To appear in the Proceedings of EuroGP 2005

Extending Particle Swarm Optimisation via Genetic Programming

Riccardo Poli, William B. Langdon, and Owen Holland

Department of Computer Science, University of Essex, UK

Abstract. Particle Swarm Optimisers (PSOs) search using a set of interacting particles flying over the fitness landscape. These are typically controlled by forces that encourage each particle to fly back both towards the best point sampled by it and towards the swarm's best. Here we explore the possibility of evolving optimal force generating equations to control the particles in a PSO using genetic programming.

1 Introduction

The class of complex systems sometimes referred to as *swarm systems* is a rich source of novel computational methods that can solve difficult problems efficiently and reliably. When swarms solve problems in nature, their abilities are usually attributed to swarm intelligence; perhaps the best-known examples are colonies of social insects such as termites, bees and ants. In recent years it has proved possible to identify, abstract and exploit the computational principles underlying some forms of swarm intelligence, and to deploy them for scientific and industrial purposes. One of the best-developed techniques of this type is Particle Swarm Optimisation (PSO) [9].

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 or function, and each evaluates the fitness at its current location. Each particle then determines its movement through the parameter 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 moving through the parameter space with a velocity determined by the locations and processed fitness values of those other members, along with some random perturbations. The next iteration takes place after all particles have been moved. Eventually the swarm as a whole, like a flock of birds collectively foraging for food, is likely to move close to the best location.

This simple model can deal with difficult problems efficiently. Naturally, different variations of the basic recipe have been tried and compared to existing techniques and different application areas have been investigated. However, the situation overall is still as it was in 2001 when Kennedy and Eberhart wrote: "...we are looking at a paradigm in its youth, full of potential and fertile with new ideas and new perspectives...Researchers in many countries are experimenting with particle swarms...Many of the questions that have been asked have not yet been satisfactorily answered." [9]

This research is part of a project that aims to systematically explore the extension 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. In this paper, in 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, we study the possibility of evolving, through the use of genetic programming (GP) [10, 12], the force generating equations to control the particles in a PSO. This gives us a methodology for routinely "inventing" specialised PSOs which are near-optimum for specific domains. Our aim is to verify the feasibility of this approach and to understand, through the analysis of the evolved components, what types of PSOs are best for different landscapes.

Section 2 provides a review of the work to date on particle swarms, while Section 3 describes how to use GP to automatically generate PSO tailored to particular tasks. The results section (4) is followed, in Section 5, by a brief restatement of our findings and future direction.

2 Particle Swarm Optimisation

The initial ideas of James Kennedy (a social psychologist) and Russ Eberhart (an engineer and computer scientist) were essentially aimed at producing computational intelligence by exploiting simple analogues of social interaction, rather than purely individual cognitive abilities. Their 1995 simulations, influenced by Heppner's work [7], involved analogues of bird flocks searching for corn. Their continuing emphasis on the social nature of intelligence, even in humans, can be seen in their book, Swarm Intelligence [9].

In the simplest (and original) 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.[1] Suppose we are dealing with 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). Starting with the acceleration vector, $a = (a_1, \cdots, a_N)$, each component, a_i, is given by

$$a_i = \psi_1 R_1 (x_{s_i} - x_i) + \psi_2 R_2 (x_{p_i} - x_i)$$

where x_{s_i} is the i^{th} component of the best point visited by the swarm, x_i is the i^{th} component of the particle's current location, x_{p_i} is the i^{th} component of its personal best, R_1 and R_2 are two independent random variables uniformly distributed in [0,1] and ψ_1 and ψ_2 are two learning rates. ψ_1 and ψ_2 control the relative proportion of cognition and social interaction in the swarm. The same formula is used independently for each dimension of the problem.

[1] Here we limit ourselves to the case where particles can socially interact freely. More general models constrain inter-particle interactions via a neighbourhood structure.

The velocity of a particle $v = (v_1, \cdots, v_n)$ and its position are updated every time step using the equations:

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

This system can lead the particles to become unstable, with their speed increasing without control. This is harmful to the search and need to be controlled. The standard technique 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 [17], where the velocity update equation is modified as follows:

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

In vector notation, the velocity change can be written as $\Delta v = a - (1-\omega)v$. That is, the constant $1-\omega$ acts effectively 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 [14]. Shortly afterwards, Clerc developed a comprehensive 5-dimensional mathematical analysis of the basic PSO system [5]. 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 is called a constriction coefficient. If κ is correctly chosen, it guarantees the stability of the PSO without the need to bound velocities.

More recently, there have been further explorations of physics-based effects in the swarm. For example, Blackwell has investigated quantum swarms [3] and charged particles [2], and Poli and Stephens have proposed a scheme in which the particles do not "fly above" the fitness landscape, but actually slide over it [15]. Krink and collaborators have looked at a range of modifications of PSO, including ideas from physics (spatially extended particles [11], self-organised criticality [13]) and biology (e.g. division of labour [16]). Finally, there has also been cross-fertilisation between PSOs and evolutionary computing. For example, Angeline [1] introduced selection and Brits *et al.* [4] have explored niching.

3 Evolution of PSOs via Genetic Programming

In the original PSO and well as in most improvements proposed in the literature (see previous section), the equation controlling the particles is of the form:

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

for some function F. Our approach to explore the space of possible PSOs is to use GP to evolve the function F so as to maximise some performance measure.

Clearly, we cannot expect to be able to evolve a PSO that can beat all other PSOs on all possible problems [18]. We should, however, be able to evolve PSOs that can outperform known PSOs on specific classes of problems of interest.

Evolving specialised search algorithms is typically a heavy computational task. So, a very efficient implementation of both PSO and GP were required for this work. Since nothing particularly fancy is required of the GP environment (except efficiency), we used a small and highly efficient C implementation of GP. For efficiency, we implemented our own minimalist PSO engine in C. This has to be compact and efficient in that it is invoked multiple times and for many time steps during the fitness evaluation of each GP program.

The function set for GP included the functions $+, -, \times$ and the protected division DIV. I.e. if $|y| <= 0.001$ $\mathrm{DIV}(x, y) = x$ else $\mathrm{DIV}(x, y) = x/y$. The terminal set included the coordinate of a particle x_i, the corresponding component of the velocity v_i, the coordinate of the best point visited by the particle x_{p_i}, the coordinate of the best point visited by the swarm x_{s_i}, the numerical constants 1.0, -1.0, 0.5, -0.5, and finally a zero-arity function `R` which returns random numbers uniformly distributed within the range $[-1, 1]$.

In order to evolve PSOs that are able to solve a class of problems as opposed to just one problem, we need to build a fitness function which uses the program being evaluated as the F function in a PSO, and evaluates the performance of the resulting PSO on a training set of problems taken from the given class.

In our study we considered two classes of benchmark problems, the city-block sphere problem class and the Rastrigin's problem class, of two different dimensions, $N = 2$ and $N = 10$. Problem instances from the city-block sphere class have the following form:

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

Every city-block problem has a single global optimum (at $\mathbf{x} = (c_1, \ldots, c_N)$, where $f(\mathbf{x}) = 0$) and no local optima. Problem instances from the Rastrigin's class have the following form:

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

This has many local optima and one global optimum $\mathbf{x} = (c_1, \ldots, c_N)$ with $f(\mathbf{x}) = 0$.

At the beginning of each GP run, 10 random problems were generated from the chosen class of functions (either the city-block sphere or the Rastrigin function class) by choosing the values c_i uniformly at random from the range $[-1, 1]$. To limit the computational load of the simulations, during fitness evaluation we used PSOs with 10 particles and run them for only 30 iterations on each of the 10 problems. The initial coordinates for the particles were drawn uniformly at random from the interval $[-5, 5]$. Since performance can vary substantially with the initial random position of the particles, for each problem the PSO was run 5

times with different initial random positions for the 10 particles. Initial particle velocity is 0. To ensure stability we updated velocities using Clerc's update rule (see previous section), using a constriction factor $\kappa = 0.7$. We also clipped the components v_i of particle velocities within the range $[-2.0, +2.0]$.

In each of the $10 \times 5 = 50$ training cases, the performance of the PSO needs to be evaluated. More than one performance measure could be used. For example, if one wanted to encourage the convergence of the swarm at the global optimum, performance could be evaluated as the sum of the distances between the global optimum and each particle at the end of the 30 PSO iterations. If one is only interested in the ability of the PSO to find the global optimum, then the distance between the swarm best and the global optimum at the end of the PSO iterations should be used as a performance measure. If one does not care about global optima, but is only interested in achieving good values of the objective function, then the difference between the best objective function value observed in a PSO run and the objective function value at the global optimum could be used as a performance measure. Fitness functions that encourage the swarm not to collapse onto the swarm best could be defined for dynamic problems which require the ability to track moving optima. And so on.

We experimented with two fitness functions: a) we measured and accumulated (over 50 fitness cases) the (city-block) distance between the swarm best and the global optimum at the end of each PSO run

$$\sum_i |x_{s_i} - c_i|$$

and b) we measured and accumulated (over 50 fitness cases and 10 particles) the (city-block) distance between each particle's position and the global optimum at the end of each PSO run

$$\sum_x \sum_i |x_i - c_i|$$

The negation of either (a) or (b) minus a parsimony pressure term (see below) was returned as the fitness of the GP program controlling the PSO.

In the GP system 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). We used population sizes of 1000 and 5000 individuals. We used 90% standard sub-tree 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. To favour readability and understandability of the evolved solutions, a mild parsimony pressure (parsimony coefficient=0.01) was applied to the fitness function to encourage the evolution of a simpler F.

4 Results

In order to be able to evaluate the PSOs produced by GP, we compared them with a number of human-designed update rules, most of which have previously appeared in the literature. The update rules included:

PSO a version of the standard PSO where $\psi_1 = \psi_2 = 1.0$, that is

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

PSOD1 a deterministic (no random coefficients) 100% social version ($\psi_1 = 0$, $\psi_2 = 1$) of the standard PSO:

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

PSOR0 a PSO controlled by random forces

$$F = 2.0\dot{R} - 1.0$$

PSOR1 a 100% social ($\psi_1 = 0, \psi_2 = 1$) version of the standard PSO

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

In our GP runs we evolved several high performance PSOs. Three of the most interesting ones are:

PSOG1 was evolved when the training set was the shifted city-block sphere functions of dimension $N = 2$, and so it was expected to perform well on unimodal objective functions:

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

This is particularly interesting since it includes a deterministic, 100% social component as well as a random friction component.

PSOG2 was also evolved when the training set included shifted city-block sphere functions of dimension $N = 2$. Its equation is equivalent to

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

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 was evolved when the training set included shifted Rastrigin functions of dimension $N = 2$, and so it was expected to perform well on highly multimodal objective functions. Its equation is equivalent to:

$$F = R_1(x_{s_i} - x_i) - 0.75R_2R_1x_ix_{s_i}^2 - 0.25R_3R_2R_1x_ix_{s_i}$$

This is interesting for a number of reasons. Firstly, it does not use information about each particle's best. Probably this is because, in a highly multimodal landscape, particles should not trust their own observations too much. Their personal best is likely to belong to the basin of attraction of a deceptive local optimum. The swarm best, instead, has a higher chance of being in the basin of attraction of the global optimum, and so particles should aim at exploring its surroundings. **PSOG3**'s second term is also interesting. Unless the swarm best is near the origin, this component will tend to push the particles towards the origin. The push is very mild if the swarm best is not too far from the origin, but it becomes quite strong otherwise. Clearly, the reason why this component is useful is that GP has found a regularity in the training set: global optima tend to be near the origin, and so that is an area of the search space that should be explored preferentially.

In order to compare the behaviour of the hand-designed and evolved PSOs, we tested them on 30 random problems taken from the city-block sphere and Rastrigin function classes for two and ten dimensions. The problems were generated by selecting the components of the global optimum c_i uniformly at random from the interval $[-C, +C]$, with $C = 1.0$ and $C = 2.0$. Note that **PSOG1** and **PSOG2** were evolved using the city-block sphere problem class, $N = 2$ and $C = 1.0$, and that **PSOG3** was evolved using the Rastrigin function problem class, $N = 2$ and $C = 1.0$. Thus, all other conditions represent off-sample test problems and are useful to assess the generalisation capabilities of these PSOs.

For each problem instance we performed 30 independent runs of each PSO. The results are reported in Tables 1 and 2. The tables show the average over the 30 problems and the standard deviation (in brackets) of the mean (over 30 independent runs) absolute error between the coordinates of the swarm best and the coordinates of the global optimum (i.e., $\sum_i |x_s - c_i|/N$) at the end of 30 PSO iterations. This gives an idea of how far the swarm best was from the global optimum in each dimension. Data in boldface represent the PSO with the best average performance in each condition.

On the two-dimensional city block sphere problem class, all PSOs do quite well, with the exception of **PSOR0** which cannot really be expected to do very well on unimodal functions. **PSOG1** is better than the standard PSO. However, to our surprise also **PSOG3** (which wasn't evolved on sphere functions) does better than the standard PSO (and in fact is even better than **PSOG1** for ten dimensions), suggesting **PSOG3** may be a good all-rounder. On the Rastrigin function problem class, **PSOG3** outperforms all other PSOs by a considerable margin. In ten dimensions **PSOG2** is second best, while in two dimensions **PSOR1** is second best. In other tests (not reported) where each PSO was run for 300 iterations instead of 30, we obtained essentially the same results.

Table 1. City Block Sphere. Normalised mean (standard deviation) of the distance between best location found by each PSO and global optima. Best results in bold.

N	C	**PSO**	**PSOD1**	**PSOR0**	**PSOR1**	**PSOG1**	**PSOG2**	**PSOG3**
2	1	.046 (.089)	.002 (.0002)	.26 (.029)	.003 (.0003)	**.001** (.0016)	.048 (.014)	.01 (.007)
2	2	.054 (.093)	.002 (.0002)	.27 (.036)	.003 (.0004)	**.001** (.0019)	.066 (.029)	.04 (.028)
10	1	.52 (.47)	.31 (.025)	1.6 (.021)	.27 (.022)	.45 (.025)	.65 (.023)	**.17** (.037)
10	2	.62 (.45)	.38 (.028)	1.6 (.036)	**.31** (.027)	.55 (.045)	.8 (.056)	.45 (.096)

Table 2. Rastrigin. Normalised mean (standard deviation) of the distance between best location by each PSO and global optima. Best results in bold.

N	C	**PSO**	**PSOD1**	**PSOR0**	**PSOR1**	**PSOG1**	**PSOG2**	**PSOG3**
2	1	.66 (.22)	.81 (.081)	.77 (.072)	.64 (.072)	.94 (.084)	.72 (.07)	**.28** (.066)
2	2	.71 (.2)	.85 (.098)	.79 (.066)	.66 (.091)	.94 (.12)	.75 (.12)	**.47** (.16)
10	1	1.2 (.35)	1.3 (.05)	1.9 (.042)	1.3 (.07)	1.3 (.055)	1.1 (.054)	**.59** (.057)
10	2	1.4 (.28)	1.4 (.073)	1.9 (.049)	1.4 (.064)	1.4 (.079)	1.3 (.063)	**.94** (.11)

It is interesting to compare the behaviour of **PSOG3** and **PSO**. Figure 1 illustrates the behaviour of **PSO** on a city block sphere function with $N = 2$. The particles tend to rapidly focus towards the global optimum at the origin. Figure 2 shows the behaviour of **PSOG3** in exactly the same conditions (including same starting positions for the particles). Here the particles tend to focus less rapidly.

Figures 3 and 4 illustrate the behaviour of **PSO** on a Rastrigin function with $N = 2$. Here the swarm controlled by **PSO** is quickly attracted towards a deceptive local optimum, while the particles in **PSOG3** perform bigger orbits and eventually discover and start converging towards the global optimum at the origin.

5 Conclusions

GP has been able to evolve a variety of particle swarm optimisers that work as well or considerably better than standard human-designed PSOs. Analysis of the evolved programs has led to new insights in the design of PSOs tailored for specific classes of landscapes.

To evolve our PSOs we have used the state of the art in GP, but we have not proposed a great deal in terms of new GP technology. However, in the more general context of machine intelligence, this work represents an important step within a new research trend: using search algorithms to discover new search algorithms. This approach has become possible thanks to the growth in computing power. We can already foresee that the results of this may be spectacular (see, for example, the case of Fukunaga's award winning work on evolving human-competitive SAT problem solvers [6]). The main contribution of this paper is to show that genetic programming can evolve better than human-designed PSOs in a few hours on a standard PC. A second contribution is to give us new ideas

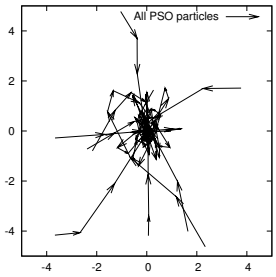

Fig. 1. Trajectories of the particles in one prototypical run of **PSO** on the 2–D city-block sphere problem.

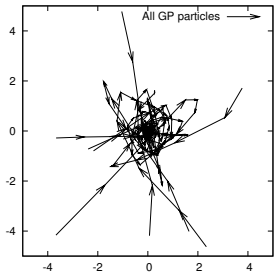

Fig. 2. A run of **PSOG3** on the 2–D city-block sphere problem (same initial conditions as in Fig.1).

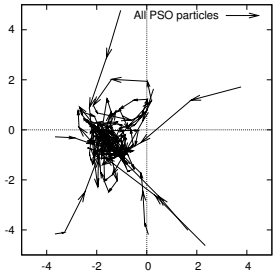

Fig. 3. A run of **PSO** on the 2–D Rastrigin function problem (global optimum at the intersection of cross hairs).

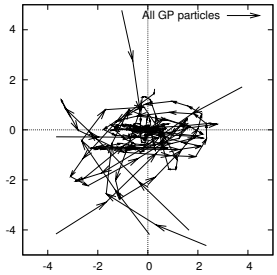

Fig. 4. A run of **PSOG3** on the 2–D Rastrigin function problem.

on what types of particle behaviours are most appropriate for which type of landscape.

In future research we intend to apply the approach to a variety of problem domains (including real-world problems) and to extend it by allowing GP to use more information on the past history of the swarm to control the particles and by allowing the evolution of coupled force-generating equations. We also intend to explore the effects and benefits of using different performance measures for PSO evolution.

Acknowledgments

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

References

1. P. J. Angeline. Using selection to improve particle swarm optimization. In *IEEE World Congress on computational intelligence, ICEC-98*, pages 84–89, Anchorange, Alaska, 1998.
2. T. M. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 19–26, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
3. T. M. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In *Applications of Evolutionary Computing*. Springer, 2004.
4. R. Brits, A. P. Engelbrecht, and B. Bergh. A Niching Particle Swarm Optimizer. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, volume 2, pages 692–696, Orchid Country Club, Singapore, Nov. 2002. Nanyang Technical University.
5. M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.
6. A. S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 483–494, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.
7. F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In *The ubiquity of Chaos*. AAAS publications, Washington DC, 1990.
8. J. Kennedy. The behavior of particles. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on evolutionary programming*, pages 581–589, San Diego, CA, 1998.
9. J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.
10. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
11. T. Krink, J. S. Vesterstrøm, and R. Riget. Particle swarm optimisation with spatial particle extension. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1474–1479. IEEE Press, 2002.
12. W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
13. M. Lovbjerg and T. Krink. Extending particle swarm opimisers with self-organized criticality, July 11 2002.
14. E. Ozcan and C. K. Mohan. Particle swarm optimization: surfing the waves. In *Proceedings of the IEEE Congress on evolutionary computation (CEC 1999)*, Washington DC, 1999.
15. R. Poli and C. R. Stephens. Constrained molecular dynamics as a search and optimization tool. In *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 150–161, Coimbra, Portugal, 5-7 Apr. 2004. Springer-Verlag.
16. J. Riget, J. S. Vesterstrm, and K. Krink. Division of labor in particle swarm opimisation, July 11 2002.
17. Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 69–73, Piscataway NJ, 1999.
18. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.