP evolved the landscape shown in Fig. 28. In 50 runs, using starting points chosen independently from those used by the GP, both our gradient-based searcher and PSO always found a solution. However, on average, N-R took 450 landscape samples compared with 2400 for the PSO.

Due to the cliff edge top of the landscape, the optima occupy only 0.05% of the feasible region. Although the PSO samples points near the optimum very quickly, the particles' energy tends to increase (cf. Section III-E) and the swarm becomes increasingly erratic. Note that even though the swarm best is stable, without constriction, friction, or position limiting, it is not sufficient to keep the swarm near the optimal region. Therefore, as time progresses, our PSO searches ever wider on this landscape, i.e., the swarm samples points further and further from the optimum, before eventually tumbling into an 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 most other population-based search algorithms, like a GA, which tend to focus, rather than expand, their search in later generations.

I. Problems Where N-R Beats DE

With a population of 1000, GP evolved the parabolic problem landscape shown in Figs. 29 and 30 in which N-R does better than DE. In 50 runs, N-R always found a solution, while DE failed nine times. N-R took on average 120 evaluations, while DE required 550 (on its 41 successful runs).

The bimodal nature of the landscape means both optimizers are quite likely to head towards the lower ridge line (at  $y = -10, z = 0.717$ ). 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 restarts from another

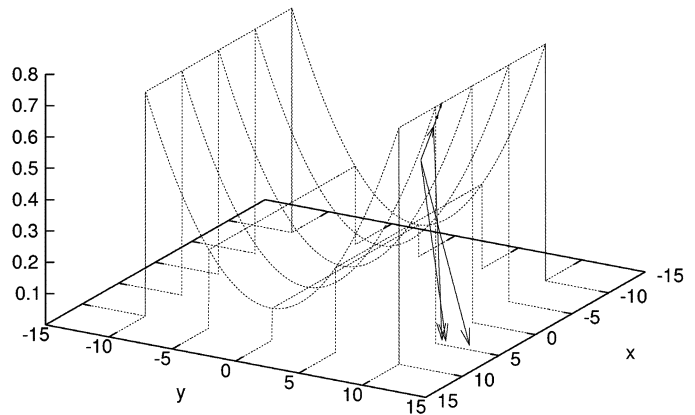


Fig. 29. Run 0 of gradient-based optimizer, showing movement of N-R on  $0.102 + 0.00189y + 0.00635y^2$  landscape. Initially, N-R uses too large a step size, which causes it to test outside the feasible region. N-R reduces its step size and finds an optimum in 28 fitness evaluations, cf. Fig. 2.

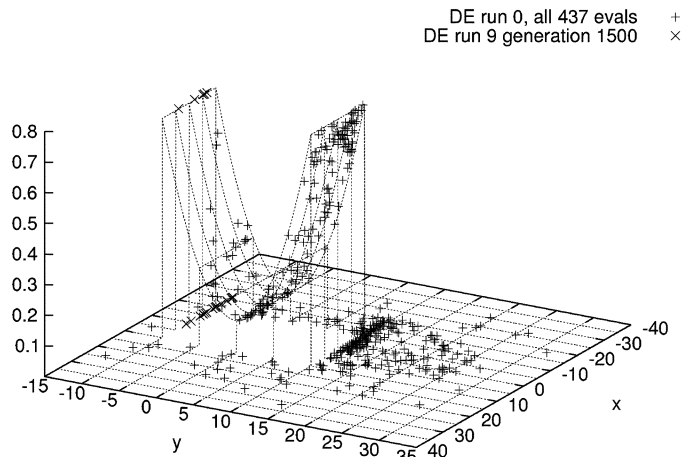


Fig. 30. A successful (+) and a nonsuccessful (x) DE run, on same landscape as Fig. 29. In the successful run, DE evaluated 437 points. In the other run, DE ascends the slightly lower hill and gets stuck at the top of it, x shows the points evaluated during the last generation. Notice the gradient leads the DE population to jump lemming like past the optima at the cliff edge, so many test points have zero fitness.

random position. Restarting virtually guarantees N-R will find the right ridge ( $y = 10, z = 0.755$ ).

The evolved landscape is smooth, allowing gradient-based search to reach the optima more quickly than population search. However, GP has reinforced this advantage by making the problem multimodal. This benefits N-R, since it rapidly restarts on reaching a local optimum, while DE may be deceived into heading in the wrong direction and does not restart.

Again, this landscape (Figs. 29 and 30) is very interesting. It emphasizes 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 nonglobal 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 neighborhood 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

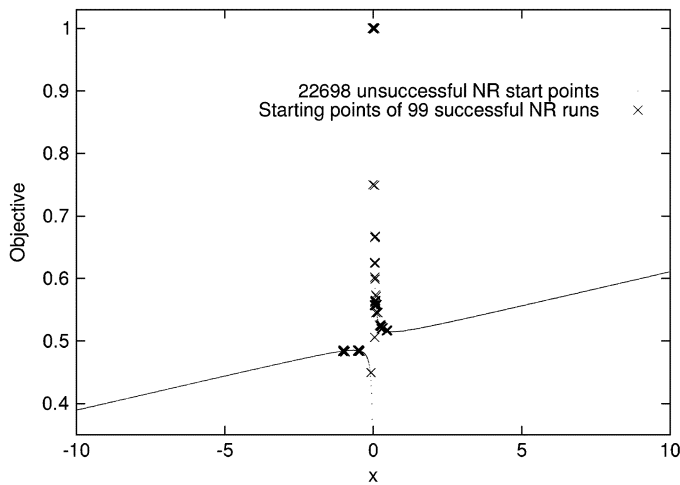


Fig. 31. Normalized evolved landscape  $2x + 0.91/x$  ( $GP \text{ pop} = 10$ ), which favors N-R against CMA. Even N-R finds it hard. Successful N-R starting points  $\times$  are mostly in the narrow region above the ridge (at  $x = 0$ ), where the gradient is negative. There are also isolated regions near  $-1$  and  $-1/2$ , where the distance to the ridge meshes nicely with N-R's heuristics. As with Figs. 23 and 26, CMA follows the low-frequency gradient and its populations tend to get stuck at  $x = 10$  causing repeated restarts.

fail. In fact, in deceptive problems, where population based algorithms perform badly, nonglobal optima have much bigger basins of attraction than that of the global optimum. This example shows that *GP* has automatically discovered that the notion of deception applies to DE.

#### J. Problems Where N-R Beats CMA

Once more GP has chosen a narrow ridge to allow our gradient following hill-climber to beat CMA. As Fig. 31 shows, the normalized landscape contains a ridge which is also difficult for N-R with only 1 in 230 starting points leading to an optimum. Nevertheless, N-R decides more quickly than CMA if it needs to restart. So N-R takes on average 6000 fitness evaluations versus 8000 for CMA, excluding the 10% of cases where CMA failed to find a solution before using 30 000 fitness evaluations. (N-R failed once in 200 test runs.)

#### K. Problems Where CMA Beats PSO

CMA has no difficulty exploiting its knowledge of the legal boundaries to beat our PSO. On average, it takes only 21 fitness evaluations to find the unique global optimum to the normalized problem shown in Fig. 32, while our PSO never found it in 100 runs. However, the PSO does get very close. On average, it comes within 0.003 of maximum fitness (0.002–0.005 quartiles). We also tried increasing the coefficients of the forces towards the previous bests from 0.5 to 1.2, cf. Table II. Despite running our PSO on the  $x - y$  problem 100 times, neither the t-test nor the sign test showed that  $c=1.2$  improved performance. Further strengthening the generality of results.

CMA finds the problem easy because GP has put the global optimum in a corner. If mutation generates any point lying in  $x \geq 10$  and  $y \geq 10$ , the boundary conditions will force it to  $x = 10, y = 10$ , which is where GP has put the solution! Obviously, this is unfair, but exactly the same conditions were used in Section IV-D when PSO defeated CMA. Also, if we

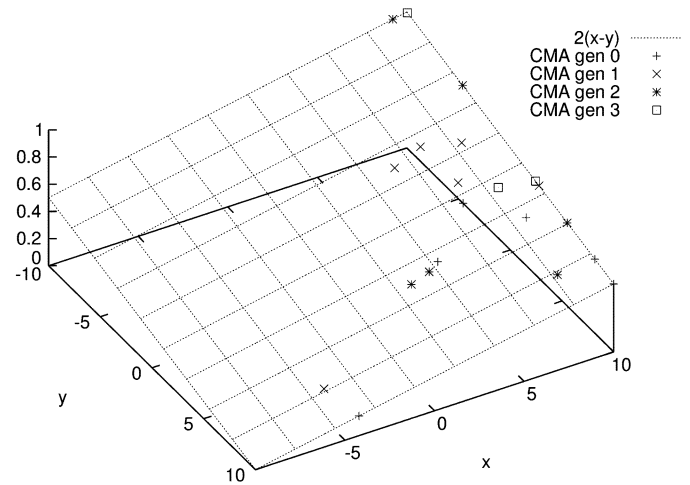


Fig. 32. Normalized evolved landscape  $x - y$  ( $GP \text{ pop} = 10$ ) which favors CMA over a PSO. Points show the CMA population in a typical run.

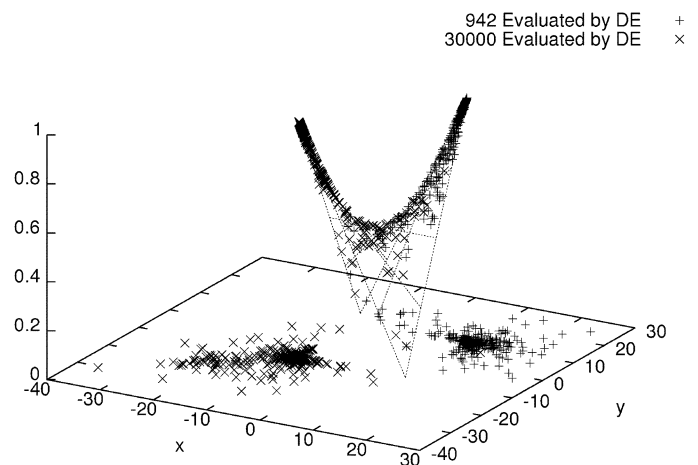


Fig. 33. Normalized evolved landscape  $(1.03 + 2.81x)y$  ( $GP \text{ pop} = 10$ ) which favors CMA over DE. In the first successful DE test run,  $+$  shows that DE converges on the optimum. In the second test run ( $\times$ ), DE converges on the slightly lower corner and never finds the solution. In both cases, DE scatters many test points beyond the local peak.

do not help CMA but instead allow it to search everywhere, it still beats our PSO and the mean number of evaluations only rises from 12 to 442. The optimum occupies only  $2.5 \cdot 10^7$  of the search space. It is too small for our velocity limited *gbest* PSO swarm to locate.

#### L. Problems Where CMA Beats DE

CMA has absolutely no difficulty in doing better than DE on the normalized evolved landscape shown in Fig. 33. CMA takes only 35 fitness evaluations (35 median, 14–282 quartiles) to find the optimum. Notice how GP has played to CMA strengths by placing the optima at one corner but has made things hard for DE. DE finds  $(1.03+2.81x)y$  difficult because: first, there is a local optima in the opposite direction from the global peak, and second, because the optimum is on a “cliff edge.” In 41 of 100 test runs, DE failed. In the other 59, DE took 1134 fitness evaluations (1134 median, 1002–1245 quartiles). See Sections IV-E and IV-I for other examples of DE and cliff edges and DE being deceived. Even if we remove CMA's “unfair” advantage and

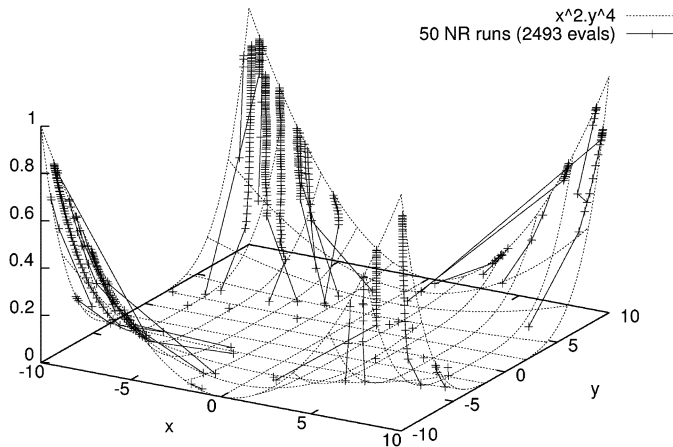


Fig. 34. Normalized evolved landscape  $x^2y^4$  (GP pop = 10) which favors CMA over a N-R. Points show first 2493 evaluations in a typical N-R run.

allow it to test outside the  $-10 \dots 10$  square, it still beats DE but it is slowed down (new median evaluation is 561, 328–1000 quartiles).

#### M. Problems Where CMA Beats N-R

In the normalized evolved landscape  $x^2y^4$  (shown in Fig. 34), GP has placed an optimum in each corner. (As noted in the previous section, when CMA is told the boundaries of the feasible region, it is especially suited to finding solutions in corners.) On average, CMA takes only 12 evaluations to find one. In contrast, N-R succeeds within 30 000 evaluations, only in 13 of 100 runs. (Even if we remove this advantage, CMA still beats N-R and solves the problem 100% of the time and the mean number of fitness evaluations only rises from 12 to 462.)

Our N-R assumes the problem is smooth and takes samples near its present location to estimate the local gradient. Except in the flat region (cf. Section IV-C) in the middle of  $x^2y^4$ , this works well and initially the gradient follower climbs rapidly. However, the concave fitness landscape causes N-R to repeatedly overestimate the distance to 1.0, causing it to reduce its step size and take more fitness evaluations (cf. Section IV-I). Moreover, when N-R nears the edges, local samples are drawn from outside of the feasible region and the estimate of the gradient becomes unreliable. This causes N-R to restart. Only if the ascent remains away from the edges (i.e., towards the diagonals) can N-R reach one of the corners. As Fig. 34 shows, *N-R performs poorly where solutions are near discontinuous regions.*

#### V. PSO VERSUS CMA IN MANY DIMENSIONS

Finally, we extended the benchmark evolved in Section IV-D to many dimensions. The objective value in high dimensions is the sum of the objective values for one dimension. This is linearly rescaled so that the global optima remain 1.0. This extends the essential features of the landscape shown in Fig. 23. That is, there is a plane, formed by  $\sum x_i$ , which guides the search towards  $(10, 10, 10, \dots)$  and away from the global optima, which are narrow spines running parallel to each axis near  $x_i = 0$ .

In Fig. 35, we show the performance of PSO and CMA after the fitness landscape has been extended up to 51 dimensions. While there is some variation (particularly for smaller problems)

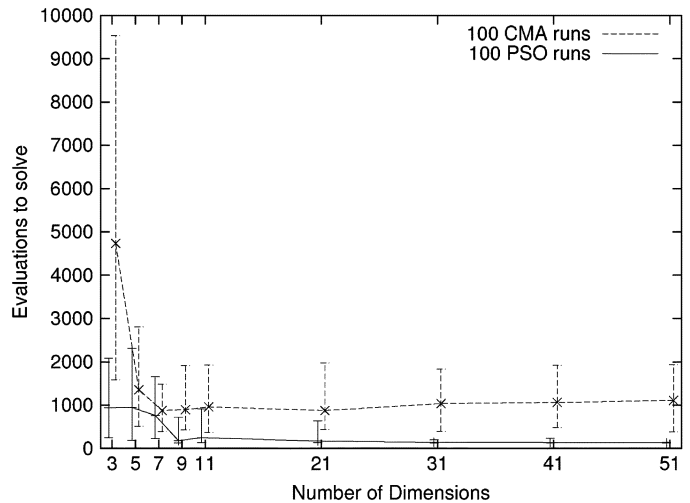


Fig. 35. Median fitness evaluations to solve  $\sum_{i=1}^n x_i - (x_i - 1)/x_i^n$  by 100 PSO and CMA runs ( $n$  must be odd). Note  $x_i - (x_i - 1)/x_i^n$  is scaled to ensure the size of both the basin of attraction and the global optima as fraction of the search space remain the same as the number of dimensions changes. Both PSO and CMA are forced to search only in the  $(-10 \dots +10)^n$  hypercube. Error bars indicate lower and upper quartiles.

for five or more dimensions, the performance of both CMA and PSO is fairly consistent. *The lesson from two dimensions (Section IV-D) applies to higher dimensions.*

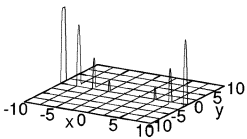
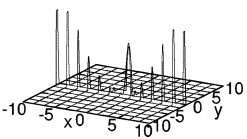
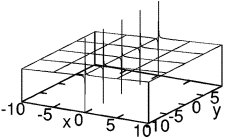
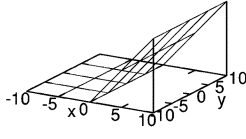

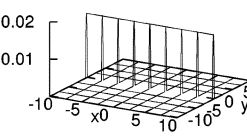
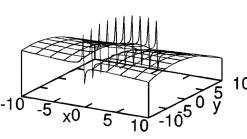
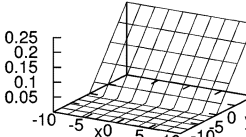
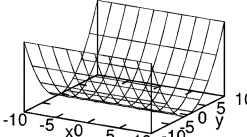

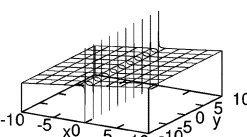
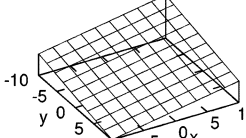
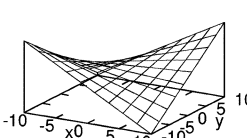
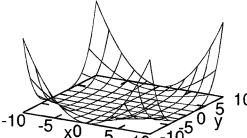
#### VI. DISCUSSION

The results of Section IV are summarized in Table III. Rather than using predefined benchmarks, it proved easy for GP to find simple landscapes where each of four very different optimizers beats its opponent. We looked at all pairwise comparisons. We also (especially for the case of particle swarms) compared different parameter settings of the same optimizer. Again, GP found landscapes which suited particular parameter settings. *In every case*, GP succeeded in finding a landscape which suited the technique (or parameter setting) over the other technique and *vice versa*. Note that not only does an alternative landscape exist, which the No Free Lunch [17] results assure us must exist in theory, but an example could be readily found.

Where the first GP run did not find a landscape which reliably separated the two optimizers, we increase the population by a factor of 10 to 100 or 1000. Only once (cf. Section IV-B) did we need to run a GP with a population of a 1000 individuals again. Each landscape shown has been examined to prove the differences are indeed statistically significant.

Run time depends heavily on a number of factors. These include the computer used and which of the optimizers are being compared, their population sizes (1, 20, 30, etc.), the number of generations they are allowed (up to 1500), the size of the GP population (10, 100, or 1000), and the number of GP generations (10). (Figures in brackets indicate values used in Sections III–V.) 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 3 GHz PC. The longest run with a population 100 times bigger took 53 hours. Doubtlessly, if need arose, these times could be greatly reduced by suitable code optimization and/or parameter tuning.

TABLE III  
MEAN FITNESS EVALUATIONS IN OPTIMIZER VERSUS OPTIMIZER EVOLVED LANDSCAPES

	Particle Swarm Optimisation (PSO)	Differential Evolution (DE)	Newton-Raphson (N-R)	Evolutionary Strategy (CMA)
PSO	See Section III	 $(0.33 - 0.32x - 2.32y)y$ 700 PSO v. 7200 DE IV-B	 $y(0.093 + 0.39y + 0.15y^2 - 0.17y^3 - (0.19y^2 + 0.20y^3)x^2)$ 460 PSO v. 1000 N-R IV-C	 $x - (x - 1)/x$ 3 000 PSO v. 12 000 CMA Sections IV-D and V
DE	 $0.063x$ 400 DE v. 2 100 PSO IV-E	 $-(0.13 + 0.24y)y$ 770 DE v. 3 600 N-R IV-F	 $(0.56x - 1.4y - 14 - 28((1/y)/x + 0.23)y)$ 2 700 DE v. 27 000 CMA IV-G	
N-R	 $0.0043 + 0.024y$ 450 N-R v. 2 400 PSO IV-H	 $0.102 + 0.00189y + 0.00635y^2$ , 120 N-R v. 5 900 DE IV-I		 $2x + 0.91/x$ 6 100 N-R v. 11 000 CMA IV-J
CMA	 $x - y$ 21 CMA v. not solved PSO IV-K	 $(1.03 + 2.81x)y$ 190 CMA v. 13 000 DE IV-L	 $x^2y^4$ 12 CMA v. 28 000 N-R IV-M	

Real-world problems typically contain many dimensions and so practical optimizers, such as those we have used, must deal with far more than two dimensions. However, it is common for such algorithms to be subdivided into parts which deal with one dimension at a time. Both PSO and DE do this, but then bring together dimensions (e.g., in the inner  $j$  loop in Table II our PSO deals with each dimension separately. Then, the following code ( $i$  loop) deals with all dimensions.) Therefore, we suggest that in many cases lessons learnt in lower dimensional problems can, with appropriate caution, be used in higher dimensions. Indeed, none of the lessons highlighted in Sections III and IV are specific to the low dimensionality used in experiments. In Section V, we describe an experiment to start confirming this.

We are greatly encouraged by the success of this new technique. There are many ways this work could be extended. For example, by considering other types of optimizers. We have used it with one algorithm with time-varying parameters (CMA), it could be used with other algorithms which adapt their parameters (e.g., population size) as they run, such as certain types of PSO. There are many other types of more sophisticated PSOs [37] (e.g., CPSO [38], UPSO [39], CLPSO [40], FDR-PSO [41], HPSO-TVAC [42]), and differential evolution (e.g., SaDE [43] and SADE [44]) where our technique might be used. Extensions to investigate constrained optimization or multiobjective optimization techniques (e.g., GDE [45]–[47])

might be needed. However, we have established the viability of using GP to devise test problems for continuous optimization problems and using them as tools to analyze real-world optimization algorithms.

## VII. CONCLUSION

Theoretic analysis of evolutionary algorithms, in general, and particle swarm optimizers, DE and CMA-ES, 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 that GP, by forcing alternative techniques to compete inside a single computer (rather than scattered across the pages of diverse conferences and journals), can readily produce fitness functions which illustrate their comparative strengths and weaknesses, cf. Table III.

## ACKNOWLEDGMENT

We would like to thank J. Kennedy, M. Clerc, M. Oltean, H.-G. Beyer, C. Stephens, T. Krink, O. Holland, C. Di Chio, A. Kucerova, and N. Hansen for papers, helpful discussions, and suggestions. We would like to thank R. Storn and N. Hansen for the use of their DE and CMA-ES Java code.



## REFERENCES

- [1] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Mateo, CA: Morgan Kaufmann, 2001.
- [2] J. Kennedy, "The behavior of particles," in *Proc. 7th Ann. Conf. Evol. Program.*, San Diego, CA, 1998, pp. 581–589.
- [3] E. Ozcan and C. K. Mohan, "Particle swarm optimization: Surfing the waves," in *Proc. Congr. Evol. Comput.*, P. J. Angelino, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzal, Eds., Washington, D.C., Jul. 6–9, 1999, vol. 3, pp. 1939–1944.
- [4] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.* vol. 6, no. 1, pp. 58–73, Feb. 2002. [Online]. Available: [http://ieeexplore.ieee.org/xpl/abs\\_free.jsp?arnumber=985692](http://ieeexplore.ieee.org/xpl/abs_free.jsp?arnumber=985692)
- [5] F. van den Bergh, "An analysis of particle swarm optimizers" Ph.D. dissertation, Dept. Comput. Sci., Univ. Pretoria, Pretoria, South Africa, Nov. 2001. [Online]. Available: [http://www.cs.up.ac.za/cs/fvdbergh/publications/phd\\_thesis.ps.gz](http://www.cs.up.ac.za/cs/fvdbergh/publications/phd_thesis.ps.gz)
- [6] R. Storn and K. Price, "Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces" *Int. Comput. Sci. Inst., Berkeley, CA, Tech. Rep. TR-95-012*, Mar. 1995. [Online]. Available: <ftp://ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.pdf>
- [7] K. V. Price, "An introduction to differential evolution," in *New Ideas in Optimization*, ser. Maidenhead, Advanced Topics in Computer Science, D. Corne, M. Dorigo, and F. Glover, Eds. Berkshire, U.K.: McGraw-Hill, 1999, ch. 6, pp. 79–108.
- [8] R. Storn, "Designing digital filters with differential evolution," in *New Ideas in Optimization*, ser. Maidenhead, Advanced Topics in Computer Science, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999, ch. 7, pp. 109–125.
- [9] J. Lampinen and I. Zelinka, "Mechanical engineering design optimization by differential evolution," in *New Ideas in Optimization*, ser. Maidenhead, Advanced Topics in Computer Science, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999, ch. 8, pp. 127–146.
- [10] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proc. 9th Int. Conf. Soft Comput. MENDEL*, R. Matousek and P. Osmera, Eds., Brno, Czech Republic, Jun. 2003, pp. 41–46. [Online]. Available: <http://www.info.uvt.ro/~dzaharie/mendel03.pdf>
- [11] A. Ostermeier, A. Gawelczyk, and N. Hansen, "A derandomized approach to self-adaptation of evolution strategies," *Evol. Comput.* vol. 2, no. 4, pp. 369–380, 1995. [Online]. Available: <http://www.bionik.tu-berlin.de/user/niko/derapproaEc.pdf>
- [12] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.* vol. 11, no. 1, pp. 1–18, Spring 2003. [Online]. Available: [http://mitpress.mit.edu/journals/pdf/evco\\_11\\_1\\_1\\_0.pdf](http://mitpress.mit.edu/journals/pdf/evco_11_1_1_0.pdf)
- [13] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *Proc. IEEE Congr. Evol. Comput.*, D. Corne, Z. Michalewicz, B. McKay, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, G. Raidl, K. C. Tan, and A. Zalzal, Eds., Edinburgh, U.K., Sep. 2–5, 2005, vol. 2, pp. 1769–1776. [Online]. Available: <http://ieeexplore.ieee.org/servlet/opac?punumber=10417&isvol=2>
- [14] N. Hansen, "The CMA evolution strategy: A tutorial," Nov. 11, 2005. [Online]. Available: <http://www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf>
- [15] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," *Tech. Rep.*, Nanyang Technological Univ., Singapore and Kanpur Genetic Algorithms Laboratory, IIT, Kanpur, KanGAL Rep. 2005005, May 2005. [Online]. Available: [http://www.ntu.edu.sg/home/epn\\_sugan/index\\_files/CEC-05/Tech-Report-May-30-05.pdf](http://www.ntu.edu.sg/home/epn_sugan/index_files/CEC-05/Tech-Report-May-30-05.pdf)
- [16] N. Hansen, "Compilation of results on the CEC benchmark function set," *Inst. Comput. Sci., ETH Zurich, Switzerland, Tech. Rep.*, 13, Sep. 2005. [Online]. Available: [http://www.ntu.edu.sg/home/epn\\_sugan/index\\_files/CEC-05/compareresults.pdf](http://www.ntu.edu.sg/home/epn_sugan/index_files/CEC-05/compareresults.pdf)
- [17] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
- [18] M. Oltean, "Searching for a practical evidence for the no free lunch theorems," in *Proc. 1st Int. Workshop Biologically Inspired Approaches to Advanced Inf. Technol.*, A. J. Ijspeert, M. Murata, and N. Wakamiya, Eds., Lausanne, Switzerland, Jan. 29–30, 2004, vol. 3141, pp. 472–483. [Online]. Available: [http://www.cs.ubb-cluj.ro/~moltean/oltean\\_biodit\\_springer2004.pdf](http://www.cs.ubb-cluj.ro/~moltean/oltean_biodit_springer2004.pdf), ser. LNCS, revised Selected Papers.
- [19] J. I. van Hemert, "Evolving binary constraint satisfaction problem instances that are difficult to solve," in *Proc. Congr. Evol. Comput.*, R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, Eds., Canberra, Dec. 8–12, 2003, pp. 1267–1273.
- [20] B. Edmonds, "Meta-genetic programming: Co-evolving the operators of variation," *Elektrik* vol. 9, no. 1, pp. 13–29, May 2001. [Online]. Available: <http://cogprints.ecs.soton.ac.uk/archive/00001776/>, Turkish J. Elec. Eng. Comput. Sci.
- [21] R. Poli, W. B. Langdon, and O. Holland, "Extending particle swarm optimization via genetic programming," in *Proc. 8th Eur. Conf. Genetic Program.*, M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, Eds., Lausanne, Switzerland, Mar.–Apr. 30–1, 2005, vol. 3447, pp. 291–300. [Online]. Available: <http://www.cs.essex.ac.uk/staff/poli/papers/eurogpPSO2005.pdf>, ser. LNCS
- [22] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
- [23] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming—An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. San Mateo, CA: Morgan Kaufmann, 1998.
- [24] W. B. Langdon, *Genetic Programming and Data Structures*. Norwell, MA: Kluwer, 1998.
- [25] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. New York: Springer-Verlag, 2002.
- [26] W. B. Langdon, R. Poli, O. Holland, and T. Krink, "Understanding particle swarm optimization by evolving problem landscapes," in *Proc. IEEE Swarm Intelligence*, L. M. Gambardella, P. Arabshahi, and A. Martinoli, Eds., Pasadena, CA, Jun. 8–10, 2005, pp. 30–37. [Online]. Available: [http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon\\_2005\\_SIS.pdf](http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon_2005_SIS.pdf)
- [27] W. B. Langdon and R. Poli, "Evolving problems to learn about particle swarm and other optimizers," in *Proc. IEEE Congr. Evol. Comput.*, D. Corne, Z. Michalewicz, M. Dorigo, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, T. K. Chen, G. Raidl, A. Zalzal, S. Lucas, B. Paechter, J. Willies, J. J. M. Guervos, E. Eberbach, B. McKay, A. Channon, A. Tiwari, L. G. Volkert, D. Ashlock, and M. Schoenauer, Eds., Edinburgh, U.K., Sep. 2–5, 2005, vol. 1, pp. 81–88. [Online]. Available: [http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl\\_cec2005.pdf](http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_cec2005.pdf)
- [28] G. Syswerda, "A study of reproduction in generational and steady state genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlings, Ed., Jul. 15–18, 1990, pp. 94–101.
- [29] R. Poli, TinyGP. (See TinyGP GECCO 2004 competition) 2004. [Online]. Available: <http://cswww.essex.ac.uk/staff/sml/gecco/TinyGP.html>
- [30] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—A survey," *IEEE Trans. Evol. Comput.* vol. 9, no. 3, pp. 303–317, Jun. 2005. [Online]. Available: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?isnumber=30975&arnumber=1438403&count=6&index=4](http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=30975&arnumber=1438403&count=6&index=4)
- [31] R. Storn, "DeApp—An application in java for the usage of differential evolution," 1999. [Online]. Available: <http://http.icsi.berkeley.edu/~storn/devol.ps>
- [32] —, "Differential evolution." Feb. 15, 2005. [Online]. Available: <http://www.icsi.berkeley.edu/~storn/code.html>
- [33] H.-G. Beyer, *The Theory of Evolution Strategies*, ser. ser. Natural Computing Series. New York: Springer, 2001.
- [34] J. Lampinen and I. Zelinka, "On stagnation of the differential evolution algorithm," in *Proc. 6th Int. Mendel Conf. Soft Computing*, P. Osmera, Ed., Brno, Czech Republic, Jun. 7–9, 2000, pp. 76–83. [Online]. Available: <http://citeseer.ist.psu.edu/317991.html>
- [35] M. Clerc, "Stagnation analysis in particle swarm optimization or what happens when nothing happens," Jan. 19, 2006. [Online]. Available: [http://clerc.maurice.free.fr/ps0/stagnation\\_analysis/stagnation\\_analysis\\_s.pdf](http://clerc.maurice.free.fr/ps0/stagnation_analysis/stagnation_analysis_s.pdf)
- [36] M. Schoenauer and Z. Michalewicz, "Evolutionary computation at the edge of feasibility," in *Lecture Notes in Computer Science*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds., Berlin, Germany: Springer-Verlag, Sep. 22–27, 1996, vol. 1141, Parallel Problem Solving From Nature—PPSN IV, pp. 245–254. [Online]. Available: <http://www.cs.adelaide.edu.au/~zbysek/Papers/p26.pdf>
- [37] M. Clerc, "Particle swarm optimization," ISTE, 2006.
- [38] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.* vol. 8, no. 3, pp. 225–239, Jun. 2004. [Online]. Available: <http://ieeexplore.ieee.org/iel5/4235/28981/01304845.pdf?tp=&arnumber=1304845&isnumber=28981>

- [39] K. E. Parsopoulos and M. N. Vrahatis, "UPSO: A unified particle swarm optimization scheme," in *Proc. Int. Conf. Comput. Methods Sci. Eng.*, ser. Lecture Series on Computer and Computational Sciences. Attica, Greece: VSP International Science, Nov. 19–23, 2004, vol. 1, pp. 868–873. [Online]. Available: <http://www.math.upatras.gr/~kostasp/papers/ICCMSE04PV.pdf>
- [40] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.* 2006. [Online]. Available: <http://ieeexplore.ieee.org/iel5/4235/26785/101109TEVC2005857610.pdf?tp=&number=101109TEVC2005857610&isnumber=26785>
- [41] T. Peram, K. Veeramachaneni, and C. K. Mohan, "Fitness-distance-ratio based particle swarm optimization," in *Proc. 2003 IEEE Swarm Intell. Symp.*, Indianapolis, IN, Apr. 24–26, 2003, pp. 174–181. [Online]. Available: <http://dx.doi.org/doi:10.1109/SIS.2003.1202264>
- [42] B. C. H. Chang, A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Particle swarm optimization for protein motif discovery," *Genetic Program. Evol. Mach.*, vol. 5, no. 2, pp. 203–214, Jun. 2004.
- [43] A. Qin and P. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proc. IEEE Congr. Evol. Comput.*, D. Corne, Z. Michalewicz, B. McKay, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, G. Raidl, K. C. Tan, and A. Zalzala, Eds., Edinburgh, U.K., Sep. 2–5, 2005, vol. 2, pp. 1785–1791. [Online]. Available: <http://ieeexplore.ieee.org/servlet/opac?punumber=10417&isvol=2>
- [44] O. Hrstka and A. Kučerová, "Improvements of real coded genetic algorithms based on differential operators preventing premature convergence," *Adv. Eng. Softw.*, vol. 35, no. 3–4, pp. 237–246, Mar.–Apr. 2004.
- [45] J. Lampinen, "A constraint handling approach for the differential evolution algorithm," in *Proc. Congress on Evol. Comput.*, D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, Eds., Honolulu, HI, May 12–17, 2002, pp. 1468–1473.
- [46] S. Kukkonen and J. Lampinen, "An extension of generalized differential evolution for multi-objective optimization with constraints," in *Lecture Notes in Computer Science*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. T. A. Kabán, and H.-P. Schwefel, Eds. Berlin, Germany: Springer-Verlag, Sep. 18–22, 2004, vol. 3242, Proc. Parallel Problem Solving From Nature—PPSN VIII, pp. 752–761.
- [47] —, "GDE3: The third evolution step of generalized differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, D. Corne, Z. Michalewicz, B. McKay, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, G. Raidl, K. C. Tan, and A. Zalzala, Eds., Edinburgh, U.K., Sep. 2–5, 2005, vol. 1, pp. 443–450. [Online]. Available: [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?isnumber=33079&number=1554717&count=127&index=57](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=33079&number=1554717&count=127&index=57)

**W. B. Langdon** is a Senior Research Fellow in computer systems engineering at Essex University, Colchester, U.K. He worked on distributed real-time databases for control and monitoring of power stations at the Central Electricity Research Laboratories. He then joined Logica to work on distributed control of gas pipelines and later on computer and telecommunications networks. After returning to academe to receive the Ph.D. degree in genetic programming (sponsored by National Grid plc.), he has worked at the University of Birmingham, the CWI, UCL and, most recently, Essex University.



**Riccardo Poli** is a Professor in the Department of Computer Science, University of Essex, Colchester, U.K. He has coauthored *Foundations of Genetic Programming* (Springer-Verlag, 2002) with W. B. Langdon. He has published over 180 refereed papers on evolutionary algorithms (particularly genetic programming), neural networks, and image/signal processing. His main research interests include genetic programming (GP) and the theory of evolutionary algorithms.

Prof. Poli was elected a Fellow of the International Society for Genetic and Evolutionary Computation (ISGEC), in recognition of sustained and significant contributions to the field and the community, in July 2003. He has been cofounder and Co-Chair of EuroGP, the European Conference on Genetic Programming for 1998, 1999, 2000, and 2003. He was the Chair of the GP theme at the Genetic and Evolutionary Computation Conference (GECCO) 2002 (the largest conference in the field) and was Co-Chair of the prestigious Foundations of Genetic Algorithms (FOGA) Workshop in 2002. He has been (the first non-U.S.) General Chair of GECCO in 2004, and served as a member of the business committee for GECCO 2005. He is Technical Chair of the International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2006) and Competition Chair for GECCO 2006. He is an Associate Editor of *Evolutionary Computation* (MIT Press), *Genetic Programming and Evolvable Machines* (Springer), and the *International Journal of Computational Intelligence Research* (IJCIR). He has been program committee member of over 50 international events. He has presented invited tutorials on GP at ten international conferences. He is a member of the EPSRC Peer Review College and has attracted, as Principal Investigator or Co-Investigator, funding for over \$1.8M from EPSRC, DERA, Leverhulme Trust, Royal Society, and others.