The Particle Swarm Algorithm

Tim Hendtlass

Centre for Information Technology Research, Swinburne University of Technology, Hawthorn VIC 3125, Australia, thendtlass@swin.edu.au

1 Introduction

Many algorithms are the result of biological inspiration and particle swarm optimization (PSO) is no exception. However, the PSO algorithm has slightly different end goals to the biological behavior that provides its inspiration and so needs to differ from its biological inspiration in some, perhaps non-biological, ways. PSO takes its inspiration from the flocking of birds and fish. In the real world, the flock needs to be compact for protection, and once food is found the flock should settle to feed. In the artificial particle swarm optimization the aim of the algorithm is to find an optimum solution to some problem, rather than the protection or food sought in the natural environment. For PSO the correct behavior once an optimum is found is not for all the particles in the swarm to converge on this, possibly local, optimum as the goal is to check *many* optima in the hope of finding the global optimum. Instead of converging, once an optimum has been found, it should be noted and the particles should immediately disperse to look for another, perhaps better, optimum. In Nature the time will come when a swarm that is feeding has consumed the food so that the place is no longer optimal: if swarming for protection the threat may change or even disappear completely. Then the swarm will again set out. Such extended periods of convergence serve no useful purpose so far as an artificial swarm is concerned.

If we model our PSO algorithm too closely on the behavior of birds and fish we run the risk that we will achieve those aspects of the natural behavior that we don't want at the expense of the artificial behavior that we *do* want. While retaining (possibly modified versions of) components that give the natural swarm its efficient search capability, we should be prepared to add such non-biological components as necessary so as to modify the natural behavior into the type of behavior we desire. This Chapter is concerned with developing the ideas behind a range of PSO algorithms, ranging from the simple (which loosely models real life and so has substantially real life behavior) through a series of progressively less biologically plausible algorithms that enable us to achieve useful and practical, but un-biological, aims. The basics are the same for all these algorithms. Since the extra abilities generally come at extra computational cost, this Chapter will describe the strengths and limitations of each so that the reader can make an informed choice as to which might be best for any particular problem.

2 The Basic Particle Swarm Optimization Algorithm

In all particle swarm algorithms the position of a particular particle in some way encodes a possible solution to the problem for which we seek, ideally an optimal, but at least a very good solution. Particles move under the combined influence of a number of factors in such a way that they tend to converge towards an optimum. A number of slight variants of the particle swarm algorithm have been proposed since the original algorithm was introduced by [19]. All of these try to balance several aspects of the behavior. See, for example, [5, 15, 23]

For an optimization algorithm to be more efficient than a random search the choice of the next position to be searched must be guided by the positions previously tested, and how good a solution those positions represented. Some way must be found that allows this previous knowledge to be exploited so that, on average, the next positions explored represent better solutions than would have been found if the next positions had just been picked randomly. Obviously storing all positions previously explored would rapidly produce an unsupportable demand for memory, and manipulating them in any way an insupportable computational load. So in the PSO only a few – often only two – kinds of good positions are kept by each particle, and probably some (and possibly all) the information about a particular particle's stored positions is shared with some or all of the other members of the swarm. There are a number of candidates for the types of position to keep:

1. All particles try to exploit (are influenced to move towards) at least one good position already found by some particle in the swarm. Often the position that is exploited is the best position yet found by any member of the swarm, a position known as the 'global best' or *gbest* position. This obviously requires communication between the members of the swarm sharing the best position each has found and forms a sort of social collective memory of the current global best *gbest* position.
2. Rather than using all the members of the swarm, sometimes each particle uses the best position – *lbest* – found by some subset of the swarm, the subset to which this particle belongs. Often this is defined as the N physically closest particles (in problem space) to this particle, but also the N closest in terms of index can be used. For example, if this particle has index 7 in

the list of particles, particles 5, 6, 8 and 9 might be the other members of its subset. Although for this latter option the particles may be far apart in problem space initially, attraction between the subset members has the effect of moving them closer together over time so that the net effect is quite similar to that of using the N physically closest particles. Not having to calculate the distance between each pair of particles can be a significant saving in time. Either the best current or the best position ever found by all the particles that make up the subset can be used. While this information is only directly shared among members of the subset, each particle can be a member of many subsets and so information in time is spread across the swarm.

3. The last position commonly used by a particle is the best position yet found by this particle – *pbest*. Since each particle is attracted to its own personal best position this does not depend on any inter-particle communication at all (although of course this is the same information that is shared as part of the calculation of whichever of *gbest* or *lbest* is also being used).

All particles are assumed to have mass so that they cannot change direction instantaneously. Furthermore, each time their velocity is updated it must contain a component that is in the same direction as their previously calculated velocity. This effectively provides a low pass filter to any change in their velocity, and smooths the track the particle follows through the problem space. The fraction of the previous component that is retained is called the 'momentum' and is a number greater than zero but less than one.

With only the use of *momentum* and *gbest*, particles would engage in a headlong rush toward the first reasonable position found by any particle, only changing this if some particle happens upon a better position during this rush. Eventually all particles would reach the same best-known position and exploration would stop. The particles would in time come to rest as, once a particle overshot a position, it would be attracted back to it and, with the momentum term being less than one, the velocity would drop with each reversal. This behavior would mimic real life birds settling at the best-known food source, but little exploration would occur during the headlong convergence and this would represent the balance between exploration and exploitation being tipped firmly towards exploitation.

With only the use of momentum and *lbest*, the behavior would depend on whether each particle is sharing the current best or the best ever position it has found. If the best ever is being kept the swarm could tend to divide and small groups of particles converge on a number of different positions. Particularly in the early stages the exact membership of the subsets would change, quite possibly changing the position towards which the current subgroup's particles are being attracted. This is less social than using *gbest* and more exploration will occur as the sub-swarms move towards their (generally) individual convergence positions. The balance between exploration and exploitation is still tipped strongly towards exploitation, but not as strongly as using *momentum* and *gbest*.

If using *momentum* and *lbest* and the position recorded is the current best, then the position that the particles in the sub-swarm are converging on is likely to change often, but can as easily change to a lesser fitness as to a greater fitness. With no collective memory of good positions found, the final position where the swarm members end up is largely a matter of chance. The balance between exploration and exploitation is now tipped quite strongly towards exploration, but some small exploitation occurs as the particles that make up a sub swarm share their current but not their historic information.

With only the use of *momentum* and *pbest*, each particle will end up on the best position it has happened across during its travels. While this obviously means that there will be some, probably many, positions explored by the swarm with poorer fitness than those positions on which the swarm members finally end up, there is no guarantee that the final positions will be good in a global sense. This is pure exploration with very little attempt at exploitation at all.

It is when *momentum* is combined with an attraction to two of these positions that the performance of the swarm improves sufficiently to make it attractive as a practical optimization algorithm. Which two you choose determines how greedy the algorithm would be.[1] A very greedy algorithm will converge fast, but not necessarily to a good position: a less greedy algorithm will converge more slowly, possibly *much* more slowly, but the probability that it will converge to a good position is enhanced.

The general form of the equation that governs the updating of each particle's velocity is given as:

$$\overline{V}_{T+t} = M \times \overline{V}_T + (1-M) \times \left(\left(\frac{G \times R_1 \times (\overline{Gbest} - \overline{X}_T)}{t} \right) + \left(\frac{L \times R_2 \times (\overline{Lbest} - \overline{X}_T)}{t} \right) \right) \tag{1}$$

or

$$\overline{V}_{T+t} = M \times \overline{V}_T + (1-M) \times \left(\left(\frac{G \times R_1 \times (\overline{Gbest} - \overline{X}_T)}{t \times \mid \overline{Gbest} - \overline{X}_T \mid} \right) + \left(\frac{L \times R_2 \times (\overline{Lbest} - \overline{X}_T)}{t \times \mid \overline{Lbest} - \overline{X}_T \mid} \right) \right) \tag{2}$$

In both of these equations, the parameter M is the momentum of the particle [0..1] and controls how fast the particle is able to respond to the

[1] A very greedy algorithm is one which attempts to optimize each step without regard to the final result; a less greedy algorithm is prepared to make one or more less optimal steps for now in the hope that these would set it up to make a 'killer' move later on.

two positions of attraction. A high momentum will make the particle slow to respond, which encourages exploration (and may enable the particle to move through small local optima entirely); conversely a low momentum makes the particle very quick to respond. Each of the two points of attraction has a constant that sets its maximum importance (G and L, respectively) but the two random numbers (R_1 and R_2 both in the range 0..1) introduce a stochastic element into the actual attraction at any time. The introduction of the $1/t$ terms is to ensure that the dimensions of each part of the equation are the same, namely the dimensions of velocity. The units of time are usually arbitrary, allowing t to be set to one and so saving it having to be written, but this time (whatever its value) is of great significance and has been included in Eqn. (1) deliberately so it is not overlooked. It must be remembered that, unlike real life, the algorithm assumes that a particle's velocity remains unchanged between fitness evaluations. A particle may travel a considerable distance, and even pass through several possible optima, between evaluations if t is large. Further implications of using a finite value for t will be discussed below.

Commonly the attraction to each position is also made dependent on the distance the particle is from the point of attraction: the further the particle is away, the higher the attraction. This is the form of the above update equations. While this can work well in simple problem spaces it may become unhelpful in more complex problem spaces. When a particle is far from a point of attraction, the acceleration of the particle towards that point of attraction would be high. By the time the particle reaches the point of attraction the particle's speed would be very high and so the distance it travels between evaluations is also large. An alternate approach (as expressed in Eqn. (1)/(2)) is to make the attraction to a point independent of the distance it is from that point. Although a particle will still accelerate as a result of the continued attraction to a point, this approach has the effect of limiting the maximum velocity particles can have and thus the maximum distance it can travel between fitness evaluations.

A limiting speed may be introduced for use with either version of the update equation. No particle is permitted to go faster than this limiting speed, if the magnitude of the right hand side of Eqns. (1) or (2) is greater than this limiting speed, the new speed of the particle is set to this limiting value but with the direction calculated from the equation. However the initial acceleration will be faster than if Eqn. (1)(Eqn. (2)) as stated above is used, and more of the journey will tend to occur at this limiting velocity, reducing the number of fitness evaluations along the journey.

To complete the description of Eqn. (1)(Eqn. (2)) it only remains to discuss which pair of attraction points to use. Originally, *gbest* and *pbest* were used, the former providing the social exploitation and the latter the personal exploration. Replacing *pbest* with *lbest* is a viable alternative as long as the size of the neighbourhood remains a modest fraction (say 10%) of the total swarm. If this is done the algorithm is made slightly greedier but a significant amount of exploration is still undertaken. Using *lbest* and *pbest* is the least greedy version of all and provides the least exploitation and the most exploration. The swarm may finally converge owing to the overlap between the sub-swarms of each of the particles.

Having calculated the new velocity for each particle the only other step in the algorithm is to move each particle ready for the next iteration. As mentioned above, it is assumed that the velocity of the particle does not change during the time t between updates (and evaluations). The position update equation is given as Eqn. (3), again with the inter-evaluation time t explicitly shown so as to make the equation dimensionally consistent.

$$\overline{X}_{T+t} = \overline{X}_T + t \times \overline{V}_{T+t} \tag{3}$$

2.1 Pseudo Code Algorithm for the Basic PSO

Algorithm 4 Basic Particle Swarm Optimization Algorithm

```
1. Randomly assign each particle to some point in problem space with some (small)
random velocity and evaluate the fitness of each particle at its current position.
while no stopping condition is met do
    2. Update gbest for the swarm, pbest for each particle (if these are to be used).
    if lbest is to be used then
        find the other particles that make up the sub-swarm of each particle, and
        update lbest for each particle from either the current best or the personal
        best fitness of it and its neighbours.
    3. Calculate the new velocity of each particle using Eqn. (1)/(2) (if not using
    either gbest or lbest, replace then with the position you are using).
    4. Move each particle using Eqn. (3) and evaluate its fitness at this new position.
```

The stopping condition could be achieving acceptable performance OR the swarm having converged without achieving adequate performance OR some maximum number of fitness evaluations having been made without either of the first two conditions being met. A swarm has converged when all the particles are (eventually) at the same position and the velocity of each particle is approaching zero.

3 Enhancements to the Basic Particle Swarm Algorithm

3.1 Constriction Factors

The whole right side of Eqns. (1) and (2) can be multiplied by a constriction factor. The purpose of this factor is to drop the average speed of the particles as time goes on. By doing this one of the problems with using a finite time between updates (t) can be addressed. If a particle is travelling fast it can cover a considerable distance in problem space in this time and may well pass over a point of interest without observing it, as no evaluation was made. This risk may be acceptable when the swarm is dispersed as the aim is to find regions of interest and there are probably other reasonably interesting points in the vicinity of this unobserved one. However, this is not so acceptable when the swarm is settling and trying to find the best point in a more restricted region as it will result in many unnecessary crossings and re-crossings of the best point thus increasing the time taken to find the best point.

The constriction factor could be just a fixed number less than one, but finding a suitable value for this number *a priori* is not easy. A better approach [8,9,11] takes into account the current behavior of the swarm and adjusts the constriction factor accordingly.

3.2 Adding Controlled Diversification

The tendency of a swarm to converge is beneficial when the swarm is exploring in the vicinity of an optimum but may well be counter productive if the swarm as a whole is still in transit searching for a suitable point to explore. Then it would be desirable for the swarm to diverge so as to explore more territory. Comparing the average velocity and the average speed of the swarm can help identify these two situations; if the average speed is significantly higher than the average velocity then it is reasonable to assume that the swarm particles are around a region of interest but approaching it in different directions. However, if the average speed and the average velocity have similar non-zero magnitudes then the swarm is probably sweeping through problem space. In this latter case deliberate measures can be taken to counter the natural tendency of the swarm to converge so as to more efficiently search the problem space – for more details see [17].

3.3 Handling Problem Constraints

Problem constraints can take many forms, for example possibly limited ranges of values for one or more variables or regions of problem space that correspond to non-viable solutions.

The first of these can be addressed by only allowing problem space to be of finite duration in the direction that corresponds to that particular variable. Any movement past the limit results in the particle re-appearing at the beginning. Effectively the axis is no longer an infinite straight line but a circle. This ensures that only permitted values of this variable are ever explored. The formula to update the position is no longer complete in itself; it has to have two conditional clauses added. For example, if the range of values is from 2 to 8, the clauses would be:

- **if** position ≥ 8 **then** position = position – 6
- **if** position is < 2 **then** position = position + 6

It might be necessary to apply these conditional clauses repeatedly until the position was within the meaningful range. The calculation of the distance between two particles would now have to take into account the fact that, for this dimension at least, the shortest distance (the one that should be used) could be in either direction around the circular axis. Should the acceptable values for this variable form a series of discontinuous ranges (say, 2 – 5 and 8 – 12) the range of the axis should be set to be the sum of these ranges ($3 + 4 = 7$ in this example). A mapping needs to be made between the position of the axis and the value this represents before the fitness is evaluated. For the example given here this would be:

- **if** position ≥ 0 and < 3 **then** value = position + 2
- **if** position is ≥ 3 and < 7 **then** value = position + 5

The distance between two particles would be calculated as described above, that is without taking into account the mapping.

The second constraint can be accommodated by a small change to the rule to update the global best and local best positions. A particle can only update these if its value is better AND it is in a region of problem space that the fitness function reports as being feasible. A particle can only include in its neighbourhood other particles that are currently in feasible space. It is possible that a particle itself and all other particles in infeasible space, in this case there is no local best and the contribution of the term involving the local best to the velocity update (Eqn. (1) or (2)) is set to zero. The momentum of the particles, useful in helping them sweep through local optima, will also help them sweep through regions of non-feasible space. Should there be no particle in valid problem space at the first fitness evaluation there will be no global best position and that term in Eqns. (1) and (2) will also be zero, and all particles will move in straight lines. However as soon as some particle finds a region of feasible space it can update and the convergence process will start. A problem space that is mostly infeasible would probably form quite a challenge, but more feasible problem spaces can be handled, as described above.

4 Particle Swarm Optimization of Multiple Optima

The performance of the basic particle swarm algorithm is good for problems with one best global optimum as long as the total number of optima does not get too high; ‘good’ in this context means both in terms of the quality of the results obtained and in the speed with which they are found. The quality of the solutions found for a given problem is comparable with those found by a Genetic Algorithm (GA), but the time take to find them is typically one tenth of that required.

However, not all problems fit into this simple category. Modifications can be made to the basic swarm algorithm to produce variations that are suited to at least some more complex classes of problem.

4.1 Exploring Multiple Optima

Many problems may have a number of optima whose fitness is similar. Allowing the swarm to converge to just one may not be the best move. Firstly there is no guarantee that convergence will occur to the global best optimum, indeed practical considerations may make the choice as to which is the best require considerations that are over and above just the numeric fitness value. Under these conditions we would like the algorithm to find several good optima and allow someone (or something) with knowledge of the broader picture to make the final choice.

An example might help. Suppose that the problem being solved is to find a good manufacturing schedule. When selecting the schedule for Thursday (say), the manager may know that certain members of their workforce would have attended a lively social occasion on Wednesday night. As a result it might be better to give them a lighter load on Thursday morning while they overcame the effects of the previous night. Given a range of possible schedules they might choose one that was likely to have the best practical outcome (given the extraneous factors) rather than chose the one with the best theoretical outcome. There could be many possible outside factors occurring too infrequently to be worth building into the fitness calculation for possible schedules. Better to give a range of good possibilities and let the human use their extra knowledge when making the final choice.

There are two basic approaches: if the number of good optima is small we might like to quickly perform a parallel exploration of these, with subsections of the total swarm each exploring a different optimum. Of course the absolute (and impractical) limit to the number of optima that can be explored in parallel is the number of particles in the swarm. This is impractical as a sub-swarm of one particle cannot use any social exploitation at all. If the number of potential optima that should be explored is large, then rather than exploring them all at once we would explore them in turn (serial exploration). Both of these approaches will be considered in this Section.

4.2 Achieving Parallel Exploration of Several Positions of Interest (niching)

The aim here is for the swarm to break automatically into sub-swarms and for these sub-swarms to converge onto different optima. While easy to state, achieving a practical realization presents many problems. If *gbest* is used the swarm will tend to converge all to one point. If *gbest* is replaced by *lbest* how many optima the swarm finds depends on the degree of overlap between the local regions from which best is derived. This degree of overlap has proved hard to control.

An alternate, and more successful, approach has been to develop a niching PSO [13: 67–69] that starts with a single main swarm and finishes with a number of smaller independent swarms. The particles in each of the smaller swarms communicate only with other members of its own swarm; there is no communication *between* swarms. The main swarm is generally trained using only the *pbest* position. Once some member of the main swarm is deemed to be in the vicinity of an optimum, a sub-swarm is formed by grouping together a number of particles in close proximity to the potential optimum. These communicate only with each other so as to further explore the optimum – now also using *gbest* attraction. These chosen particles refine the absolute best position, never leaving the vicinity of the optimum. Meanwhile the members of the main swarm that were not chosen for this sub-swarm continue using only *pbest* and look for another optimum (or at least a place that seems worthy of closer study).

Ideally the sub-swarms would never come close and no member of the main swarm would wander into a region being explored by a sub-swarm. In reality, of course, both things do happen. In the first case the two sub-swarms are merged (become aware of the performance of each other's members), which at the cost of using more particles may provide a more thorough exploration of the local space. In the second case the particle is just recruited to (joins) the sub-swarm it is moving through.

The niching PSO can explore a few different regions but since the particles that form a sub-swarm never leave the region they are exploring, the maximum number of regions that can be explored is set by the size of the original main swarm and by how many particles are recruited to form each sub-swarm. As the number of potential regions that require investigation increases, parallel exploration will obviously become quite inefficient.

4.3 Achieving Serial Exploration of Many Positions of Interest (WoSP)

An alternate approach that uses Waves of Swarm Particles (WoSP), introduced by [14], achieves serial exploration of an, in principle, infinite number of positions of interest. Actually it is not strictly serial as at any given time a small number of regions of interest are typically being explored in parallel. This behavior is achieved by reinforcing the tendency to totally converge rather than trying to slow or even inhibit total convergence, but once they have converged forcing particles to be ejected with significant velocities so that they carry on searching for other optima. This behavior is achieved by adding an extra short-range force of attraction to the basic swarm equation (as shown in Eqn. (4)) and making use of the finite time between velocity updates for particles.

$$\overline{V}_{T+t} = M \times \overline{V}_T + (1 - M) \times \left(\left(\frac{G \times R_1 \times (\overline{Gbest} - \overline{X}_T)}{t\times \mid \overline{Gbest} - \overline{X}_T \mid} \right) + \left(\frac{L \times R_2 \times (\overline{Lbest} - \overline{X}_T)}{t\times \mid \overline{Lbest} - \overline{X}_T \mid} \right) \right) + \overline{SRF} \tag{4}$$

where $\overline{SRF}$ is the net short range force acting on this particle. The ith component of the short range force exerted on particle x by particle y is given by

$$SRF_{xyi} - SRF^{factor} \times \frac{V_{xyi}}{D_{xy}^{SRF_{power}}} \tag{5}$$

where V_{xyi} is the ith component of the velocity of particle x with respect to particle y, D_{xy} is the distance from particle x to particle y, SRF_{factor} is the magnitude of the short range force at unit distance, and SRF_{power} sets how fast this force decreases with distance.

As a result of the discrete way in which fitness evaluations and updates to the velocity of the particles is done, an aliasing effect causes pairs of particles to approach, pass each other and then continue at very high velocities. The closer particles approach each other the higher the probability of this happening. There is no longer a need to try to stop the particles fully converging; once converged this aliasing effect will cause particles to be ‘ejected’. As the velocity with which the particles leave the swarm is variable, exploration can continue both locally and at a distance.

The way in which this aliasing effect works is as follows. As particles approach each other the magnitude of the short-range force will increase significantly, producing a substantial increase in the velocity of the particles towards each other. For discrete evaluation, *by the time of the next evaluation,* particles may have passed each other and be at such a distance apart that the short-range attraction that might bring them back together is far too weak to do this. As a result, the particles will continue to move rapidly apart with almost undiminished velocity, exploring beyond their previous positions. This process is shown in Fig. 1. The ‘snapshots’ are taken starting at some arbitrary time T (at the top), with the lower ‘snapshots’ being taken progressively later. The time interval between ‘snapshots’ is the basic time interval for the PSO and is represented by t.

At time T, the separation between the particles is moving into the region in which the magnitude of the short-range attraction (shown by broad arrows) is becoming significant. This augments the effect of their velocities (shown by thin arrows) so that the particles move close together. By time $T + t$ the particles are close and the short-range effect is large. As a result, the velocity

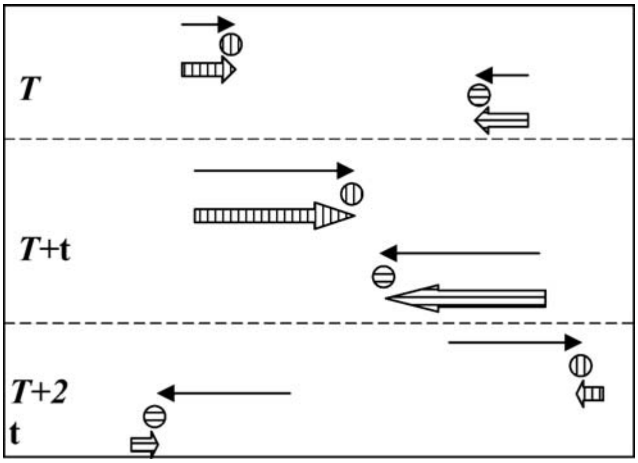

Fig. 1. A series of 'snapshots' showing the two particles (shown as circles), their velocities (thin arrows), and forces (thick arrows)

of the particles increases substantially, almost entirely as a consequence of the short-range attraction. By time $T + 2t$ when the next evaluation is made the particles have passed each other, and are so far apart the short-range force is weak. Consequently, the particles continue to diverge, retaining at $T + 2t$ much of the velocity obtained as a result of the short-range forces acting at time $T + t$. The short-range forces will continue to decrease as the particles move apart, leaving only the normal swarm factors to influence their future movement in the absence of other close encounters.

The total swarm automatically becomes broken into a number of sub-swarms called 'waves', each with its own *gbest* value. Particles that leave a converging swarm as a result of this aliasing effect leave the wave they were in and are 'promoted' to join the highest numbered (most recently created) wave. Should the particle already be part of the highest numbered wave, the particle becomes the founder member of a new 'most recently created' wave. Importantly, a form of evolution is introduced by making all particles from some lower performing wave be compulsorily recruited into a better performing higher numbered wave. Waves that run out of particles (owing to promotion or recruitment) die out. In this way there is a continual automatic updating of best position information available to the successive waves.

For static problems[2] each particle keeps a tabu list of places that it has already explored and is repelled from any place on its tabu list. In this way re-exploration of any point in problem space is largely (but not totally) eliminated and much pointless computation saved.

[2] The minor changes to suit WoSP to dynamic problems, ones that change even whilst the optimization algorithm is running, will be discussed later.

5 Controlling the Computational Expense

Whichever version of PSO is used, every time a particle moves to a new position the fitness of this particle at this position has to be calculated. For real life complex problems this fitness calculation may well dominate the computational load of the algorithm. Although the PSO lends itself to being run on a number of computers in parallel (for example each particle's current fitness being evaluated simultaneously by a different computer), for really complex problems even this approach is not adequate and it behoves us considering ways to improve the computational efficiency of the PSO algorithm. Two possible computational enhancement possibilities will be mentioned here.

5.1 Using a Dynamic Swarm Size

Since the fitness of each particle has to be evaluated for every iteration of the algorithm, one possible approach is to limit the number of particles. While swarms do not need to be large, too few swarm members will limit the search capability and thus the average quality of the results obtained. However, the number of particles does not need to be constant. When the swarm is converging and the particles get very close together it may be a waste of resources to support so many particles. Some particles could simply be 'switched off' and take no further part in the swarm thus saving their evaluations. Of course, the monitoring required and deciding when such action should be taken, must not add so much computational cost as to negate the computational advantage we seek to gain. In addition, knowing precisely when and which particles should be switched off is in itself a non-trivial matter.

5.2 Fitness Estimation

It is possible to avoid having to measure the fitness at every position if you can instead estimate it. This has been shown to work for genetic algorithms [23] and a variation has been shown to work for PSO for a range of problems [12]. The idea is to estimate the fitness of a particle at a new position using the fitness of this particle last time it was estimated or evaluated together with the fitness of the particle that last iteration was closest to this new position. These fitnesses may be estimates themselves and may have been estimated from other fitnesses that were themselves estimates. Obviously a fitness based on estimates that were based on estimates, and so forth, would not be very accurate and so a new parameter is associated with a particle's fitness – its 'reliability'. This is set to one if the fitness was found as a result of a true evaluation, and this figure is decreased every time an estimation of the fitness is made. When a fitness drops below a threshold, the estimation is discarded and a true evaluation done, returning the reliability to one.

The formulae for estimating the fitness and reliability of the particle in the new position (F_{new} and R_{new}) in terms of the fitness and reliability of the two closest positions to its new position during the last iteration (F_1, R_1 and F_2, R_2) are:

$$F_{new} = \frac{W_1 F_1 R_1 + W_2 F_2 R_2}{W_1 R_1 + W_2 R_2} \tag{6}$$

and

$$R_{new} = \frac{(W_1 R_1)^2 + (W_2 R_2)^2}{W_1 R_1 + W_2 R_2} \tag{7}$$

where W_1 and W_2 are the relative weightings to be placed on the two closest positions. These weightings are derived from the Cartesian distances between the new position and each of the two closest positions last iteration (D_1 and D_2):

$$W_1 = 1 - \frac{D_1}{D_1 + D_2} \tag{8}$$

and

$$W_2 = 1 - \frac{D_2}{D_1 + D_2} \tag{9}$$

Figure 2 shows the fitness values (and their reliabilities) that would be calculated for a simple one-dimensional case, where the values are derived from a fitness of 1 (reliability 0.8) at position 15 and another fitness of 2 (reliability 0.6) at position 25. Notice how the fitness and reliability matches at each known point. The fitness is linearly interpolated between the two known points and moves asymptotically to the average of the two fitnesses at points far from them. The reliability however falls away the further we move

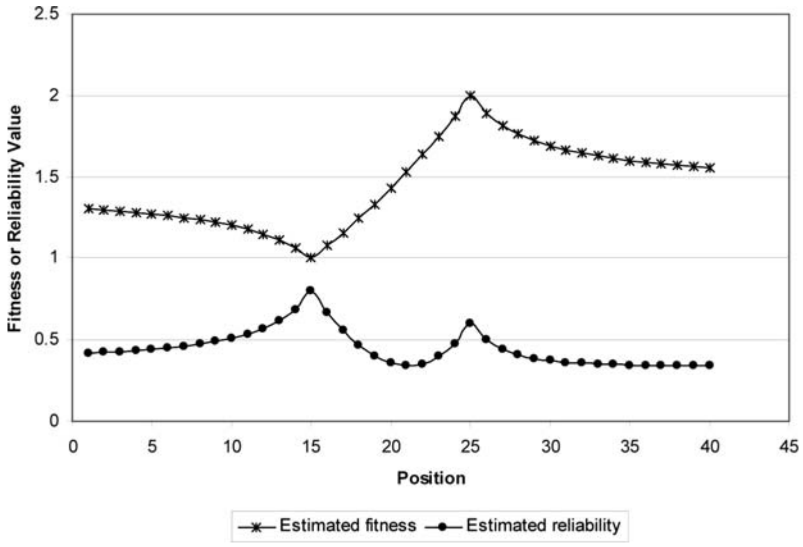

Fig. 2. An example fitness and reliability calculation in one dimension

from a known point – note how the value is influenced by the reliability of the closest known point.

The only new parameter introduced to the algorithm is the *reliability threshold* T [0..1] that is used to determine when a true fitness evaluation is required (a threshold of 1 would result in true evaluations every time as in a conventional PSO). The initial positions of all particles must be truly evaluated and their reliability set to 1. However, after this the following algorithm is used whenever a particle's fitness is required. Note that it only requires a list to be kept of the positions, fitness and reliability of all particles' last iteration. It would, of course, be possible to keep a list of positions, fitness and reliability triads derived from more than just the last iteration, but experience so far suggests that any modest increase in performance is not worth the computational cost.

Algorithm 5 PSO position, fitness and reliability algorithm

```
1. Record the distance this particle has moved since last iteration and this
particle's fitness and reliability last iteration.
2. Find the particle whose position last iteration was closest to this particle's
current position and record its fitness and reliability, together with the distance
from this particle's current position.
3. Using Eqns. (8) and (9) calculate the relevant weighing factors for each of these
two positions.
4. Using Eqns. (6) and (7) estimate the fitness and reliability of the fitness estimate
for this particle's current position.
if this reliability is greater than the threshold then
   keep these fitness and reliability estimates.
else
   discard the fitness estimate and perform a true fitness evaluation;
   set reliability of this fitness to 1.
```

Experiments have shown that there is no significant difference in the average performance at any number of iterations of the algorithm with or without fitness estimation. However, using fitness estimation would have required far fewer real fitness evaluations, a saving in time if the time for a true fitness evaluation is greater than the time required to find the closest particle and calculate the fitness estimate. This condition will be met for many real life practical problems, and so for these problems using fitness estimation will allow you to obtain essentially the same result in a shorter time.

6 Dynamic Optimization Problems

Dynamic optimization problems are problems in which the objective function being optimized changes in some way while the optimization is taking place. Obviously there will be some limit to the rate of change that can be tracked, but swarms are able to adapt to changing conditions as long as they are not too rapid. The changes can be divided into three groups [22]:

1. *The actual problem being changes alters as time passes.* For example, when scheduling the delivery of goods to multiple places the original aim of minimizing fuel usage is replaced by an overwhelming need to minimize the number of late deliveries.
2. *The components available to use in building the solution change.* To continue the same example used above, some delivery trucks break down and others are returned from repair and become available.
3. *There is a change to the constraints on the problem.* Still continuing the same example, some of the roads that might be used now have altered speed restrictions.

In practice however, much of the work in the area of dynamic problems has been done on function optimization problems in which the position and/or magnitude of peaks in the function vary with time.

When attempting to find and track optima in dynamic problems, the swarm behavior must become even more un-biological. It is a more biologically plausible scenario to have to track an optimum that slowly changes position (while remaining a good optimum) than to have to find the global optimum from a number of local optima that change their relative quality ranking (and possibly position). It is not surprising therefore that, provided the swarm is not allowed to fully converge so that the velocity of each particle is reduced to zero, a slowly moving peak can expect to be followed in the sense of there being a high probability that one or more particles will traverse sufficiently close to the moving peak at the time of a fitness evaluation that the new position of the optimum is discovered. A number of methods to ensure incomplete convergence will be described below but on their own none of these is a complete solution as some mechanism also has to be introduced to update the best fitness known based on a combination of the value found and the time at which it was found.

To observe the emerging eminence of a local optimum far from the position towards which the swarm was just converging requires some swarm particles to be exploring in the vicinity of this distant position. Various methods to achieve this will also be described below. Again the maintenance of a number of explorer particles is not a complete solution. The social factors (the best position found by each particle and by the swarm) must also be updated as the old information goes out of date.

6.1 Ways to Achieve these Adaptations

As suggested above there are a number of non-biological adaptations that need to be made to the classical swarm algorithm to suit it for dynamic problems. These can be summarized as:

- preventing the complete convergence of the swarm,
- keeping personal and social reference points up-to-date, and
- maintaining or generating explorer particles far from any current point of convergence.

Approaches that achieve at leat one of these aims will be considered.

6.2 Preventing Total Convergence

Social influences between particles – attractions to *gbest* and *lbest* – will clearly tend to result in total convergence. In order to change this it is necessary to introduce some counter influence.

One method, introduced by [1] is to give at least some particles a charge so that, by analogy with electrostatics, two particles would experience a repulsion force as they approached and the swarm would then not be able to fully converge. The particles would in time reach some (possibly dynamic) equilibrium between the convergence and divergence effects, but this does not mean that they are actively exploring.

A second method, introduced by [5], is to divide the swarm into sub-swarms so that not all particles are converging on the same point. As well as the main swarm, a particle and its closest neighbours may form a sub-swarm if the variance in the fitness of the particles is less than some threshold. Any particle that is not a member of a sub-swarm belongs to the main swarm. These sub-swarms may merge or acquire extra particles from the main swarm or collapse back into the main swarm. While developed for multi-modal functions this niching behavior could also be used, in principle, to limit total swarm convergence. However the algorithm depends on a uniform distribution of particles in the search space, a condition that may be able to be met after initialization but which is not met after convergence into the sub-swarms has taken place.

6.3 Refreshing the Best Positions

If an attraction to *pbest* is being used these best positions may be updated by allowing particles to replace their previous best position with the current position periodically [6]. Choosing a suitable period without knowledge of the problem being optimized can be problematic. If an attraction to *gbest* is being used then the fitness at this position may be periodically re-evaluated [2]. As the fitness at that point deteriorates, the probability that it will be replaced by another position as a result of the current fitness at that position increases. Again a suitable re-calculation frequency has to be chosen.

6.4 Forcing Explorer Particles

The simplest approach just requires that a number of particles be periodically moved to randomly chosen points and have their fitness re-evaluated [16]. Another approach organizes particles in a tree with each particle being influenced by the particle above it (social) and itself (best position and momentum). A particle swaps with the one above it if it out performs it. This gives a dynamic neighbourhood that does require extensive calculation. This has been adapted to dynamic problems by [17,18]. After the value of the best-known position (*gbest*) changes (it is re-evaluated every cycle) a few sub-swarms are re-initialized while the rest are reset (have their old personal best information erased and replaced with the current position). The sub-swarms then search for the new optimum.

[1] introduce a more elaborate approach using quantum particles. Using an analogy to quantum mechanics, a particle on measurement is placed randomly within a given radius of its net current point of attraction. A uniform (and very un-physical) distribution is used, but this could be changed so that there was not a uniform probability of the particle being at every distance, and a function chosen so that a finite probability exists of a movement to a distance far from the point of attraction.

6.5 Adapting WoSP to Dynamic Problems

WoSP, because of the sequential way it explores optima, is inherent suited to dynamic problems. All that needs to be changed is the removal of the tabu list that each particle keeps, recording the positions from which it has been promoted. While for static problems repulsion for these points makes sense, for dynamic problems a particular point in problem space may be a good optimum at several disjoined times and a poor optimum in between these times. Completely removing the list may be inefficient, it may be better to periodically review the entries on each particle's list. Extending the idea from [18], each of the previously explored optima on all the lists could be periodically re-examined and all points for which significant changes were found to have occurred in the fitness would be removed from the tabu lists of the particles. This would need to be done frequently and whether the reduced re-exploration would be worth the computational expense of this housekeeping is not clear.

7 Particle Swarm and Quantized Problem Spaces

So far all the descriptions of the PSO have been in terms of continuous problem spaces, indeed all the swarm update equations presented in this Chapter explicitly require a continuous problem space. However, this does not mean

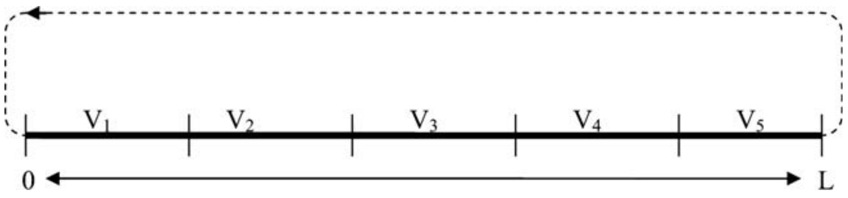

Fig. 3. An axis in PSO space suitable for a five value quantized parameter in problem space

that the PSO cannot handle other types of problems with other types of problem spaces. All that is required is the ability to map the continuously variable positions in the PSO particle space to the problem space. There are many problem spaces that are not continuous problem spaces and there is not room to describe possible mappings for more than one. The one problem space chosen is the quantized problem space, a space in which the values associated with some parameter are constant for a while and then change instantaneously to a new value. The values of this parameter are discrete (quantized) values. As an example of the mapping required, consider the case where a certain parameter can only have one of five values, it does not matter what the actual values are, we will refer to then as V_1, V_2, V_3, V_4 and V_5.[3] Let the length of the axis in PSO space be L. Figure 3 shows this axis.

Note that the axis wraps around, a particle reaching the end of the axis at L immediately reappears at position zero. Effectively, Eqn. (3) becomes:

$$\overline{X}_{T+t} = (\overline{X}_{T+t} \times \overline{V}_{T+t}) mod(L) \tag{10}$$

If L is made equal to the number of categories, the value that a particle's position corresponds to can be found by applying Eqn. (10) and then taking the integer part of the answer. However, there is obviously a length of L/N on an axis that will correspond to the same quantized value (where N is the number of values – five in this example). The fitness function is required to be continuous in the sense that adjacent positions should in general return different fitness values so as to provide guidance to the swarm as it converges. For this reason the distance of a particle from the closest cell centre may also be recorded (expressed as a fraction of L/N). The fitness function now becomes comprised of two parts, the 'normal' fitness function (F) and the average of these fractional distances across all axes in the problem (f). When deciding if one position is fitter than another the 'normal' fitness function parts of the total fitness (the two F values) are consulted first. If these differ,

[3] Each quantum value has been allocated an equal length on the axis in this example. This is not essential and different lengths could be allocated, each length being set proportional to the relative probability of this quantized value occurring, perhaps.

the answer is clear. However, should they be the same then the position with the better value of f is chosen.

8 Some Sample Results

8.1 Problems used as Examples in this Chapter

This Chapter has described a number of variants of the PSO algorithm and a number of problem domains to which they may be applied. Space will not allow results to be presented for every variant and every domain; indeed the limitations of fixed type do not readily allow the presentation of dynamic problem results in a simple and clear way. The results from four problems have been chosen in order to show the behavior of the basic and WoSP variants of PSO both with and without fitness estimation and on continuous and quantized problem spaces. These problems are described below.

Finding the Origin

The first of these is the apparently trivial problem of finding the origin, the fitness of each particle being its distance from the origin, as shown in Eqn. (11).

$$f = \sum_{i=1}^{100} \sqrt{(x_i)^2} \tag{11}$$

where f is the fitness and x_i is the ith component of the position of the particle.

This becomes quite hard as the number of dimensions increases for any algorithm that makes simultaneous updates to all dimensions (as the PSO does). For a new position to be more successful than the old position the net effect of all the changes in all the dimensions must be a decrease. As the number of dimensions increases this becomes harder, especially approaching the origin. Results will be presented for PSO seeking the origin in 100 dimensions.

Rastrigin's Function

The second problem is Rastrigin's Function Eqn. (12).

$$f = ((x_i)^2 - 10cos(2\pi x_i) - 10) \qquad x_i \in [-5.12 \cdots 5.12] \tag{12}$$

where f is the fitness and x_i is the ith component of the position of the particle.

This is a well known function commonly used as a test problem for optimization algorithms. This can be readily solved by a traditional PSO algorithm.

Schwefel's Function

The third problem is Schwefel's function [24] in 30 dimensions. This is another well known function commonly used as a test problem for optimization algorithms. It has a large number of local optima but one unique global optimum. No matter the number of dimensions, the position and size of this global optimum can be readily calculated, as can the positions and sizes of any local optimum.[4]

$$f = \sum_{i=1}^{30} x_i sin(\sqrt{\mid x_i \mid}) \qquad x_i \in [-512.03 \cdots 511.97] \tag{13}$$

where f is the fitness and x_i is the ith component of the position of the particle.

This problem is included as the chance of the conventional PSO algorithm finding the global optimum position is very low. This is because in 30 dimensions there are 1.2 $\times 10^{27}$ local optima that need to be explored. Sequentially exploring optima using the WoSP PSO variant gives an approximately 40% chance of finding the one global optimum [4]. Like many real life problems that also have many local optima, fitness evaluation now constitutes a significant fraction of the total computational load.

A Timetabling Problem

Finally a simple quantized problem space is used to illustrate how the PSO may solve this type of problem. The problem used is a simple timetabling problem involving scheduling nineteen classes for four groups of students in three rooms for one day of six time periods. While all classes can occur in any of the six available time periods there are various constraints as to the rooms each class may occur in and the group(s) of students that will be involved. The aim is to timetable the classes so that these constraints are met and no student is required to undertake two classes at once and no room is required to contain more than one class at a time. The constraint details are shown in Table 1, with the classes, room and groups identified by numbers.

There are approximately 1.7×10^{26} ways in which these classes can be arranged but only slightly more than 2,500 that meet all constraints. While this is a trivial problem as far as timetabling is concerned, it is more than adequate to explore the behavior of PSO particles in quantized problem spaces as it is easy to comprehend and has the advantage of fast fitness evaluation.

[4] The version of Schwefel's function given here is the form in which the value of the fitness function can be negative at some places in problem space. If it is more convenient for the fitness to always be positive (for example if in a GA with the breeding probability directly proportional to the fitness) this can be achieved by adding 418.9829 times the number of dimensions to the result given by equation 13.

Table 1. Timetable problem constraints

Class	Possible rooms	Groups involved
1, 2, 3	1, 2, 3	1
4	4	1
5, 6, 7	1, 2, 3, 2	
8	4	2
9, 10, 11	1, 2, 3	3
12	4	3
13, 14, 15	1, 2, 3	4
16	4	4
17	1, 2, 3	1, 2
18	1, 2, 3	3, 4
19	1	1, 2, 3, 4

Table 2. Parameter values used for each of the four problems

Parameter	Origin problem	Rastringin's function	Schewefel's function	Timetable problem
Particle count	30	30	30	30
M	0.5	0.9	0.9	0.9
G	0.5	0.9	0.9	0.3
L	0.5	0.5	0.5	0.7
Neighbourhood	Particle & 3 closest	Particle & 3 closest	Particle & 3 closest	Particle & 3 closest
Search scale	–	–	500	2
SRF^{factor}	–	–	5000	500
SRF^{power}	–	–	3.5	3.5

Two quantized variables were associated with each class, the time it is to be scheduled and the room it is to occur in. Each of these is mapped to a different axis in problem space. A total of 38 axes were therefore required to schedule these 19 classes. The number of possible quantized values these axes contain varies from 1 to 6.

8.2 Experimental Details

The values used for the parameters for each of these four problems are shown in Table 2.

9 Sample Results

All the figures below contain multiple plots, each corresponding to a different threshold. The concept of a threshold is really only meaningful when using some fitness estimation, but setting the threshold to one has the effect of not allowing any fitness estimation – the 'traditional' PSO. For thresholds below one, the lower the threshold value the higher the ratio of fitness estimations to true fitness evaluations.

9.1 Minimizing the Distance to the Origin in 100 Dimensions

The fitness values reported at a particular iteration are the average distance from the origin of all 30 particles in 100 independent repeats of the experiment.

There is some evidence on all the plots in Fig. 4 of two phases of activity, in the first of which fast progress is made. In the second phase (from about 1000 iterations onwards) progress is slower as the algorithm finds it harder to make a move that has a net beneficial effect on the fitness over all 100 dimensions. It could be argued that PSO (like other algorithms that simultaneously update all dimensions) is not a very suitable algorithm for this second phase.

Having a threshold of either 0.75 or 0.5 has little effect on the average best fitness per iteration compared to a threshold of one, despite the fact that the first two thresholds correspond to a mixture of fitness estimation and true fitness evaluation, and the last to only using true fitness evaluation. When the threshold is as low as 0.25, the average fitness falls more slowly. The fact that the estimated fitness can never be lower (or higher) than the lowest (highest) fitness of the two reference points from which it is derived makes it even harder for the algorithm to find points whose estimated fitness is better than the current $\overline{G}_{best}$ and $\overline{L}_{best}$. This may, at first sight, suggest that the fitness

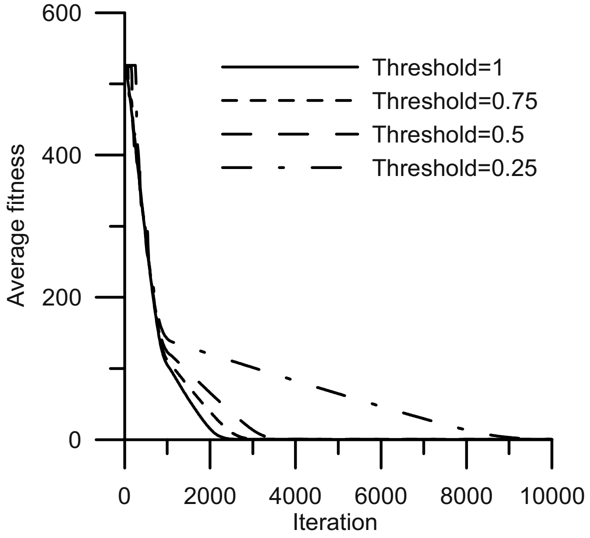

Fig. 4. Finding the distance to the origin in 100 dimensions as a function of the iteration

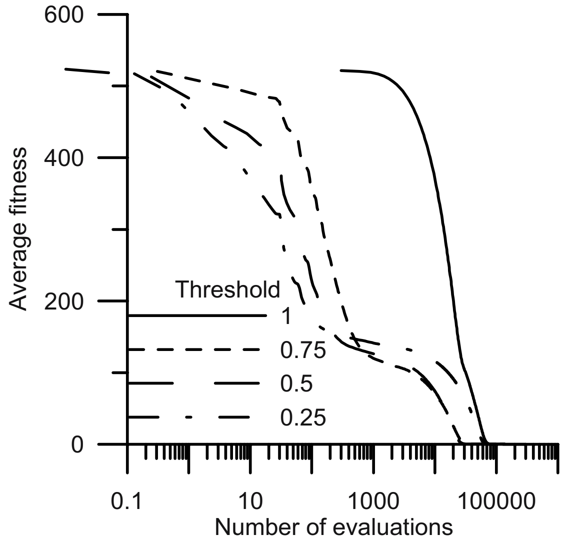

Fig. 5. Finding the distance to the origin in 100 dimensions as a function of the number of true evaluations as opposed to fitness estimates made (the fitness values reported at a particular iteration are the average)

estimation PSO algorithm is not particularly suited to problems containing regions of problem space that are smooth changes.

However, when the average fitness is plotted against the number of true evaluations as in Fig. 5, it becomes clear that for this problem it will take less true evaluations to achieve a given performance using fitness estimation than when not using it (if only marginally for a threshold of 0.25). The quality of the final solution is comparable in all cases. Had the time taken to do a true evaluation been significantly greater than the time taken to estimate the fitness (which is not the case for this particular simple demonstration problem), the overall result would have been less computing for results of comparable quality.

9.2 Rastrigin's Function in 100 Dimensions

Consider the bold line in Fig. 6 which shows the average best known fitness (averaged over 100 independent repeats) for a PSO solving Rastrigin's Function in 100 dimensions using only true fitness evaluation. While the gradient of the graph varies, progress is almost continuous, with only short periods of apparent stagnation. A genetic algorithm would be likely to show longer periods of apparent stagnation. Like a GA, the solid curve tends to plateau out in the vicinity of, but not actually at, the global optimum of zero. Once

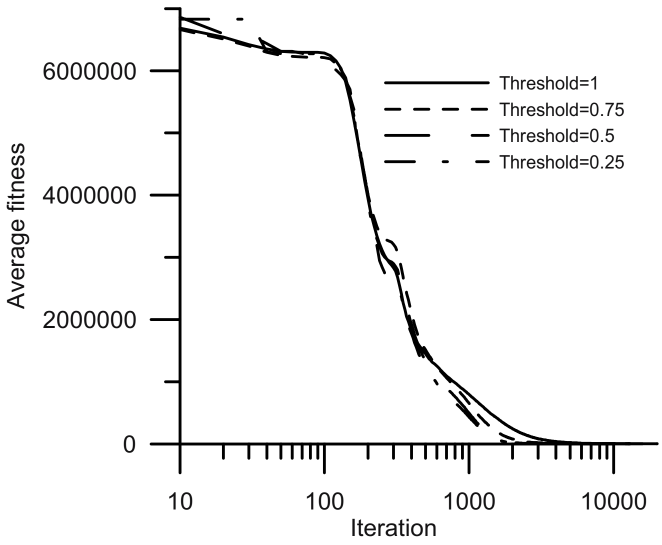

Fig. 6. Best known fitness per iteration (averaged over 100 independent repeats) for Rastrigin's function in 100 dimensions

Table 3. Final statistics for Rastrigin's function (averaged over 100 independent repeats)

Threshold	1	0.75	0.5	0.25
Average fitness	1553	1511	1525	1619
Standard deviation	221	180	121	153
Maximum	2544	2510	1899	2074
Minimum	1241	1166	1262	1314

in the vicinity of an optimum it is often better to switch to using a simple gradient descent algorithm for the local search.

Figure 6 shows that it would be hard to pick whether fitness estimation was being used (and, if it were, what value was being used for the threshold) if one just has the average best known fitness at each iteration (an iteration is all particles making one position update). The values presented here are for 30 particles and the average is over 100 independent repeats.

However, Fig. 7 once again shows that the performance as a function of the number of true evaluations differs substantially with the four threshold values. The performance with a threshold of one (no fitness estimation) is poorer than any of those that do allow fitness estimation.

Table 3 shows the average final fitness and the standard deviation for the four tested threshold values, along with the maximum and minimum final values found (a total of 100 independent repeats were done for each threshold

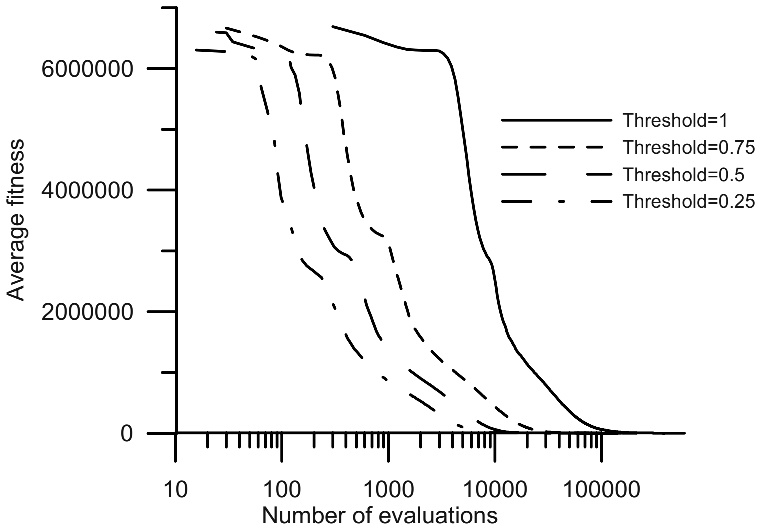

Fig. 7. The best known fitness per true evaluation (averaged over 100 independent repeats) for Rastrigin's function in 100 dimensions

value). As Rastrigin's Function is a minimization function (with a global best value of zero) the final values show that the exact optimum had not been located, although compared to the initial values of over 6,000,000 the particles had made significant progress in the number of iterations that they had been allowed (20,000). Again PSO, with or without fitness estimation, is not efficient in the final stages of converging to an optimum. Table 3 also shows that the threshold in use could not be identified from these end results in a blind test.

Figure 8 shows the ratio of the cumulative totals of the number of fitness evaluations to the number of fitness estimations for Rastrigin's function as a function of the iteration number. Note that whenever fitness estimation is being used this ratio is asymptotic to a number less than one. This means that the number of true fitness evaluations is always less, often significantly less, than one half of the number of true fitness evaluations that would be required by a PSO algorithm that does not use fitness estimation.

9.3 Schwefel's Function in 30 Dimensions

This function, with its many local optima, is highly problematic for a traditional PSO algorithm. However, it can be solved by successive exploration of optima using the WoSP variant of the PSO algorithm [14] The results presented here only show relatively early stages of this exploration (only the first 10,000 iterations, by which stage the best known result is well within the top thousandth of one percent of all results). However, given 200,000 iterations, WoSP has a 41% chance of finding the global optimum [14].

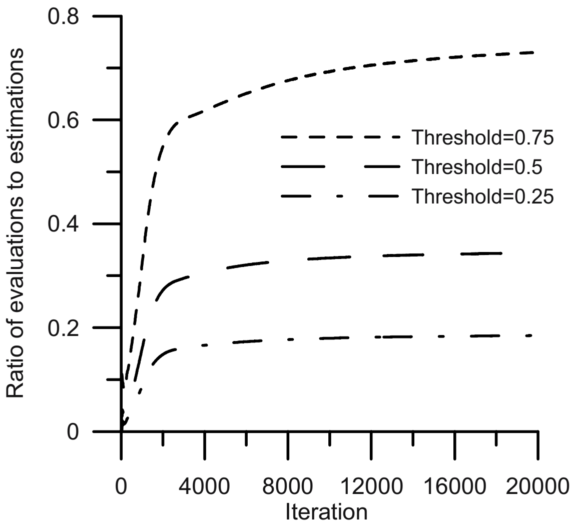

Fig. 8. The ratio of the total number of true fitness evaluations to total number of fitness estimations for Ratrigin's function plotted per iteration

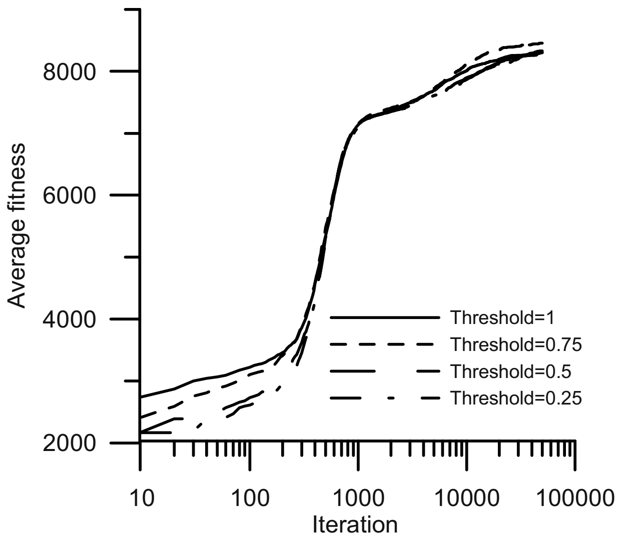

Fig. 9. The best known fitness per iteration (averaged over 100 independent repeats) for Schwefel's function in 30 dimensions

Although there is some difference in the early stages and a slight difference in the late stages, the plots of the average best known fitness versus iteration shown in Fig. 9 are very similar, especially during the time that the swarm is making good progress. Since this is a plot of the best optimum known, many

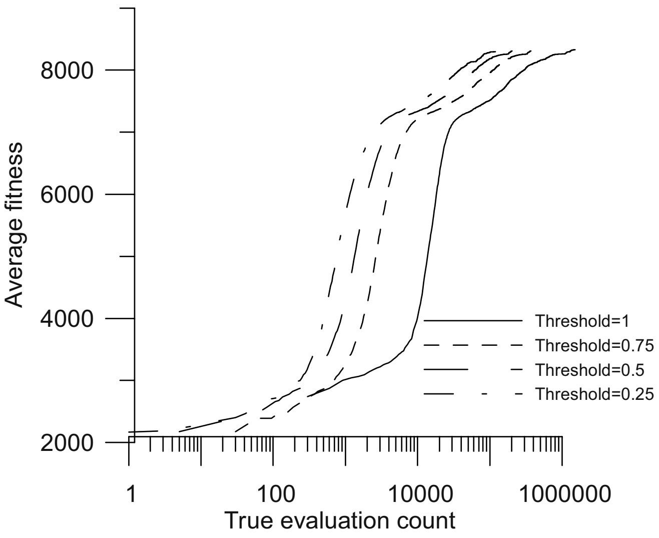

Fig. 10. Best known fitness per true evaluation (averaged over 100 independent repeats) for Schwefel's function in 30 dimensions

optima are explored without affecting the plot. Nevertheless progress is fairly continuous with few regions of apparent stagnation.

When plotted as the average best known fitness versus the number of true evaluations (Fig. 10), the advantage of using fitness estimation becomes clear. If there was only enough time to perform 10,000 true fitness evaluations, the best location found by the conventional WoSP PSO (threshold=1) would have, on average, a fitness of some 3500. Using a threshold of 0.25, the best location the WoSP PSO with fitness estimation would have found, on average, would have a fitness of about twice this.

The particular significance of this result is that it is achieved in a system that frequently moves particles far from the region(s) of problem space that have been explored. This underlies the importance of the reliability value associated with the fitness, and the way that this decreases, not only with the reliability associated with the two reference positions in use, but also with the distance of the new position from these two reference positions. This clearly demonstrates that the fitness estimation algorithm not only decreases the number of fitness evaluations required but also is reasonably efficient at deciding when true evaluation is really required.

Results from a Simple Quantized Problem Space

Because of the large number of ways that the classes can be arranged, attempts to solve it with the traditional PSO algorithm were, as expected, highly ineffectual (as will be seen from Table 5). However, the WoSP variant of the PSO was able to solve the problem. Details of the basic parameter values used are shown in Table 2. It is not claimed that the values used were optimal but they do follow the general guidelines given in [15]. Each swarm was randomly initialized and then allowed 1,500,000 evaluations after which the best solutions found were recorded.

A simple problem specific local heuristic was used that took a solution with one or more constraint violations and repeated the following set of steps until either the number of constraint violations was reduced to zero or a user specified number of attempts had been made. A list of all classes that were involved in these violations was made. One class was chosen at random from this list and a second list made of all the other room time combinations to which it could be moved. One possible move was chosen from this second list and the change this possible move would make in the constraint violation count was calculated. Only if this was positive was the move actually made. A new list of clashing classes was then made and the process repeated until either no clash occurred or a total of fifty tries had been made. The algorithm described in the above paragraph is very greedy and can easily result in a local optimum being reached that does not meet all constraints. As a result the original quantized position passed to the local heuristic was saved and the algorithm described above was tried twenty times, with the original quantized position being restored at the start of each time. The best result found in any of these twenty tries was the result actually used. The extra computational load was insignificant as the local heuristic was only run when a wave died – typically a few hundred times per run. The results of running the conventional and WoSP variant of the PSO on this problem are shown in Table 4.

Table 4 shows that in terms of the best performing repeat out of each group of 100 there is a steady improvement as progressive enhancements are made to the classical particle swarm algorithm (from left to right in the table). Interestingly, adding waves alone produces a greater positive effect than adding

Table 4. An overview of the performance of 100 independent repeat runs for each of the four possible combinations of waves and local heuristic

Using waves Local heuristic	No No	No Yes	Yes No	Yes Yes
Number of times one or more solutions found that satisfied all constraints	0	0	8	80
Average number of constraint violations per run	5.72	3.81	2.25	0.2
Number of constraint violations in best solution found	1	1	0	0
Number of constraint violations in worst solution found	8	8	3	1

Table 5. The number of constraint violations for 100 repeats of each of all combinations of waves and local heuristic

Number of constraint violations in best solutions found	Waves Local heuristic	No No	No Yes	Yes No	Yes Yes
0				8	80
1			6	26	20
2		1	10	32	
3		1	32	17	
4		15	22	10	
5		24	13	5	
6		29	14		
7		27	2	2	
8		3	1		

Table 6. The average reduction in constraint violations obtained using the local heuristic for 100 repeats with and without waves

	Average	Minimum	Maximum
Without waves	1.9	0	5
With waves	2.1	0	7

the local heuristic alone but the best results are obtained when both of these are used. While Table 4 shows only the best result for each repeat, Tables 5 and 6 show statistics derived from all the repeats for all combinations.

Comparing the columns in Table 5 that do not involve the local heuristic, it is clear that the addition of waves consistently and substantially improves the performance. Again it can be observed that the performance with waves alone is better than the performance with the local heuristic alone.

Table 6 shows that the improvement made by the use of the local heuristic was essentially independent of the use of waves.

The best performing combination by far is when both waves and the local heuristic are used and these results have been examined in more detail.

During the 100 independent repeats, each run reported on average 653 (max 671, min 634) candidate solutions that were passed to the local heuristic. A candidate solution corresponds to the best position found by a wave during its existence and subsequently refined using the local heuristic. The range of the number of constraint violations in all these candidate solutions is from 0 to 12. During the runs a grand total of 1242 solutions that satisfied all constraints (absolute solutions) were found. Twenty runs produced no absolute solutions; the other 80 runs produced between 1 and 51 solutions each, with an average of 15.7 absolute solutions per run. However, even though each particle maintained an individual list of promotion points, some optima were explored by more than one wave during a run. On average, the runs that found absolute solutions found 5.8 different absolute solutions each (the maximum for any wave being 21, the minimum 1). Overall, the 100 repeats found a grand total of 446 different absolute solutions out of the approximately 2500 that exist.

10 Concluding Remarks

The particle swarm optimization algorithm has proved to be efficient, fast and flexible. It shows promise as an effective algorithm for a wide range of optimization problems. A number of variations have been suggested to better suit it to particular classes of problems and some of these, together with the basic algorithm, have been discussed in this Chapter. No algorithm is ideal for all situations and it has been noted the PSO, like other algorithms such as the genetic algorithm, is really a coarse search algorithm that becomes inefficient in the final stages of homing in on an optimum. But, when coupled with an appropriate local search technique, PSO and its many variants deserve a prominent place in the armory of everyone seriously involves with optimization in the real world.

References

1. Blackwell T, Branke J (2004) Multi-swarms optimization in dynamic environments. In: *Lecture Notes in Computer Science,* 3005. Springer-Verlag, Berlin: 489–500.
2. Blackwell TM, Bentley PJ (2002) Dynamic search with charged swarms. In: Langdon WB et al. (eds.) *Proc. Genetic and Evolutionary Computation Conf. – GECCO-2002* , 9–13 July, New York, NY. Morgan Kaufmann, San Francisco, CA: 19–26.
3. Braendler D, Hendtlass T (2002) The suitability of particle swarm optimisation for training neural hardware. In: Hendtlass T, Ali M (eds.) *Lecture Notes in Artificial Intelligence,* 2358, Springer-Verlag, Berlin: 190–199.
4. Brits R (2002) Niching strategies for particle swarm optimization. *Master's Thesis.* Department of Computer Science, University of Pretoria, South Africa.
5. Brits R, Englebrecht A, van der Bergh F (2002) A niching particle swarm optimiser. In: *Proc. 4th Asia-Pacific Conf. Simulated Evolution and Learning (SEAL'2002),* 18–22 November, Singapore: 692–696.
6. Carlisle A, Dozier G (2000) Adapting particle swarm optimization to dynamic environments. In: Arabnia HR (ed.) *Proc. Intl. Conf. Artificial Intelligence,* 26–29 June, Las Vegas, NV: 429–433.
7. Carlisle A, Dozier G (2002) Tracking changing extrema with adaptive particle swarm optimizer. In: Jamshidi M, Hata Y, Fathi M, Homalfar A, Jamshidi JS (eds.) *Proc. World Automation Congress (Intl. Symp. Soft Computing in Industry – ISSCI'2002)* 9–13 June, Orlando FL, TSI Press, Albuquerque, NM: 265–270.
8. Clerc M (1998) Some math about particle swarm optimization. (available online at http://clerc.maurice.free.fr/PSO/PSOmathstuff/PSOmathstuff.htm – last accessed January 2007).

9. Clerc M (1999) The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *Proc. Congress Evolutionary Computation (CEC1999),* 6–9 July, Washington, DC. IEEE Press, Piscataway, NJ, 3: 1957.
10. Digalakis J, Margaritis K (2000) An experimental study of benchmarking functions for genetic algorithms. *Intl. J. Computer Mathematics,* 79: 403–416.
11. Eberhart R, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimisation. *Proc. 2000 Congress Evolutionary Computation,* 16–19 July, La Jolla, CA. IEEE Press, Piscataway, NJ: 84–88.
12. Hendtlass T, (2007) Fitness Estimation and the Particle Swarm Optimisation Algorithm, *Proc Congress on Evolutionary Computing (CEC2007).* IEEE Computer Society Press, Piscataway, NJ: 4266–4272.
13. Hendtlass T (2006) A particle swarm algorithm for complex quantised problem spaces. *Proc. Congress Evolutionary Computation (CEC2006),* 16–21 July, Vancouver, Canada. IEEE Computer Society Press, New York, NY: 3760–3765.
14. Hendtlass T (2005) WoSP: a multi-optima particle swarm algorithm. *Proc. Congress Evolutionary Computing (CEC2005)* 2–5 September, Edinburgh, UK. IEEE Press, Piscataway, NJ: 727–734.
15. Hendtlass T (2004) A particle swarm algorithm for high dimensional problem spaces. *Proc. IEEE Swarm Workshop,* 9–11 May, Ann Arbor, MI (available online at http://www.cscs.umich.edu/swarmfest04/Program/Abstracts/abstracts.html #HendtlassT – last accessed 22 May 2007).
16. Hu X, Eberhart R (2002) Adaptive particle swarm optimisation: detection and response to dynamic systems. In: *Proc. Congress Evolutionary Computation (CEC2002),* 12–17 May, Honolulu, Hawaii. IEEE Press, Piscataway, NJ: 1666–1670.
17. Janson S, Middendorf M (2003) A hierarchical particle swarm optimizer. *Proc. Congress Evolutionary Computing (CEC2003),* 9–12 December, Canberra, Australia. IEEE Press, Piscataway, NJ: 1666–1670.
18. Janson S, Middendorf M (2004) A hierarchical particle swarm optimizer for dynamic optimization problems. In: Raidl GR, Cagnoni S, Branke J, Corne DW, Drechsler R, Jin Y, Johnson CG, Machado P, Marchiori E, Rothlauf F, Smith GD, Squillero G (eds.) *Proc. EvoWorkshop 2004,* 20–24 June, Toulouse, France. Lecture Notes in Computer Science 3005, Springer-Verlag, Berlin: 513–524.
19. Kennedy J, Eberhart RC (1995) Particle swarm optimization. *Proc. IEEE Conf. Neural Networks (ICNN95)*, November, Perth, West Australia. IEEE Press, Piscataway, NJ: 1942–1947.
20. Paquet U, Engelbrecht AP (2006) Particle swarms for equality-constrained optimization. *Fundamenta Informaticae,* 76: 1–24.
21. Parrot D, Li X (2004) A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In: *Proc. 2004 Congress Evolutionary Computation (CEC2004),* 20–23 June, Portland, OR. IEEE Press, Piscataway, NJ: 98–103.
22. Randall M (2005) A dynamic optimisation approach for ant colony optimisation using the multidimensional knapsack problem: recent advances in artificial life. *Advances in Natural Computation,* 3: 215–226.
23. Salami M, Hendtlass T (2002) A fast evaluation strategy for evolutionary algorithms. *J. Soft Computing,* 2(3): 156–173.
24. Schwefel HP (1981) *Numerical Optimization of Computer Models.* Wiley, Chichester, UK.

Resources

1 Key Books

Engelbrecht AP (2005) *Fundamentals of Computational Swarm Intelligence.* Wiley, London, UK.

Kennedy J, Eberhart RC, Shi Y (2001) *Swarm Intelligence.* Morgan Kaufmann, San Francisco, CA.

Bonabeau M, Dorigo M, Theraulaz G (1999) *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, UK.

2 Organisations, Societies, Special Interest Groups, Journals

IEEE Computational Intelligence Society
http://www.ieee-cis.org

3 Key International Conferences/Workshops

IEEE Congress on Evolutionary Computing – CEC (IEEE)

Genetic and Evolutionary Computation Conference – GECCO
(ACM SIGEVO)

Swarmfest
http://www.swarm.org

4 (Open Source) Software

`CIlib` – a public domain framework and library for CI algorithms
http://cilib.sourceforge.net

Optimization Algorithm Toolkit (`OAT`) 'A workbench and toolkit for developing, evaluating, and playing with classical and state-of-the-art optimization algorithms on standard benchmark problem domains; including reference algorithm implementations, graphing, visualizations and much more.'
http://optalgtoolkit.sourceforge.net/