Exploring Extended Particle Swarms: A Genetic Programming Approach

Riccardo Poli Cecilia Di Chio William B. Langdon
Department of Computer Science
University of Essex, UK
{rpoli, cdichi, wlangdon}@essex.ac.uk

ABSTRACT

Particle Swarm Optimisation (PSO) uses a population of particles that fly over the fitness landscape in search of an optimal solution. The particles are controlled by forces that encourage each particle to fly back both towards the best point sampled by it and towards the swarm's best point, while its momentum tries to keep it moving in its current direction.

Previous research started exploring the possibility of evolving the force generating equations which control the particles through the use of genetic programming (GP).

We independently verify the findings of the previous research and then extend it by considering additional meaningful ingredients for the PSO force-generating equations, such as global measures of dispersion and position of the swarm. We show that, on a range of problems, GP can automatically generate new PSO algorithms that outperform standard human-generated as well as some previously evolved ones.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Performance

Keywords

Particle Swarm Optimisation, Swarm Intelligence, Genetic Programming

1. INTRODUCTION

Researchers use the expression *swarm intelligence* to refer to the emergent collective intelligence of groups of simple agents, of which colonies of social insects such as termites, bees and ants are the best-known examples. Swarm intelligence offers a way to design intelligent systems (*swarm systems*) in which functions such as autonomy, emergence and distribution have replaced some less "natural" ones, e.g. control, programming and centralisation [5].

The class of swarm systems is a rich source of novel computational methods that can solve difficult problems efficiently and reliably. One of the best-developed techniques of this type is Particle Swarm Optimisation (PSO) [9]. PSOs use a population of candidate solutions (interacting particles) that fly over the fitness landscape and, controlled by forces that encourage each particle to fly back towards both the best point sampled by it and the swarm best, search for an optimal solution to a given problem. The forces act to over come the particle's momentum, which tries to keep the particle moving in the same direction at the same speed.

A variety of improvements to the basic form of PSO have been proposed and tested in the literature. As a result, it is very hard for a practitioner to decide what is the best PSO to use to solve a particular problem. This research is part of a project that aims to systematically categorise PSOs and explore extensions of particle swarms by including strategies from biology, by extending the physics of the particles and by providing a solid theoretical and mathematical basis for the understanding and problem-specific design of new particle swarm algorithms.

We take a purely computational approach, first pioneered in [17], where the force generating equations which control the particles in a PSO are automatically evolved through the use of genetic programming (GP). Interesting PSOs were evolved in this way in [17]. However, the exploration was limited by the use of the same ingredients present in the original PSO model by Kennedy and Eberhart, namely: the position and velocity of a particle, the position of the particle best and the position of the swarm best. The aim of the research presented here is to independently verify the findings of [17] and to extend the search by considering additional meaningful ingredients for the PSO force-generating equations, such as global measures of dispersion and position of the swarm. These are believed to be very important so as to provide a way of adapting the nature of the forces used depending on the current situation of the swarm as a whole.

Our results indicate that indeed GP can explore this extended space of possible PSOs, finding update rules that outperform human-generated as well as some previously evolved ones. Naturally, not all PSOs evolved can have be seen as having a connection with natural swarms, but this is not unusual in computational intelligence. For example, also in the field of neural networks researchers started from biologically inspired neurons. However, later research focused on non-biologically plausible models, such as the back-propagation rule, because of their interesting and useful properties. We expect something like this might happen in the field of PSOs. This paper is part of an effort to extend PSO models beyond real biology and physics and push the limits of swarm intelligence into the exploration of swarms as they could be.

Section 2 provides a brief review of the work on particle swarms to date. Since we build on [17], Section 3 describes in detail [17]'s use of GP to automatically generate and evolve PSOs tailored to particular tasks. In Section 4 we describe and motivate our variations on [17]'s basic approach. Section 5 describes our results and compares them to the performance of previously designed or evolved PSOs. Finally, Section 6 briefly restates our findings and lists potential future directions for research.

2. PARTICLE SWARM OPTIMISATION

PSO is a method for optimisation of continuous nonlinear functions, discovered through the simulation of a simplified social model (particle swarm) [10]. PSO has its roots in two main component methodologies:

- artificial life (ALife), in particular flocking, schooling and swarming theory, and
- evolutionary computation (EC).

In PSOs, which are inspired by flocks of birds and shoals of fish, a number of simple entities (the *particles*) are placed in the parameter space of some problem and each evaluates the fitness at its current location. Unique to the concept of PSO is "flying" over potential solutions through the hyperspace, accelerating towards better ones. Each particle determines its movement through the space by combining some aspect of the history of its own fitness values with those of one or more members of the swarm, and then moves through the parameter space with a velocity determined by the locations and processed fitness values of those other members, possibly together with some random perturbations. The next iteration takes place after all particles have been moved. Eventually the swarm as a whole is likely to move close to the best location.

More formally, in the case of an N dimensional problem, each particle's position, velocity and acceleration, can each be represented as a vector with N components (one for each dimension). In the original version of PSO, the equation controlling the particles is of the form

$$a_i = F(x_i, x_{s_i}, x_{p_i}, v_i) \tag{1}$$

for some function F and where

- a_i is the i^{th} component of the acceleration vector $a = (a_1, \cdots, a_N)$,
- x_i is the i^{th} component of the particle's current location $x = (x_1, \cdots, x_N)$,
- x_{s_i} is the i^{th} component of the best point visited by the swarm $x_s = (x_{s_1}, \cdots, x_{s_N})$,
- x_{p_i} is the i^{th} component of its personal best $x_p = (x_{p_1}, \cdots, x_{p_N})$, and
- v_i is the i^{th} component of the velocity vector $v = (v_1, \cdots, v_N)$.

In particular, in the simplest version of PSO, each particle is moved by two elastic forces, one attracting it with random magnitude to the fittest location so far encountered by the particle, and one attracting it with random magnitude to the best location encountered by any member of the swarm. More precisely,

$$a_i = \phi_1 R_1 (x_{s_i} - x_i) + \phi_2 R_2 (x_{p_i} - x_i) \tag{2}$$

where

- R_1 and R_2 are two independent random variables uniformly distributed in $[0, 1]$, and
- ϕ_1 and ϕ_2 are two learning rates which control the relative proportion of cognition and social interaction in the swarm.

The velocity of a particle v and its position x are updated every time step using the equations:

$$v_i(t) = v_i(t-1) + a_i \qquad x_i(t) = x_i(t-1) + v_i(t)$$

As this system can lead the particles to become unstable, with their speed increasing without control, the standard technique to avoid this happening is to bound velocities so that $v_i \in [-V_{max}, +V_{max}]$.

Early variations in PSO techniques involved the addition of analogues of physical characteristics to the members of the swarm, such as the "inertia weight" ω [18], where the velocity update equation is modified as follows:

$$v_i(t) = \omega v_i(t-1) + a_i \tag{3}$$

As in vector notation the velocity change can be written as $\Delta v = a - (1 - \omega)v$, the constant $1 - \omega$ effectively acts as a *friction coefficient*.

Following Kennedy's graphical examinations of the trajectories of individual particles and their responses to variations in key parameters [8] the first real attempt at providing a theoretical understanding of PSO was the "surfing the waves" model presented by Ozcan [15].

Clerc developed a comprehensive 5-dimensional mathematical analysis of the basic PSO system [7]. A particularly important contribution of that work was the use and analysis of a modified update rule:

$$v_i(t) = \kappa(v_i(t-1) + a_i)$$

where the constant κ (the *constriction coefficient*), if correctly chosen, guarantees the stability of the PSO without the need to bound velocities.

Further explorations of physics-based effects in the swarm have been done in recent years:

- Blackwell has investigated quantum swarms [4] and charged particles [3]
- Poli and Stephens have proposed a scheme in which the particles do not "fly above" the fitness landscape, but actually slide over it [16]
- Krink and collaborators have looked at a range of modifications to PSO, including ideas from physics (spatially extended particles [12], self-organised criticality [14]) and biology (e.g. division of labour [19])

- Angeline [1] introducing selection and Brits *et al.* [6] exploring niching are two examples of the intersection between PSOs and evolutionary computing

- Finally, in [17] the possibility of evolving the force generating equations to control the particles in a PSO through the use of GP was proposed. We describe this approach in detail in the next section, since this is the starting point for the work reported in this paper.

3. EVOLVING PARTICLE SWARMS

As has already been pointed out in the previous section, EC paradigms are closely related to the particle swarm method, as they provide the tools to build intelligent systems that model intelligent behaviour. One of the methodologies in the field of EC is genetic programming (GP), which is a technique used to evolve computer programs using a specialised representation (tree structures) and genetic operators, e.g., crossover and mutation, that are especially designed to handle such a representation [11, 2, 13].

As highlighted in Section 2, the equation controlling the particles in a PSO is of the form $a_i = F(x_i, x_{s_i}, x_{p_i}, v_i)$. The approach taken in [17] was to consider the function F as a program to be evolved using GP so as to maximise some performance measure. The aim was not to evolve a PSO that could beat all other PSOs on all possible problems, which is know to be impossible [20]. Instead, the aim was to evolve PSOs that could outperform known PSOs on specific classes of problems of interest.

The F functions evolved in [17] used only the original ingredients listed in Equation 1, namely: x_i, v_i, x_{p_i} and x_{s_i}. These were included in the terminal set together with a limited set of permissible constants (1.0, -1.0, 0.5 and -0.5) and a uniformly random number generator R, which returned random numbers in the range $[-1, 1]$. The function set included the standard arithmetic functions $+$, $-$, $\times$ and the protected division DIV.[1]

Since the aim was to obtain extended PSOs (XPSOs) which could solve a class of problems rather than just one problem, [17] used a fitness function which, using the program evolved by the GP as the function F, evaluated the performance of an XPSO on a *training set of problems taken from the given class*. The two classes of benchmark problems considered were:

City-block sphere problem class, instances of which have the form

$$f(x) = \sum_{i=1}^{N} |x_i - g_i|$$

with a single global optimum at $x = (g_1, \cdots, g_N)$, with $f(x) = 0$, and no local optima

Rastrigin's problem class, instances of which have the form

$$f(x) = 10N + \sum_{i=1}^{N} \left((x_i - g_i)^2 - 10\cos(2\pi(x_i - g_i))\right)$$

with one global optimum at $x = (g_1, \cdots, g_N)$, with $f(x) = 0$, and many local optima

[1]If $|y| <= 0.001$ DIV$(x, y) = x$ else DIV$(x, y) = x/y$.

The fitness function used to evaluate the performances of the PSO in each of the training cases consisted in measuring and accumulating the distance between the position of each particle in the swarm and the global optimum at the end of each PSO run, i.e. $\sum_x \sum_i |x_i - g_i|$. This particular fitness function was chosen to encourage the convergence of the swarm to the global optimum. A mild parsimony pressure was used to encourage the evolution of a simpler F.

The results of this approach were quite interesting. GP evolved the following XPSOs:

PSOG1 evolved when the training set was the shifted city-block sphere functions of dimension $N = 2$ and had the following form:

$$F = (x_{s_i} - x_i) - (v_i R)$$

This was expected to perform well on unimodal objective functions. It is particularly interesting since it includes both a deterministic, 100% social component and a random friction component (cf. Equation 3).

PSOG2 evolved when the training set included shifted city-block sphere functions of dimension $N = 2$ and had the following form:

$$F = 0.5\left((x_{s_i} - x_i) + (x_{p_i} - x_i) - v_i\right)$$

This is interesting because it is completely deterministic (particles are attracted towards the middle between swarm best and particle best) and because it includes standard friction.

PSOG3 evolved when the training set included shifted Rastrigin functions of dimension $N = 2$ and had the following form

$$F = R_1(x_{s_i} - x_i) - 0.75 R_2 R_1 x_i x_{s_i}^2 - 0.25 R_3 R_2 R_1 x_i x_{s_i}$$

This was expected to perform well on highly multi-modal objective functions. Interestingly, it does not use information about each particle's best, probably because, in a highly multi-modal landscape, particles should not trust their own observations too much. The second term tends to push the particles towards the origin unless the swarm best is near the origin. The third term appears to be junk code.

In tests with off-sample problem instances **PSOG1** resulted to be better than a standard PSO on the unimodal city-block sphere problem class. On the Rastrigin function problem class, **PSOG3** outperformed all other (hand-designed and evolved) PSOs by a considerable margin. Also, **PSOG3** (which wasn't evolved on sphere functions) did better than the standard PSO on the sphere problem class, suggesting **PSOG3** may be a good all-rounder.

4. OUR APPROACH

One of our aims was to independently verify the findings of [17] and to extend the search by considering additional meaningful ingredients for the PSO force-generating equations. So, in our work we followed the approach in [17] summarised in the previous section as closely as possible.

As evolving specialised search algorithms is typically a heavy computational task, we used a very efficient C implementation of GP and implemented a minimalist PSO engine, also in C.

A limitation of our approach, and of the standard PSO, is that only very limited information about the global state of the swarm is provided to particles (via the force generating equations). Indeed, only the coordinates of the swarm best are provided. However, it is arguable that better decisions about which direction to explore next could be made by each particle if more information about the rest of the swarm was provided.

So, in addition to the standard ingredients x_i, v_i, x_{p_i} and x_{s_i}, we decided to also give our GP system global measures of dispersion and position of the swarm. GP was now allowed to evolve force-generating equations of the form

$$F'(x_i, x_{s_i}, x_{p_i}, v_i, x_{c_i}, d), \tag{4}$$

where x_{c_i} is the i-th component of the centre of mass of the swarm and d (dispersion) is the average distance of each particle from the centre of mass of the swarm.

Of course, the GP may find that using such terms adds nothing to the efficiency of the PSO (i.e. no such terms would be included in the evolved force equation). However, if it does, the final form will provide insight into precisely how such terms have increased the PSO's efficiency.

Following [17] we also used the city-block sphere problem class and the Rastrigin problem class, both of dimensions $N = 2$ and $N = 10$. At the beginning of each GP run, 10 random problems from either the city-block sphere or the Rastrigin function class were generated by choosing the values g_i uniformly at random from the range $[-G, G]$, where $G = 1.0$ and $G = 2.0$.

The fitness function mentioned in the previous section was used. During fitness evaluation, in order to limit the computational load of the simulations, PSOs with 10 particles have been used and they were run for 30 iterations on each of the 10 problems.

Initial coordinates for the particles were drawn uniformly at random from the interval $[-5, 5]$, while their initial velocity is set to 0. As the initial configuration of the swarm is random, the performance of the PSO can vary substantially according to the position of the particles. Thus, as in [17], for each problem, the PSO was run 5 times with different initial random configurations, i.e. positions of the 10 particles.

To ensure stability, velocities have been updated using Clerc's update rule (see Section 2), using a constriction factor $\kappa = 0.7$, and constrained the components v_i of particle velocities within the range $[-2.0, +2.0]$.

As far as the GP system is concerned, we used steady state binary tournaments for parent selection and binary negative tournaments to decide who to remove from the population. The initial population was created using the grow method with max depth of 6 levels (the root node being at level 0). A population of size of 1000 was used, with 90% standard subtree crossover (with uniform random selection of crossover points) and 10% point mutation with a 2% chance of mutation per tree node.

Runs were terminated either manually, when fitness appeared to be sufficiently good, or automatically at generation 100. When not stopped prematurely, typically runs took between 30 minutes and 5 hours to complete on a 3GHz single CPU Linux PC.

5. RESULTS

Seven new PSOs have been automatically created by GP. Two (**PSOCENTRE1** and **PSOCENTRE2**) were created by GP when it was given the centre of the swarm. Three more (**PSODISP1**, **PSODISP2** and **PSODISP3**) when it was given the spread of the swarm (dispersion). The last two (**PSOCD1** and **PSOCD2**) evolved when the function set included both. Apart from **PSODISP3**, each new PSO was created using instances of the city-block sphere problem.

Tables 1 and 2 give the performance of our 7 newly evolved XPSOs. The tables show on average how close they came to the global optima, both on new instances of problems like those on which they were created (e.g. city-block sphere) and also on instances of different problems (e.g. shifted Rastrigin). For comparison, both tables also give results for standard (human designed) PSOs (**PSO**, **PSOD1**, **PSOR0** and **PSOR1**, cf. Equation 2) and PSOs given in [17] (**PSOG1**, **PSOG2** and **PSOG3**). The remainder of this section describes the standard and our new extended PSOs (Section 3 described **PSOG1**, **PSOG2** and **PSOG3**). While Section 5.1 analyses them in terms of the results presented in Tables 1 and 2.

The standard human-designed PSOs (cf. Equation 2) we used for comparison are:

PSO a version of the standard PSO where $\phi_1 = \phi_2 = 1.0$

$$F = R_1(x_{s_i} - x_i) + R_2(x_{p_i} - x_i)$$

PSOD1 a deterministic (no random coefficients) 100% social version of the standard PSO (i.e. the swarm best totally dominates individual particle bests, so $\phi_1 = 1$, $\phi_2 = 0$)

$$F = (x_{s_i} - x_i)$$

PSOR0 a PSO controlled by random forces

$$F = 2.0R - 1.0$$

PSOR1 a 100% social ($\phi_1 = 1, \phi_2 = 0$) version of the standard PSO

$$F = R(x_{s_i} - x_i)$$

In several GP runs where only the centre of mass was added to the original primitive set we obtained the following new functions for controlling the PSO force

PSOCENTRE1 was evolved when the training set included two dimensional ($N = 2$) city-block sphere functions shifted by random amounts chosen from the range ±1.0 (i.e. $G = 1.0$). It is completely deterministic and includes a friction (cf. Equation 3) term

$$F = (x_{s_i} - x_i) + (x_{p_i} - x_i) + x_{s_i}x_{c_i} - 0.5v_i$$

When the third term of the equation is zero, e.g. because the swarm best or the centre of the swarm are near the origin, the behaviour of the particles is similar to **PSOG2** (cf. Section 3). When $x_{s_i}x_{c_i}$ is non-zero, it tends either to slow or to accelerate the motion of the particles, according to the positions of the center of mass and the position of the swarm best. Note however, unlike the other terms, it is not independent of co-ordinate transformations. For example the origin of the co-ordinate system has a special significance which would change if the co-ordinates where translated.

PSOCENTRE2 was evolved when the training set included the shifted two dimensional ($N = 2$) city-block sphere functions but with larger random shifts (i.e. $G = 2.0$ rather than 1.0). It is interesting since it includes both a deterministic component and a random friction component

$$F = (x_{s_i} - x_i) - x_i x_{c_i} - (v_i R)$$

When the second term of the equation is zero (e.g. because either the swarm centre or the individual particle is close to the origin), this rule (like **PSOCENTRE1**) becomes **PSOG2**. When $x_i x_{c_i}$ is non-zero it tends either to slow or to accelerate the motion of the particles, according to the position of the centre of mass with respect to the position of the particles. Note again, unlike the other terms, $x_i x_{c_i}$ is dependent on the co-ordinate system.

In GP runs where only the dispersion around the centre of mass was added to the primitive set we obtained the following rules

PSODISP1 was evolved when the GP was trained on two dimension ($N = 2$) city-block sphere functions randomly shifted by an amount uniformly selected from $-1 \ldots +1$ ($G = 1.0$). The evolved functions contains a random friction component and is 100% social (i.e. $\phi_1 = 1$). However these are dominated by the $1/d$ term, which increases the force as the swarm concentrates about a point. This may be a good strategy on multi-modal landscapes where it might prevent premature convergence to a poor local optima. The constant term tends to increase particle's velocities. Perhaps this is need to counteract the tendency for velocity to be reduced?

$$F = \frac{1}{d}(x_{s_i} - x_i - v_i R) + 1$$

PSODISP2 was evolved when the GP was trained as above, except the city-block sphere functions were randomly shifted by twice as much ($N = 2$ and $G = 2.0$). The evolved function is completely deterministic

$$F = (x_{s_i} - x_i) + (x_{p_i} - x_i) - d x_i$$

Should the swarm collapse (in which case the dispersion term d would be zero) then this rule would become the standard (deterministic) PSO. Normally d is non-zero, so the third component ($d x_i$) tends to push the swarm towards the origin.

PSODISP3 was evolved when the training set included the shifted two dimensional ($N = 2$) Rastrigin functions with $G = 2.0$. It is completely deterministic

$$F = -2x_i + x_{s_i}\,(0.75 - 2d)$$

The first term tends to drive particles towards the origin however the oscillations about the origin may be counteracted by the second term, which (if the dispersion $d > 3/8$) tends to concentrate the swarm closer to its best point.

Finally, in runs where both the centre of mass and the dispersion were available to GP we evolved the following PSOs

PSOCD1 was evolved by training the GP on the two dimensional shifted city-block sphere functions of dimension $N = 2$ and $G = 1.0$

$$F = (x_{s_i} - x_i) - \frac{R^2}{d} v_i$$

The first term is the 100% social term, which tends to move the swarm closer to its best point. The random friction term ($\frac{R^2}{d} v_i$) is inversely proportional to swarm's dispersion d. So when the particles are close to each other the friction is higher than it is when the swarm is more spread. This could mean that at the beginning the search is more rapid while, when the particles get closer their movement becomes slower.

PSOCD2 was evolved as above but with larger random shifts of city-block sphere ($N = 2$ and $G = 2.0$). We include it because it is completely deterministic and includes standard friction as well as the centre and spread of the swarm.

$$F = x_{s_i} + x_{p_i} + d^2 \frac{x_i}{x_{c_i}} + 0.5 v_i$$

5.1 Analysis of results

In order to compare the behaviour of the standard human designed PSOs and the evolved extended ones, we tested all of them on 30 random problems taken from the city-block sphere and Rastrigin function classes for dimension $N = 2$ and $N = 10$. The problems were generated by selecting the components of the global optimum g_i uniformly at random from the interval $[-G, +G]$, with $G = 1.0$ and $G = 2.0$. For each problem instance, 30 independent runs of each PSO have been performed.

Tables 1 and 2 show the mean (over the 30 runs) and the standard deviation (in brackets) of the normalised distance between the best location found by each PSO and the global optimum, i.e. the average absolute error between the coordinates of the swarm best and the coordinates of the global optimum, $\sum_i |x_s - g_i|/N$. This gives an idea of how close the particles got to the optimum in each dimension.

City-block sphere

Except **PSOCD2**, all PSOs do quite well on the city-block sphere problem (see table 1). When the dimension of the problem is $N = 2$, **PSOG1** and **PSOCD1** are both better than the standard PSO, and **PSOCD1** is even better than **PSOG1** when the global optimum is in the range $[-2.0, 2.0]$. However, for $N = 10$ and $G = 1.0$, **PSOG3** is the best, even though it wasn't evolved on sphere functions. For $G = 2.0$, the random PSO **PSOR1** is the best, but **PSOCD1** is the second best.

Rastrigin

In this case as well, **PSOCD2** is worse than any other in each configuration of the problem (see table 2). For $N = 2$, and both $G = 1.0$ and $G = 2.0$, **PSOG3** outperforms all other PSOs. **PSODISP2** is a close second best, surprisingly as it has been evolved on the city-block sphere class of problems. For $N = 10$ and the same (two) G values, the situation is reversed, **PSODISP2** being the best and **PSOG3** the second best (but still with values very close to each other). In all four configurations, **PSODISP1** is the third best, with values close to the best ones, while all other

City-block sphere

	N=2 G=1.0	N=2 G=2.0	N=10 G=1.0	N=10 G=2.0
PSO	0.184	0.22	0.826	0.946
	(0.248)	(0.273)	(0.579)	(0.548)
PSOD1	0.002	0.002	0.309	0.381
	(0.0005)	(0.0005)	(0.02)	(0.04)
PSOR0	0.257	0.28	1.6	1.654
	(0.032)	(0.032)	(0.016)	(0.04)
PSOR1	0.003	0.003	0.279	**0.315**
	(0.0005)	(0.0005)	(0.018)	(0.029)
PSOG1	**0.0005**	0.002	0.455	0.554
	(0.0005)	(0.002)	(0.02)	(0.047)
PSOG2	0.048	0.069	0.664	0.797
	(0.011)	(0.034)	(0.031)	(0.072)
PSOG3	0.009	0.043	**0.183**	0.456
	(0.004)	(0.025)	(0.022)	(0.101)
PSOCENTRE1	0.124	0.245	0.776	1.026
	(0.057)	(0.089)	(0.046)	(0.107)
PSOCENTRE2	0.271	0.297	1.131	1.26
	(0.022)	(0.042)	(0.04)	(0.07)
PSODISP1	0.047	0.051	1.212	1.256
	(0.005)	(0.007)	(0.043)	(0.076)
PSODISP2	0.003	0.009	0.262	0.545
	(0.002)	(0.004)	(0.029)	(0.094)
PSODISP3	0.054	0.239	0.381	0.809
	(0.021)	(0.1)	(0.049)	(0.159)
PSOCD1	**0.0005**	**0.0005**	0.283	0.353
	(0.00)	(0.00)	(0.019)	(0.036)
PSOCD2	0.576	0.611	1.733	1.78
	(0.021)	(0.038)	(0.015)	(0.037)

Table 1: Mean (and standard deviation) over 30 runs of normalised distance between swarm best found by each PSO and the centre of the City-block sphere. Best PSO in bold.

Rastrigin

	N=2 G=1.0	N=2 G=2.0	N=10 G=1.0	N=10 G=2.0
PSO	0.726	0.859	1.336	1.471
	(0.322)	(0.31)	(0.47)	(0.382)
PSOD1	0.777	0.896	1.309	1.406
	(0.046)	(0.108)	(0.039)	(0.065)
PSOR0	0.778	0.829	1.875	1.901
	(0.083)	(0.073)	(0.044)	(0.055)
PSOR1	0.638	0.687	1.332	1.402
	(0.076)	(0.09)	(0.065)	(0.06)
PSOG1	0.89	1.058	1.293	1.389
	(0.101)	(0.1)	(0.063)	(0.072)
PSOG2	0.701	0.871	1.131	1.267
	(0.082)	(0.119)	(0.045)	(0.067)
PSOG3	**0.307**	**0.556**	0.578	**0.943**
	(0.078)	(0.152)	(0.062)	(0.119)
PSOCENTRE1	0.885	1.039	1.268	1.458
	(0.147)	(0.113)	(0.057)	(0.086)
PSOCENTRE2	0.928	1.048	1.516	1.639
	(0.08)	(0.107)	(0.065)	(0.062)
PSODISP1	0.659	0.704	1.895	1.935
	(0.082)	(0.089)	(0.062)	(0.072)
PSODISP2	0.326	0.642	**0.562**	**0.943**
	(0.067)	(0.144)	(0.077)	(0.121)
PSODISP3	0.353	0.795	0.602	0.993
	(0.107)	(0.233)	(0.089)	(0.13)
PSOCD1	0.717	0.745	1.326	1.405
	(0.089)	(0.108)	(0.067)	(0.082)
PSOCD2	1.131	1.14	1.948	1.994
	(0.092)	(0.104)	(0.036)	(0.056)

Table 2: Mean (and standard deviation) over 30 runs of normalised distance between the swarm best location found by each PSO and the global optimum of the Rastrigin function. Best PSO in bold.

PSOs (both the hand-designed and the extended ones) are much less efficient. The fact that **PSODISP2** is very good on Rastrigin problems even though it has been evolved on the city-block sphere class, suggests that this XPS could be a good all-rounder.

It is interesting to compare the dynamic behaviour of the best PSOs found. In particular, the figures visually compare the behaviour of the swarm and how the dispersion of the swarm around the centre of mass varies in time for two XPSOs, **PSOG3** and **PSODISP2**, together with the standard **PSO**, for the Rastrigin class of problems with dimension $N = 2$ and global optimum in the origin.

PSO (figure 1) and **PSODISP2** (figure 2) show typical behaviour for efficient PSOs, with particle trajectories rapidly converging to a tight symmetric spiral around the global optimum, with **PSODISP2** doing so more efficiently. It seems that the dispersion term ensures that the swarm remains in a denser cloud.

PSOG3 (figure 3) behaves differently: the particles still converge rapidly, but to a point slightly away from the optimum. This is still close enough to ensure good GP fitness values. However the swarm best converges to the optimum better than all the others, as table 2 confirms.

6. CONCLUSION

We have shown that, through the use of genetic programming (GP), it is possible, in only a few hours, to automatically evolve a variety of new Particle Swarm Optimisation (PSO) algorithms that work at least as well as, and in some cases considerably better than, the standard existing human-designed ones. Including information about the spread of the swarm allows GP to evolve **PSODISP2**, which appears to be a good all-rounder. Moreover, through the analysis of the evolved components, we have suggested what types of PSOs are best for different landscapes. This work represents an important step within a new research trend: using search algorithms to discover new search algorithms.

In future research, we intend to:

1. apply the approach to a variety of real-world problems
2. extend the approach by allowing GP to use more information on the past history of the swarm to control the particles
3. explore the effects and benefits of using different performance measures for PSO evolution.

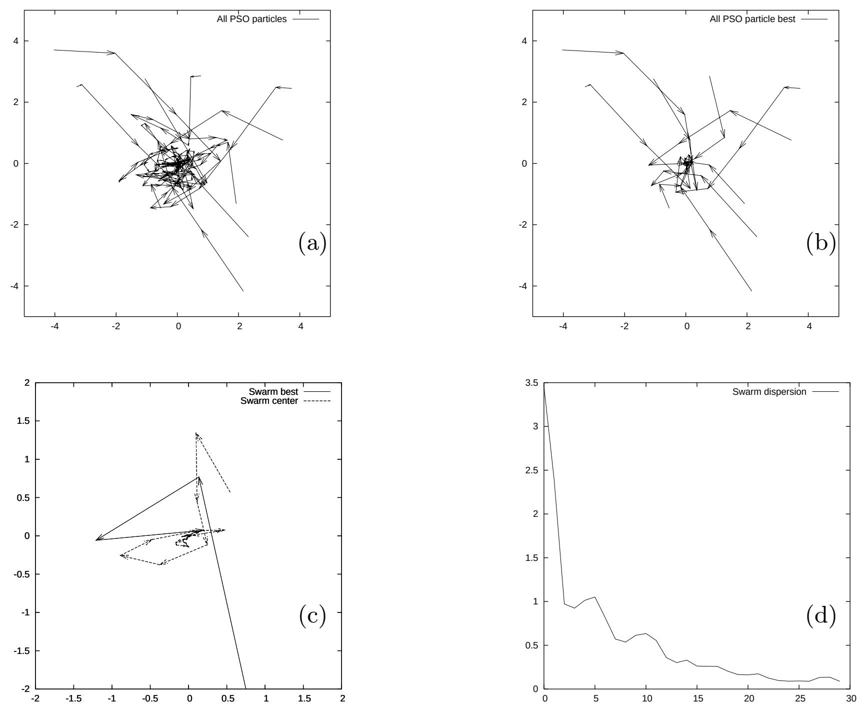

Figure 1: Prototypical run of PSO on the Rastrigin problem of dimension $N = 2$ showing, from top to bottom: (a) Trajectories of the particles, (b) Trajectories of the particle bests, (c) Trajectories of the swarm bests and of the centre of mass of the swarm, (d) Dispersion of the swarm around the centre of mass

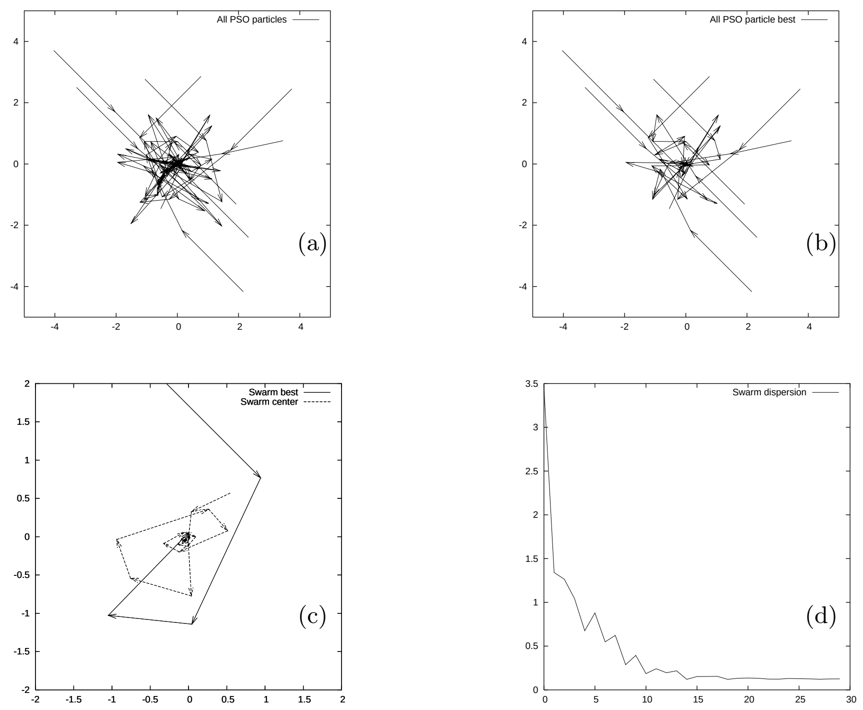

Figure 2: Prototypical run of PSODISP2 on the Rastrigin problem of dimension $N = 2$ showing, from top to bottom: (a) Trajectories of the particles, (b) Trajectories of the particle bests, (c) Trajectories of the swarm bests and of the centre of mass of the swarm, (d) Dispersion of the swarm around the centre of mass

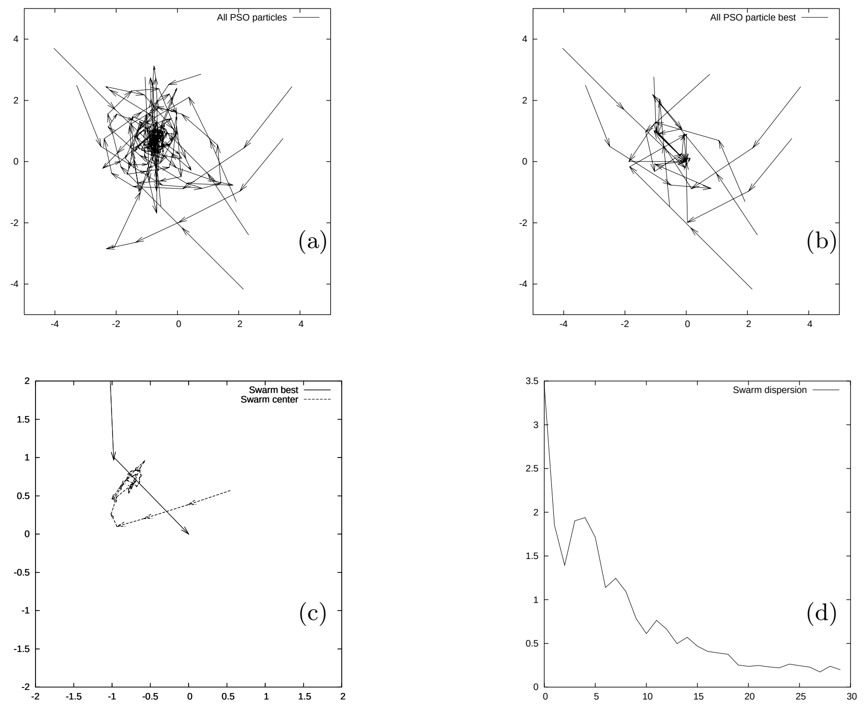

Figure 3: Prototypical run of PSOG3 on the Rastrigin problem of dimension $N = 2$ showing, from top to bottom: (a) Trajectories of the particles, (b) Trajectories of the particle bests, (c) Trajectories of the swarm bests and of the centre of mass of the swarm, (d) Dispersion of the swarm around the centre of mass

7. ACKNOWLEDGMENTS

The authors would like to thank EPSRC (grant GR/T11234/01) for financial support.

We would also like to thank Owen Holland, Chris Stephens and Thiemo Krink for their comments.

8. REFERENCES

[1] P.J. Angeline, *Using Selection to Improve Particle Swarm Optimization*, ICEC, 84–89, 1998.

[2] W. Banzhaf, P. Nordin, R.E. Keller and F.D. Francone, *Genetic Programming: an Introduction*, Morgan Kaufmann Publishers, 1998.

[3] T.M. Blackwell and P.J. Bentley, *Dynamic Search with Charged Swarms*, GECCO, 19–26, 2002.

[4] T.M. Blackwell and J. Branke, *Multi-swarm Optimization in Dynamic Environments.*, EvoWorkshops, 489-500, 2004.

[5] E. Bonabeau, M. Dorigo and G. Theraulaz, *Swarm Intelligence: from Natural to Artificial Systems*, Oxford University Press, 1999.

[6] R. Brits, A.P. Engelbrecht and B. Bergh, *A Niching Particle Swarm Optimizer*, SEAL, 692–696, 2002.

[7] M. Clerc and J. Kennedy, *The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space*, IEEE Trans. Evolutionary Computation, 6, 1, 58-73, 2002.

[8] J. Kennedy, *The Behavior of Particles*, Evolutionary Programming, 581-589, 1998.

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

[10] J. Kennedy and R.C. Eberhart, *Particle Swarm Optimization*, IEEE ICNN, 1942-1948, 1995.

[11] J.R. Koza, *Genetic Programming*, MIT Press, 1992.

[12] T. Krink, J.S. Vesterstrøm, and R. Riget, *Particle Swarm Optimisation with Spatial Particle Extension*, CEC, 1474-1479, 2002.

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

[14] M. Løvbjerg and T. Krink, *Extending Particle Swarms with Self-Organized Criticality*, CEC, 1588-1593, 2002.

[15] E. Ozcan and C.K. Mohan, *Particle Swarm Optimization: Surfing the Waves*, CEC, 1939–1944, 1999.

[16] R. Poli and C.R. Stephens, *Constrained Molecular Dynamics as a Search and Optimization Tool*, EuroGP, 150-161, 2004.

[17] R. Poli,W.B. Langdon and O. Holland, *Extending Particle Swarm Optimisation via Genetic Programming*, EuroGP, 291-300, 2005.

[18] Y. Shi and R.C. Eberhart, *A Modified Particle Swarm Optimizer*, CEC, 69–73, 1999.

[19] J.S. Vesterstrøm, J. Riget and T. Krink, *Division of Labor in Particle Swarm Optimisation*, CEC, 1570-1575, 2002.

[20] D. Wolpert and W.G. Macready, *No Free Lunch Theorems for Optimization*, IEEE Trans. Evolutionary Computation, 1, 1, 67-82, 1997.