Adaptively Choosing Niching Parameters in a PSO

Stefan Bird
School of Computer Science and IT
RMIT University
376 Swanston St
Melbourne, Australia
stbird@seatiger.org

Xiaodong Li
School of Computer Science and IT
RMIT University
376 Swanston St
Melbourne, Australia
xiaodong@cs.rmit.edu.au

ABSTRACT

Niching techniques play an important role in evolutionary algorithms. Existing niching methods often require user-specified parameters, limiting their usefulness. This paper proposes a niching method for Particle Swarm Optimisation (PSO) where population statistics are used to adaptively determine the niching parameters during a run. The proposed niching method is compared to another niching based PSO, SPSO. Our results show that the proposed adaptive niching PSO can solve difficult multimodal functions more reliably and with fewer evaluations.

Categories and Subject Descriptors

G.1 [**Numerical Analysis**]: Optimisation; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Evolutionary Computation, Particle Swarms, Multimodal Function Optimisation

1. INTRODUCTION

Niching algorithms are designed to increase the number of solutions an evolutionary algorithm (EA) finds on a multimodal problem. In many problems there exist multiple global optima and it is often desirable to locate either all or a substantial portion of these solutions. Even in the presence of multiple optimal solutions, a non-niching EA will generally converge to a single solution. Several diversity-preserving algorithms have been developed to prevent this behaviour, namely derating [1], restricted tournament selection [6], crowding [8, 14], fitness sharing [5] and speciation [2, 7, 12, 16, 17].

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

Without niching, an EA will generally concentrate on the best peak found so far, forgetting about any other promising solutions it may have previously found. Using niching allows the algorithm to search multiple peaks simultaneously. Even on problems with only one global optimum, this has advantages as it reduces the risk of the algorithm getting stuck in a local optimum – some of the population will exploit the optimum, but others will be exploring other areas of the search space.

Existing niching algorithms often require a parameter to specify how large each niche should be. While the representation of the parameter varies, it is essentially a measure of distance between either two individuals or an individual and a point in the problem space. Niching EAs are sensitive to this parameter [7, 12, 13], but the optimal value differs for each problem and can be difficult to determine.

This paper presents a new Adaptive Niching PSO (ANPSO) that adaptively determines this parameter; the user is not required to predetermine any niching parameters. The paper is organised as follows: Section 2 introduces the basics of a PSO and how subpopulations can be formed using niching. Section 3 describes how this niching PSO works. The performance measurement will be described in Section 4 followed by the actual results in Section 5. Conclusions will be given in Section 6.

2. PRIOR WORK

This work combines niching algorithms with particle swarms. We will give an introduction to particle swarms, followed by a review of the relevant existing niching algorithms.

2.1 Particle Swarms

Particle Swarm Optimisation (PSO) is a search strategy involving many autonomous "particles" [10]. These particles move around in the problem domain, searching for good solutions. Each particle has a neighbourhood of particles that it is connected to. At each step t, each particle i is attracted to a random point near the particle's personal best point $\vec{p}_{(i,t)}$ and the best point found by any of its neighbours $\vec{p}_{(g,t)}$. For our testing we have used the constriction coefficient [3, 10] variation of PSO.

$$\vec{v}_{(i,t+1)} = \chi(\vec{v}_{(i,t)} + \varphi_1(\vec{p}_{(i,t)} - \vec{x}_{(i,t)}) + \varphi_2(\vec{p}_{(g,t)} - \vec{x}_{(i,t)})) \quad (1)$$

$$\vec{x}_{(i,t+1)} = \vec{x}_{(i,t)} + \vec{v}_{(i,t+1)} \quad (2)$$

where:

$$\varphi_1 = c_1 r_1, \quad \varphi_2 = c_2 r_2,$$

$$\chi = \frac{2\kappa}{\left|2 - c - \sqrt{c^2 - 4c}\right|}$$

c_1 and c_2 are constants, typically 2.05. $c = c_1 + c_2$. κ is also a constant, usually set at 1. r_1 and r_2 are uniform random numbers between 0 and 1.

To determine the next place to search $\vec{x}_{(i,t+1)}$, the particle calculates its new velocity $\vec{v}_{(i,t+1)}$ using equation (1) and adds it to its current location $\vec{x}_{(i,t)}$ (equation (2)). As the particles have momentum, they have a tendency to "overshoot" the point they are aiming for. This causes them to turn back towards the original point and overshoot it again. By continually overshooting the best locations, the particles thoroughly explore the surrounding search space while largely ignoring unpromising areas [10]. The velocity component in each dimension is limited to the search space size in that dimension. Particles "bounce" off the search space boundaries to prevent them from leaving. The constriction factor χ provides a dampening effect on the velocity and is used to prevent particles from oscillating wildly around an optimum [3].

2.2 Niching

Several niching techniques already exist. Pétrowski [17] presented a clearing method for genetic algorithms where the individuals are sorted by fitness. The algorithm then sets the fitness of any individual within σ of a fitter individual to 0. This means that each peak is represented by a single "winner".

Li extended Pétrowski's idea to create SPSO [12] and SDE [13]. These algorithms differ from Pétrowski's by creating explicit niches. In SPSO, particles in different niches do not communicate, exploring different areas of the problem space in isolation. Both SPSO and SDE also use the concept of each niche acting as a subpopulation - only individuals within the same niche may interact or breed.

Brits *et al* developed NichePSO [2] which differs from a normal PSO in that the unniched particles do not interact with each other, performing individual searches instead. If the fitness variance of a particle over the last 3 steps was less than a predefined amount δ, a subswarm is created consisting of that particle and its nearest topological neighbour. Other particles can subsequently join the created niche if they move to a point within the niche's area.

Kennedy described a clustering technique [9] which assigns each particle to a cluster and substitutes the cluster centre for the particle's personal best. Although Kennedy's technique does not use a radius parameter, it is necessary to set the number of clusters. The number of optima that can be located is limited by the value of this parameter; usually only one optimum will be located per cluster. Kennedy does not report how the number of clusters affects performance.

One of the difficulties with existing niching algorithms is specifying the radius parameter. Although different algorithms call it different things, it essentially controls how far away two individuals can be while still being considered as in the same niche. Setting this parameter to an appropriate value requires knowledge of the problem domain. Setting it too small reduces exploitation of an optimum and may result in many population members becoming trapped in local optima. If it is set too large, niches may overlap multiple optima. This makes it difficult or impossible for the algorithm to find all of the solutions [13].

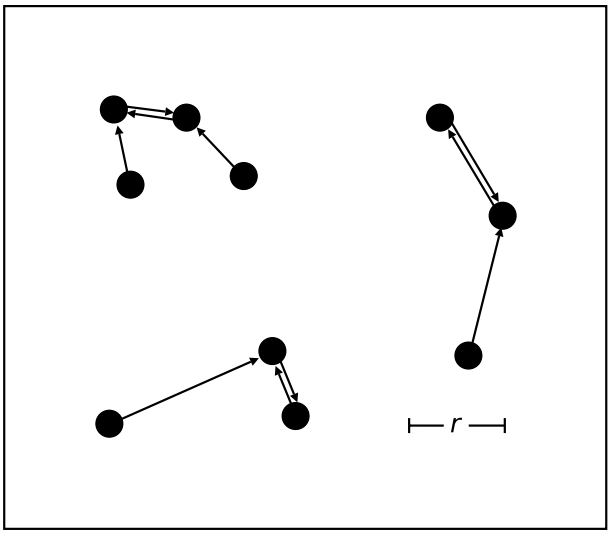

Figure 1: Calculating the distance from each particle to the particle closest to it. r is calculated by averaging these distances.

3. ADAPTIVELY DETERMINING NICHING PARAMETERS

Instead of requiring the user to specify the niche radius r, we adaptively determine it from the population statistics at each iteration. This method uses the intrinsic nature of the particles to converge on optima and creates niches when they do so. When a particle discovers a local peak, its velocity will reduce and it will explore the area closely. If multiple particles explore the same area it is likely that it is a peak of interest, so a niche is created with those particles as its initial members. Any other particles that later converge on the peak will also join the niche.

The set of particles is represented by P, which has n members. In **step 1**, for each member p_i of the population, we measure the distance d_i to the closest other particle in the decision space using equation (3). We set r to the average of these distances, as shown in equation (4) and Figure 1. $||p_i - p_j||$ measures the euclidean distance between p_i and p_j.

$$d_i = \min\left\{||p_i - p_j|| \,; \forall j \, p_i, p_j \in P \wedge p_i \neq p_j\right\} \qquad (3)$$

$$r = \frac{\sum_{i=1}^{n} d_i}{n} \qquad (4)$$

In **step 2**, an undirected graph g is created containing a node for each particle. Initially there are no edges between any of the nodes. **Step 3** finds every set of particles that have been close to each other for at least 2 steps. The algorithm finds the pairs of particles that are within r of each other, as shown in Figure 2. A counter is maintained for each pair - if the pair have been close for two[1] or more

[1]Two steps has been chosen as it provides the best (or close

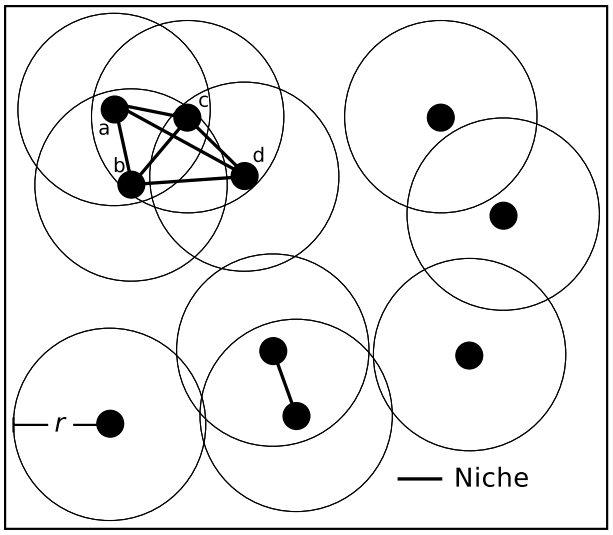

Figure 2: If a particle remains in another particle's circle (they are closer than r) for 2 consecutive steps, the particles form a niche. Notice that *d* is in the same niche as *a* and *b*, even though the only particle it is close to is *c*.

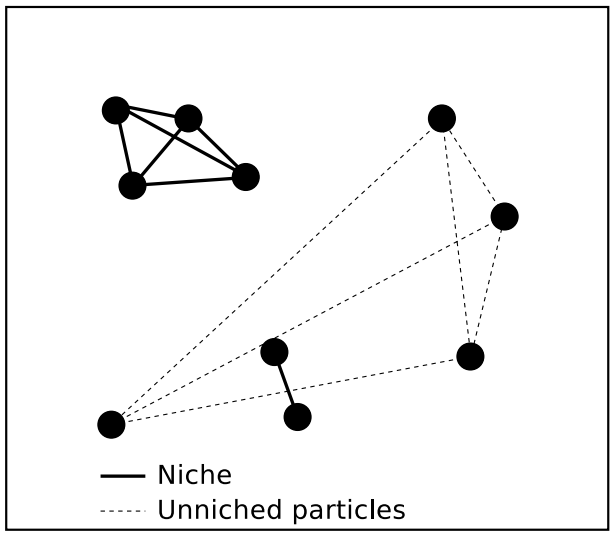

Figure 3: Each niche acts as an independent subpopulation. The unniched particles continue to look for other optima. This improves on Pétrowski's method as it does not require the user to specify the r parameter.

steps, a niche is formed. A formal description is given in Algorithms 1 and 2, which are invoked at every iteration.

While the algorithm is more computationally expensive than most other niching techniques ($O(n^2)$ complexity), the speed penalty is minimal for swarms with fewer than 1000 particles. This cost is more than offset by the time taken to tune the niche size parameter in other niching algorithms, especially in many real-world problems where the fitness evaluations take a non-trivial amount of time.

Particles must leave a niche if they find a better location elsewhere, otherwise it may destabilise the remaining niche members. Particles will leave a niche if they are further than r from the nearest particle within their niche for 2 consecutive steps. As a niche is defined as a set of particles that are close together, it is impossible to have a niche containing only one particle. If all of a niche's particles leave, the niche will disappear.

To implement this, the algorithm tracks how many steps each particle has been within r of every other particle in an array called s. At each step, if two particles are closer than r, the counter is incremented for that pair. Every step that the two particles are apart decrements the counter. A particle is considered to be in the same niche as another particle if the counter is ≥ 2. By requiring particles to be away for more than one step before allowing them to leave a niche, the niches are more stable.

The counter for each pair stays within the range of $[0, 4]$ in order to allow particles to join or leave niches at a reasonably fast pace. If two particles converge at step 50, normally they would form a niche at step 52. Presumably they had been apart for the first 50 steps. If the counter was allowed to go below 0, when they initially converged their counter would have been at -50. In this case they would have to wait an additional 50 steps before forming the niche, just to return the counter to 0. The upper limit is 4 so as to allow a particle to leave a niche if it has been distant for more than 2 consecutive steps.

As the number of particles in each niche is generally very low (between 2 and 5), the particles within each niche use a fully connected topology[2]. With very small populations most neighbourhood topologies act similarly to this; with even a moderately connected topology each particle is connected to almost every other. When using a fully connected neighbourhood only one neighbourhood best needs to be computed per niche; most other topologies require one computation per particle.

The unniched particles are placed in a von Neumann neighbourhood which is updated each step. This provides good results on a wide range of problems and encourages simultaneous convergence on multiple optima [11]. Since all particles are still indirectly connected to each other, knowledge of "better" optima (even if only infinitesimally better) will eventually propagate throughout the whole swarm, making it very difficult for the system to converge on new optima. Optima may also be lost when the particles converging on them are attracted to other more precisely located optima. Eventually the swarm will stabilise on only a few optima, even on highly multimodal problems.

As particles form niches, they are removed from the unniched network as in Figure 3. This serves two purposes: it prevents a particle from being attracted away from its local optimum by other particles, and causes the rest of the population to "forget" about the newly found optimum, allowing it to explore other areas of the decision space.

To prevent too many particles converging on the same optimum (a problem when a function has many optima but the population size is small, for example Himmelblau with a population of 10), a limit is placed on the number of particles

to) performance on every problem tested. See the results section and Figure 4.

[2]When applied across the entire population, a fully connected topology is called the global neighbourhood model.

s is a n x n array that is created at the beginning of the run to track how many steps each particle has been near every other particle. Initially every element is 0.

1. Determine r using equation (4);
2. Create an undirected graph g containing a node for each particle, but no edges;

3. Calculate values for s and add edges to g:

```
for i=1 to n-1 do
    for j=i+1 to n do
        if ||p_i − p_j|| < r then
            Increment s_{i,j};
            if s_{i,j} > 4 then
                s_{i,j} ← 4;
            end
            if s_{i,j} >= 2 then
                Create an edge in g from p_i to p_j;
            end
        end
        else
            Decrement s_{i,j};
            if s_{i,j} < 0 then
                s_{i,j} ← 0;
            end
        end
    end
end
```

Algorithm 1: The procedure for creating the niche graph. In the algorithm, the number 4 is chosen so as to allow a particle to leave the niche if it is further than r for 2 steps. s is used to keep track of how long each particle has been close to every other particle.

allowed in a single niche as was done in [16]. If a niche contains too many particles, the ones with the worst fitness will be reinitialised to a random position and velocity, and their personal best memory deleted.

4. PERFORMANCE MEASUREMENT

As the goal of a niching algorithm is to find and maintain niches on all of the global optima, we have measured the performance of this algorithm as the number of evaluations required to form a stable niche on every optimum. This is measured by counting the number of niches where the best individual has a personal best fitness within the acceptance threshold $\epsilon = 0.00001$ of the known global best fitness. To be considered stable, a niche must remain on an optimum for 10 consecutive steps after having reached the required accuracy. Without this requirement, a species may temporarily form on a global optimum but disperse because the velocities of the particles are too high to converge there. As it is just as important for the algorithm to be reliable, we performed 50 runs for each parameter combination. The *success rate* was measured as the number of runs where every peak was located. A run is considered unsuccessful if the swarm failed to find every optimum within 2000 steps.

After creating the graph in Algorithm 1:

Create a set variable $visited \leftarrow \oslash$;
Create a set variable $reachable$;

```
for i=1 to n do
    if p_i ∉ visited and d_i < r then
        reachable ←
        {∀j p_i, p_j ∈ P ∧ p_j reachable from p_i in g};
        Create a new niche s ← {p_i};
        for p ∈ reachable and p ∉ visited do
            visited ← visited ∪ {p};
            s ← s ∪ {p};
        end
    end
end
```

Algorithm 2: Creating the niches from the niche graph.

Five different multimodal test functions have been chosen to allow comparison with SPSO: Brainin RCOS, Six-Hump Camel Back, Deb's 1st Function, Himmelblau and Shubert 2D. These functions are described in Table 1.

The r parameter for SPSO has been set to the optimal value for each fitness function - the largest value that did not cause nearby peaks to interfere with each other. On most of the functions this was around half the distance between the two closest optima.

There are two parameters that can be reasonably applied to ANPSO:

- The number of steps two particles must be close before forming a niche.
- The maximum number of particles in each niche.

To test how robust these parameters are, we ran the experiments with a range of values.

For the steps before niche parameter, we tested values from 2 to 5 for each population size. Preliminary experiments showed ANPSO performed extremely poorly if this parameter was set at 1. Without a delay before forming a niche, niches were formed and broken between different particles on each step as the particles travelled past each other. As the particles changed niche at each step, each particle's neighbourhood best location changed dramatically, preventing its velocity from reducing sufficiently to allow convergence.

The maximum niche size parameter was tested with values between 2 and 20. It should be noted that as many niches consist of only a few particles (most niches have less than 10), values larger than this have a minimal effect on the algorithm.

The algorithm was also tested on the 3D and 4D Shubert functions. The Shubert function can be generalised to any number of dimensions using the following form [7]:

$$f(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} \sum_{j=1}^{5} j \cos[(j+1)x_i + j]$$

Shubert 3D has 81 optima in 27 groups and it is thought that Shubert 4D has 324 optima in 81 groups [7]. The tests were run to see how quickly ANPSO and SPSO locate the optima. For SPSO, several values of r were tested: 0.65,

Table 1: Test functions

Function	Function Name	Range	Comments
F1	Brainin RCOS [15]: $F1(x,y) = (y - \frac{5.1x^2}{4\pi^2} + \frac{5x}{\pi} - 6)^2 + 10(1 - \frac{1}{8\pi})cos(x) + 10$	$-5 \leq x \leq 10$; $0 \leq y \leq 15$	3 global optima
F2	Six-Hump Camel Back [15]: $F2(x,y) = -4[(4 - 2.1x^2 + \frac{x^4}{3})x^2 + xy + (-4 + 4y^2)y^2]$	$-1.9 \leq x \leq 1.9$; $-1.1 \leq y \leq 1.1$	2 global optima and 4 local optima
F3	Deb's 1st function [4]: $F3(x) = sin^6(5\pi x)$	$0 \leq x \leq 1$	5 equally spaced global optima
F4	Himmelblau [1]: $F4(x,y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2$	$-6 \leq x, y \leq 6$	4 global optima
F5	Shubert 2D[7]: $F5(x,y) = \sum_{i=1}^{5} i\cos[(i+1)x + i] \sum_{i=1}^{5} i\cos[(i+1)y + i]$	$-10 \leq x, y \leq 10$	18 global optima in 9 clusters; many local optima

Table 2: Number of evaluations required to find all global peaks (mean and standard deviation).

Function	Pop.	ANPSO	SPSO
F1	30	5220 ±3323	3169 ±692
	60	6927 ±2043	6226 ±1707
F2	30	2798 ±857	2872 ±827
	60	4569 ±1316	5820 ±1469
F3	30	6124 ±2465	2007 ±703
	60	8665 ±2974	4848 ±2092
F4	30	16308 ±13157	4069 ±731
	60	17168 ±12006	7590 ±2018
F5	300	82248 ±10605	166050 ±42214
	500	114580 ±18392	219420 ±80179

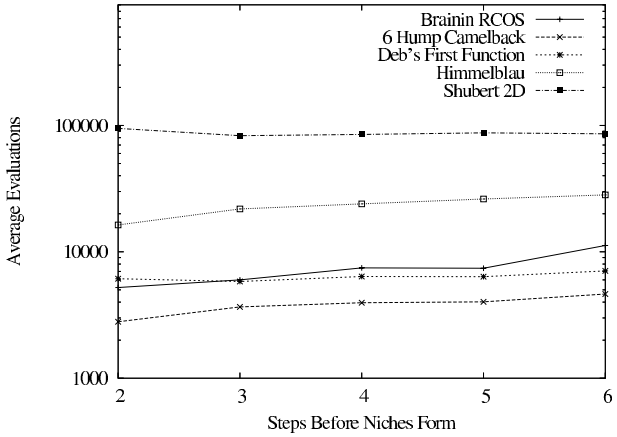

Figure 4: Average number of evaluations required to find all optima, waiting the specified number of steps before a niche forms. All functions except Shubert 2D used a population of 30. Shubert 2D used a population of 300.

1, 3 and 6. Due to the large number of optima, a much larger population size was used: 700 for Shubert 3D and 2100 for Shubert 4D. To increase the search power on these more difficult problems, no species size limit was used. An optimum was considered to be found if a niche represented it within a fitness accuracy $\epsilon = 0.1$.

5. RESULTS

Four sets of experiments were carried out. We studied the niche formation delay, the effect of using a maximum niche size and the sensitivity to the population size using five 1 or 2 dimensional multimodal functions. We then also tested our algorithm on the Shubert 3D and 4D problems to see how well it scales to higher dimensions.

The results presented have been averaged over 50 runs. Except when using small population sizes (< 30), both ANPSO and SPSO were successful on every run of F1, F2, F3 and F4. SPSO did not perform as well on Shubert 2D (F5); the reasons for this will be discussed further in the paper.

Table 2 shows that on the problems of Brainin RCOS, Deb's First Function and Himmelblau, SPSO required approximately $\frac{1}{2}$ to $\frac{1}{4}$ as many evaluations to locate all optima. ANPSO required a similar number of evaluations to solve Six-Hump Camelback as the SPSO algorithm. It should be noted that SPSO required many "tuning runs" to achieve these results. If these were taken into account, the number of evaluations required by SPSO would be much higher.

5.1 Niche formation delay

As can be seen in Figure 4, the number of steps that two particles must be close does not have a large effect. There is a slight trend upwards as this parameter is increased, but on every function tested a value of 2 gave the best performance except for Shubert 2D where the best value was 3. While Shubert 2D performed the worst with the parameter set at 2, it was only 14% slower and was still successful on every run with a population > 100. This shows the parameter is very robust. On all of the values tested, ANPSO successfully located all global optima on every run. It is recommended this parameter be set to 2 for all problems.

5.2 Maximum niche size

Figure 5 illustrates the effect of setting a maximum niche size. It can be seen that the algorithm performs relatively poorly when this parameter is set to 2; this is due to the difficulty PSOs have locating an optimum when less than 3 particles are interacting [2]. Other than this, the curve in the figure is relatively flat, showing that on problems where there were an adequate number of particles to locate each niche (to achieve 100% success rates, the algorithm generally needs 5 particles for each peak), the parameter had very little effect on the algorithm's performance. No point has been plotted for Shubert 2D with a maximum niche size of 2 as there were no successful runs with this setting.

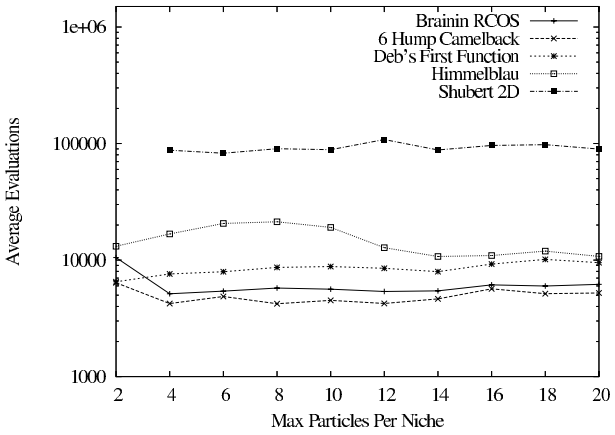

Figure 5: Average number of evaluations required to find all optima using an upper limit on how many particles can belong to a particular niche. Population sizes are the same as in Figure 4.

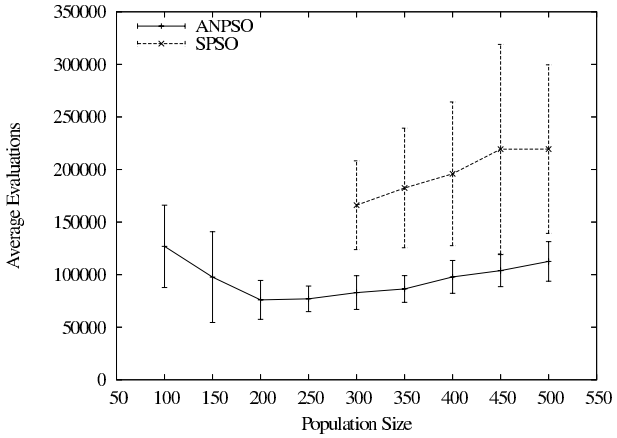

Figure 6: The effect of population size on the number of evaluations required to find all 18 optima on Shubert 2D.

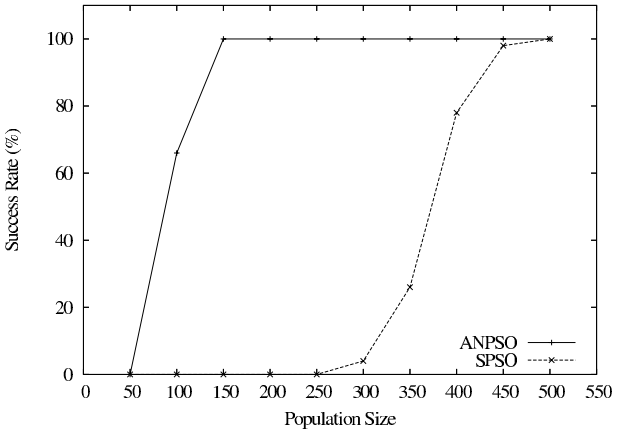

Figure 7: The percentage of successful runs with different population sizes solving Shubert 2D. A run is successful only if it finds all 18 optima.

5.3 Sensitivity to population size

In terms of the number of evaluations required, ANPSO's sensitivity to population size is comparable to a standard PSO. The optimal population size for Shubert 2D is around 200 (Figure 6); below this the number of evaluations increases rapidly. Population sizes larger than 200 do not require considerably more evaluations, although performance does degrade when more than 350 particles are used.

The differences between SPSO and ANPSO are most pronounced on the Shubert 2D function. ANPSO required half as many evaluations as SPSO and had a much higher success rate (Figure 7). On every population size above 100, ANPSO had a 100% success rate. In comparison, even at 300 particles SPSO was only able to find all of the global optima in 2 of the 50 runs. As the optima are very close, SPSO requires a very small r value to separate them, however this causes many particles to form niches containing only themselves. With no outside neighbourhood best to guide them, most particles end up stopped on a local peak. This wastes many evaluations as the particles continually sample the same point. The stopped particles are unable to locate new optima and do not help each other in the search, greatly reducing efficiency.

It is likely the primary advantage ANPSO has on this function is that it does not create niches containing a single particle. Instead the unniched particles act like a normal PSO, greatly increasing the search power on this highly multimodal function.

5.4 Higher dimensional problems

ANPSO performs well on higher dimensional problems. While it was unable to find all global optima on the Shubert 3D and 4D functions, it found a significant portion and performed far better than SPSO. Figure 8 shows the number of optima found over time for both test functions.

Most of the global optima are located within a short space of time; it is only necessary to run the algorithm for an extended time if a large number of optima are required. When properly tuned, SPSO is initially able to discover new optima at a faster rate than ANPSO, however it rapidly plateaus, unable to find more peaks. ANPSO locates the optima more gradually, catching up to SPSO at around tick 400. Most of the optima it will discover are known by tick 1200, although it will continue to find more until around tick 1600.

SPSO performs very poorly on Shubert 3D and 4D - when r was set to either 0.65 or 1 it had great difficulty locating even one optimum; all of the particles were becoming trapped in local optima because there was insufficient communication. The best results on both test functions are achieved by setting r to 3. This is a very large radius however; with this value the peaks within each cluster are too close for SPSO to differentiate them; the best it could do was locate the cluster. On Shubert 3D, SPSO was able to locate most of the clusters, finding 24 of the 27 on average. In comparison, ANPSO located 41 global peaks, showing that it was able to differentiate the peaks within each cluster.

On Shubert 4D, neither SPSO nor ANPSO are able to locate all of the clusters, although ANPSO was able to locate approximately 50% more. While we do not know why ANPSO is unable to find all of the optima on these functions, we suspect it may be due to the r value becoming extremely small once many niches have converged. As r

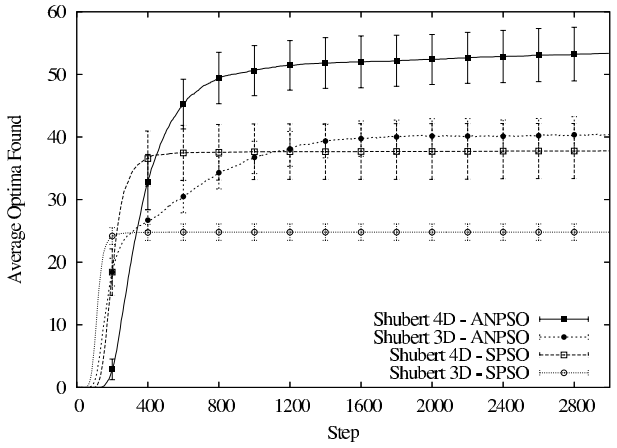

Figure 8: The average number of global optima found over time solving Shubert 3D and 4D. For SPSO, $r = 3$.

gets smaller it becomes increasingly difficult for new niches to form; particles must wait until they are tightly converged before they can separate from the main population. This increases the chance of a neighbour locating a better point, causing the particle to deconverge and start exploring again. Future research could look at placing a lower limit on r to prevent this.

Li *et al* [7] developed a GA niching method called SCGA that is able to locate every cluster in Shubert 3D and 4D. This algorithm requires many parameters to be set however, and they also used a much larger population size. In their testing, Li *et al* set their parameters so as to only locate each cluster - not its constituent peaks. In Shubert 3D, ANPSO was able to differentiate the peaks within each cluster. More research is required to determine whether this was also the case for Shubert 4D.

6. CONCLUSION

The primary advantage of ANPSO over previous niching techniques is that it does not require a parameter specifying how large each niche is. While on simple problems with known properties it requires more evaluations to locate every optimum, it performed very well on the most complex and multimodal function tested, Shubert. The algorithm is insensitive to the value of the parameters, allowing it to be used on a variety of problems without tuning. This ability makes it very suitable for real-world tasks where it is impractical or impossible to perform multiple "tuning runs" beforehand.

Due to its adaptive nature, this niching method has promising applications for optimisation in dynamic environments. Further research would be required to determine exactly what kind of problems it is most suitable for and how performance could be improved on the simpler functions tested.

7. REFERENCES

[1] D. Beasley, D. Bull, and R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[2] R. Brits, A. Engelbrecht, and F. van den 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, 2002.

[3] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. In *IEEE Transactions on Evolutionary Computation*, volume 2, pages 58–73, 2002.

[4] K. Deb and D. Goldberg. An investigation of niche and species formation in genetic function optimization. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, 1989.

[5] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[6] G. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 24–31. Morgan Kaufmann, 1995.

[7] J.Li, M. Balazs, G. Parks, and P. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–234, 2002.

[8] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[9] J. Kennedy. Stereotyping: improving particle swarm performance with cluster analysis. In *Congress on Evolutionary Computation (CEC2000)*, volume 2, pages 1507–1512, 2000.

[10] J. Kennedy and R. Eberhart. *Swarm intelligence.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[11] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC)*, pages 1671–1676, 2002.

[12] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of Genetic and Evolutionary Computation Conference 2004 (GECCO'04) (LNCS 3102)*, pages 105–116, 2004.

[13] X. Li. Efficient differential evolution using speciation for multimodal function optimization. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 873–880, 2005.

[14] S. W. Mahfoud. Crowding and preselection revisited. In *Parallel problem solving from nature 2*, pages 27–36, Amsterdam, 1992. North-Holland.

[15] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, New York, 1996.

[16] D. Parrott and X. Li. A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In *Congress on Evolutionary Computation (CEC2004)*, volume 1, pages 98–103, 2004.

[17] A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation*, pages 798–803, 1996.