Dynamic Niche Clustering: A Fuzzy Variable Radius Niching Technique for Multimodal Optimisation in GAs

Justin Gan & Kevin Warwick
Cybernetic Intelligence Research Group
Department of Cybernetics,
University of Reading,
Whiteknights, Reading,
Berkshire, RG6 6AY, UK
Email: jgrg@cyber.reading.ac.uk

Abstract- This paper describes the recent developments and improvements made to the variable radius niching technique called Dynamic Niche Clustering (DNC) originally described in [4] and [5]. DNC is fitness sharing based technique that employs a separate population of overlapping fuzzy niches with independent radii which operate in the decoded parameter space, and are maintained alongside the normal GA population. We describe a speedup process that can be applied to the initial generation which greatly reduces the complexity of the initial stages. A *split* operator is also introduced that is designed to counteract the excessive growth of niches, and it is shown that this improves the overall robustness of the technique. Finally, the effect of local elitism is documented and compared to the performance of the basic DNC technique on a selection of 2D test functions. The paper is concluded with a view to future work to be undertaken on the technique.

1 Introduction

Multimodal fitness landscapes in large multidimensional search spaces are a fact of modern everyday problem domains. Thus, when using genetic algorithms to optimise such problems, some method of preserving genetic diversity or encouraging speciation is necessary to avoid getting stuck at a local optima – this can either be through fitness modification, or a mechanism whereby individuals are selected for reproduction from each niche[1]. One would also not want to ignore other peaks of comparable fitness that could provide valid alternative solutions to the problem. Ideally, the overhead of using such a mechanism should not unduly delay the processing time.

There has been much research within this particular branch of genetic optimisation, of particular note are the fixed radius niching algorithms; fitness sharing [3], [7] and [8], sequential niche [1], and cluster analysis [16]. All these methodologies employ a single niche radius to describe all of the niches within the search domain. This is obviously not the ideal situation when one is dealing with irregular unevenly spaced peaks in multidimensional spaces. Choosing the 'best' value of niche radius has become known in the literature as the *niche radius problem.*

A number of techniques that attempt to solve this problem have been published in the recent past, including multi-national GAs [15], GAS [9], and forking GAs [14], to name just a few. Each of these methodologies loosely employs a form of variable radius niching, where each niche may have a niche radius independent of other niches in the problem space.

There are other techniques that do not employ a radius as such, and therefore are, perhaps, not nichers in the true sense of the word, but are worthy of note. In particular are tagging [13], and deterministic crowding [11].

For the sake of brevity the specific minutiae of all these mechanisms have been omitted and the reader is referred to the citations for further details. There are many more methodologies and techniques designed to preserve diversity and encourage speciation in publication that are too numerous and beyond the scope of this paper to mention.

We present here the recent developments and improvements that have been made to the Dynamic Niche Clustering (DNC) technique described in [4] and [5]. Most notably is the use of the fitness topology function called the hill-valley function described in [15]. This function facilitates the analysis of niches and individuals, and its incorporation into DNC yields a true dynamic niching mechanism that is robust, flexible, accurate and fast. We start by first discussing some of the philosophies and ideas behind DNC in order to provide a better insight into the workings of the algorithm.

2 The Philosophies Behind DNC

A large number of clustering algorithms tend to start by looking at a single cluster that encompasses the entire search space. This is extremely wasteful because all the information contained within the initial population is simply discarded. DNC, on the other hand, uses this information and starts by concentrating on the points that are already known, i.e. the initial population. These individuals will be randomly distributed about the search space. The premise is that if we start by looking at the points that are already

[1] We define a niche as a species or subset of individuals that exhibit similar traits or occupy the same region on the fitness landscape.

0-7803-6657-3/01/$10.00 ©2001 IEEE

known and create small niches centred on each of these individuals, when the population begins to converge on the various peaks in the fitness landscape the small niches will get closer together. These can then be merged to create single, slightly larger niches on each of the peaks. It follows that larger hypervolume peaks will attract more niches, so the resulting niche will be much larger than the niches occupying smaller hypervolume peaks – the result: independent niche radii that approximately match the occupied peak.

The choice of niche radius is a matter for much debate and previous clustering algorithms have tended to choose a radius based on the expected number of peaks. This assumption implies knowledge of the fitness landscape *a priori*, which is not always available. So within this work a different approach was adopted. The initial value of niche radius is calculated based on the population size and the bounds of the search space, because it is the number of individuals in the population that defines the maximum number of peaks that the GA can find and populate. The larger the population size, the more peaks the GA can potentially identify.

The single trait that is common to all of the nichers mentioned in Section 1 is that they are all hard clustering algorithms. That is to say that the niches within the various algorithms are not allowed to overlap, and therefore an individual may only be a member of one niche. DNC utilises a fuzzy set of overlapping niches. This overlap is a requirement of the workings of the merge process and will be described in detail later.

One of the most common reasons for designing a new niching algorithm is to overcome the ubiquitous $O(p^2)$ complexity of standard fitness sharing, where p is the population size. So, the final philosophy of DNC was to provide a good speedup in complexity against classical fitness sharing. DNC can do this in the later stages of evolution because we merely have to compare each niche to each individual in the population, leading to an approximate complexity of $O(np)$, where n is the number of niches. Obviously, in the initial generations, $n = p$, so the complexity should be the same as standard fitness sharing. However, because of the additional processes performed by DNC, this complexity is actually much greater. It was this problem that lead to the change in calculation of initial niche radius and the process performed on the nicheset in the initial generation in order to provide an improvement in time complexity in the early stages of evolution.

3 Dynamic Niche Clustering

The nicheset is the set of niches that are maintained from generation to generation. Each niche is persistent, so the same set of niches from generation t will be carried over to generation $t+1$. The nicheset also maintains the maximum and minimum values that the niche radii may take. These are currently fixed throughout the lifetime of the GA and are defined as:

$$\sigma_{sh_{\max}} = 2 \bullet \sigma_{sh_{initial}}$$
$$\sigma_{sh_{\min}} = 0.5 \bullet \sigma_{sh_{initial}} \quad (1)$$

3.1 Niches

Each niche maintains its current midpoint $\overline{mid}_i$ in decoded parameter space, the current niche radius σ_{shi}, the original midpoint where the niche was spawned, the generation at which the niche was spawned, and a list of references to the individuals that are currently members of the niche. Individuals are considered to be a member of a niche if they lie within the hypersphere described by $\overline{mid}_i$ and σ_{shi}, i.e.

$$\left\| \overline{mid}_i - \overline{v}_x \right\| \leq \sigma_{sh_i} \quad (2)$$

Where $\overline{v}_x$ is the position of individual x in decoded parameter space, and individual x is a member of niche i.

We also define two more radii that are based on the current niche radius. The inner niche radius is half of the current niche radius and is used to determine when two niches should be merged together or not. The outer niche radius is twice the current niche radius. The niche maintains a separate list of references to individuals that lie in this outer radius. These individuals are not considered as true members of the niche, and as such are not taken into consideration when recalculating the midpoint, but *are* subject to the fitness sharing mechanism described later. The reason for this will become apparent when we discuss the sharing function in Section 3.8.

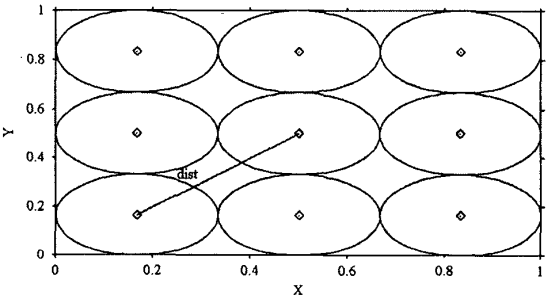

Figure 1. Average distance between individuals

3.2 The Initial Niche Radius

The initial niche radius of all new niches is calculated based on the design philosophy mentioned earlier, i.e. it is based on the population size and the bounds of the search space. If we assume that the randomly generated population of individuals is evenly spread throughout the search space, the average distance between any two adjacent individuals can be calculated as:

$$dist = \frac{\sqrt{d}}{\sqrt[d]{p}} \quad (3)$$

Here, d is the number of dimensions, and p is the population size. The search space is assumed to be normalised. This is very similar to the niche radius calculation for fitness sharing described in [2], except that the niche radius was calculated as half of *dist* in order to have 0 overlap along the diagonal. Figure 1 shows an example of the ideal spread of

individuals and the average distance between any two adjacent individuals, in a 2 dimensional space with population size 9.

If the initial niche radius is too small, then DNC will take much longer to converge, and so the number of niches will initially be approximately equal to the number of individuals. In this situation, DNC has far greater time complexity than standard fitness sharing. If, on the other hand, the initial niche radius is too big, then the niches will tend to merge quite rapidly and form a single large niche encompassing much of the search space. Both of these situations are obviously unacceptable because the peaks have been incorrectly identified. Ideally, each niche should initially cover at least one of the nearest adjacent individuals, so the initial niche radius was calculated as,

$$\sigma_{sh_{initial}} = \frac{\lambda \bullet \sqrt{d}}{\sqrt[d]{p}} \quad \lambda : [0.5..1] \tag{4}$$

Here, λ defines the amount of overlap. For this paper, λ was set to 1.0 and investigation into the effects of altering λ is left for future work.

3.3 The Initial Generation

The initial generation of individuals is randomly generated and a new niche is created for each individual, centred on that individual, with initial niche radius calculated using equation (4), and the individual as the only member of the niche. Due to the high amount of processing required in the initial generations when the number of niches is equal to the population size, it is desirable to reduce the number of niches to a level where the complexity of DNC is approximately equivalent to that of standard fitness sharing. If we assume that the individuals are uniformly distributed throughout the n-dimensional search space, then a fair amount of niche overlap is going to occur and, as a result, some niches will be redundant. The following process is performed to eliminate these redundant niches.

1) *Calculate the nearest pairset.* Each niche is compared to each other niche in the nicheset, and the euclidean distance between the midpoints is calculated. This value is stored along with a reference to the two niches that were compared in a list of nearest niche pairs. This is a $(n^2-n)/2$ process because a niche, i, need only be compared to niche j once (it is not necessary to then compare niche j to niche i), and there is no need to compare niche i to itself.

2) *Sort the nearest pairset* based on the midpoint distance using quickSort, see [12]. This is typically an $n \log_2 n$ process.

3) *Cycle through the list of nearest niche pairs.* If the midpoints of a pair of niches lie within the inner niche radius of one another *and* the hill-valley function (see Section 3.5) does not indicate that a valley lies between the two points, then delete the niche with lower fitness from the nicheset and the nearest pairset. Continue to cycle through the nearest pair list until the distance between any two niches is greater than the initial niche radius - the remaining pairs of niches are too far apart to be considered.

Generation	DNC without Speedup (ms)	DNC with Speedup (ms)	Std Fitness Sharing (ms)
0	127100	34230	10160
1	1480	1700	10110
2	720	550	9940
3	380	330	10000
4	270	270	9990
5	220	220	9830
6	220	280	9890
7	270	270	9940
8	330	220	10050
9	220	220	9840
10	220	220	9880

Table 1. Comparison of speedup process

The fitness of a niche is equal to the fitness of the individual upon which it was spawned. The exact amount of speedup that can be achieved is dependant on the fitness landscape, the initial spread of individuals, and the initial niche radius. Table 1 compares the time taken per generation of DNC without the speedup process and DNC with the speedup process. The time taken for standard fitness sharing is also included as a reference. In all 3 cases the initial starting populations were the same, the population size was 300, the initial niche radius was calculated using equation (4), and the test function was DeJong F5 (Shekel's foxholes). It is immediately apparent that considerable improvements in time complexity can be attained using the speedup process; DNC is almost 4 times faster in the initial generation with the speedup process than without it.

3.4 The Generational Process

In each generation, the following process is performed after any fitness scaling procedures.

1) *Recalculate the niche members.* Each individual x, is compared to each niche i, using equation (2) to determine niche membership. An individual may be a member of more than one niche. If any individual is not a member of a niche, then a new niche is created with its midpoint centred on the individual's location in parameter space, and niche radius calculated using equation (4).

2) *Move the midpoints of each niche.* Each niche's midpoint is modified by the following equation.

$$\overrightarrow{mid_i} = \overrightarrow{mid_i} + \frac{\sum_{x=1}^{n_i} (\vec{v_x} - \overrightarrow{mid_i}) \bullet f_x}{\sum_{x=1}^{n_i} f_x} \tag{5}$$

Here, n_i is the niche count of niche i, $\vec{v_x}$ is the position of individual x in parameter space, and f_x is the fitness of individual x. Only individuals that lie inside the current niche radius are considered in this calculation. Individuals that lie in the outer niche radius are not included.

3) *Calculate the nearest pairset.* This process is exactly the same as for step 1) of the Initial Generation speedup process described in Section 3.3.

4) *Sort nearest pairset* based on the midpoint distance using quickSort, see [12]. This is typically an $n \log_2 n$ process.

5) *Check pairs of niches.* Cycle through the nearest pairset, terminating when the distance between any given pair of niches is greater than half the maximum niche radius (because after this point, any pair of niches will not be merged because they are too far apart). Check each pair of niches; if they are close enough together *and* the hill-valley function does not detect a valley between the two midpoints, then the two niches are merged into the single niche *i*. Niche *j* is then deleted from the nicheset and the nearest pairset. The merge process is explained in detail in Section 3.6. If a merge occurred, then part of the nearest pairset may need to be recalculated because the midpoint of the merged niche may have moved.

6) *Cycle through the niches in the nicheset.* If any niche has a population size greater than 10% of the total population size, then check a number of random pairs of individuals from within the niche using the hill-valley function. The total number of individuals checked was set to 50% of the niche population size. If the hill-valley function indicates that a valley lies between two individuals, then the niche is *split* into two niches. This is described in detail in Section 3.7. The old niche is deleted and the new niches are added to the nicheset

7) *Apply sharing function.* The sharing function described in Section 3.8 is applied to the individuals within each niche. The fitness of each individual is divided by the sharing function of the niche to which it belongs. If an individual is a member of more than one niche, then its fitness is only divided by the niche with greater value sharing function. This prevents the *striation* effect documented in [5]. This is typically an *O(np)* process, however, due to the fuzziness and niche overlap, the actual complexity may be slightly higher.

3.5 The Hill-Valley Fitness Topology Function

The hill-valley fitness topology function described in [15] is used heavily in all the processes performed by DNC because it allows DNC to make more informed decisions based on analysis of the fitness landscape. The concept is simple – given two end points in euclidean space, generate a line that intersects them both. Then, choose a number of points along the line in between the two end points and calculate the fitness of those points. In this way it is possible to determine if a hill or a valley lies between the two end points. The implementation of the hill-valley function used in DNC is slightly different from that described in [15]. In [15], the function terminated on the first point discovered that had lower fitness than either of the two end points. For DNC, *all* of the interpolated points are calculated and the lowest point is returned. For example, given two endpoints *ip* and *iq*, and 3 samples in between (see Figure 2), if hill-valley terminates on the first sample it would miss the more important valley on the 3rd sample. The DNC hill-valley function returns the depth of the lowest interpolated point relative to the least fit endpoint (i.e. *iq* in Figure 2). If no point has lower fitness than either of the endpoints, then hill-valley returns 0.

It is quite obvious how powerful this function is and how the decisions of whether to merge and split niches can be made using it. However, this function will only work in decoded parameter space. The samples used in all hill-valley calculations in DNC are *{0.25, 0.5, 0.75}*. So the 1st interpolated point is one quarter of the way along the line bisecting *ip* and *iq*, and so on.

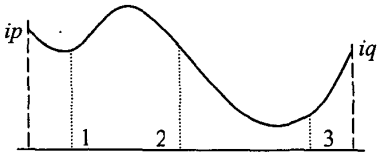

Figure 2. Hill-valley function example

3.6 Merging Two Niches

As two niches, *i* and *j*, converge on the same peak, they get closer and closer together. When they overlap enough so as to effectively be the same niche and a valley is not detected between the two niches, they are merged into a single niche. The decision to merge is made if hill-valley function does not detect a valley *and* the following condition is satisfied.

$$\left\|\overrightarrow{mid_i} - \overrightarrow{mid_j}\right\| \le \frac{\sigma_{sh_i}}{2} \quad or \quad \left\|\overrightarrow{mid_i} - \overrightarrow{mid_j}\right\| \le \frac{\sigma_{sh_j}}{2} \tag{6}$$

So two niches are merged if either midpoint lies inside the inner niche radius of the other niche. In order to calculate where the single midpoint of the newly merged niche is, we must first look at the naïve midpoint. This is simply the point that lies directly in the middle of the line bisecting the two niches.

$$\overrightarrow{mid_{naive}} = \overrightarrow{mid_i} + \frac{\overrightarrow{mid_j} - \overrightarrow{mid_i}}{2} \tag{7}$$

The new midpoint can then be calculated by analysing the spread of individuals within both niches. Again, only the individuals that lie inside the niche radius of a niche are considered in this calculation, not those that lie within the outer niche radius.

$$\begin{aligned} \overrightarrow{mid_{new}} &= \overrightarrow{mid_{naive}} + \frac{\overrightarrow{w_i} + \overrightarrow{w_j}}{\sum_{a=1}^{n_i} f_a + \sum_{b=1}^{n_j} f_b} \\ \overrightarrow{w_i} &= \sum_{a=1}^{n_i} (\overrightarrow{v_a} - \overrightarrow{mid_{naive}}) \cdot f_a \quad a \in i \\ \overrightarrow{w_j} &= \sum_{b=1}^{n_j} (\overrightarrow{v_b} - \overrightarrow{mid_{naive}}) \cdot f_b \quad b \in j \end{aligned} \tag{8}$$

This will move the new midpoint to the average weighted distance of all the individuals within both niches from the naïve midpoint.

As well as calculating a new midpoint, the niche radius of the resulting niche must also be considered. If niche *i* was completely encompassed by niche *j*, then the new niche radius is simply the old niche radius of niche *j* (See Figure 3a). This can be ascertained by using the equation of a line and the equation of a sphere in n-dimensions.

$$L(t) = Q + t\bar{v}$$
$$(X - P) \bullet (X - P) = r^2 \quad (9)$$

Here, $L(t)$ is the line that bisects the midpoints of niches i and j. For the sphere, P is the midpoint of the niche, and r is its niche radius. Substituting $L(t)$ for X and solving the resulting quadratic equation yields two possible values of t. These values of t are where the line intersects with the edges of the niche with midpoint P and radius r. With two niches, there are four possible values of t, t_{i1}, t_{i2} where the line intersects niche i, and t_{j1} and t_{j2} where the line intersects niche j. By comparison of these four values, it is a simple matter to determine whether a niche is encompassed by another niche.

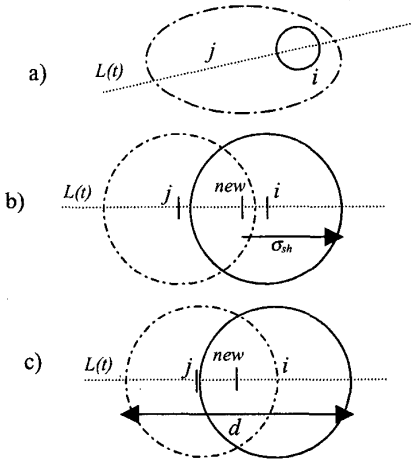

Figure 3. Merging of niches

If neither niche is completely encompassed by the other, then the new midpoint is compared to the original midpoints of niches i and j. If the new midpoint is closer to niche i, then the new niche radius is calculated as the distance from mid_{new} to the furthest extent of niche i's hypersphere, along $L(t)$ away from niche j (see Figure 3b). The converse is true if the new midpoint is closer to niche j.

If mid_{new} is equidistant from niches i and j, then the new niche radius is calculated as half the distance between the furthest extents of either niche's hypersphere, in both directions along $L(t)$ from the naive midpoint (see Figure 3c).

3.7 Splitting a Niche

As mentioned previously, a niche is chosen for splitting if it has population size greater than 10% of the total population size, and a valley is detected between a pair of randomly selected individuals from within the niche. The premise behind this is that a niche that has a significantly large population size may potentially be too large and cover more than one peak. By selecting random pairs of individuals from within the niche and using the hill-valley function, it is possible to detect whether this is in fact the case.

When selecting pairs of individuals, all individuals within the niche may be selected with equal probability. This allows an individual to be selected more than once, but with the restriction that an individual may not be compared to itself. Having analysed all the pairs of individuals, the pair that has the deepest valley between them is used in the split function. The current niche is deleted and two new niches are created with midpoints centred on these two individuals. The niche radius of each niche is calculated as,

$$\omega \bullet \left\| \bar{v}_i - \bar{v}_j \right\| \quad \omega : [0.5..1] \quad (10)$$

Here, $\bar{v}_i$ is the position of individual i in parameter space and ω defines the amount of niche overlap between the two new niches. For this work, $\omega = 0.5$. If either niche radius is less than the minimum niche radius, then the niche radius is set to the minimum niche radius. Finally, the members of each niche are recalculated by comparison of all the individuals within the original niche and each of the new niches.

This is a much simplified split function because it ignores a lot of the information that is actually available. For instance, it is possible to calculate the distance from the midpoint of the original niche to each of the individuals within the split pair. From this it is possible to calculate the angle between these two vectors. More sophisticated split techniques that use this information can be employed, but they are a topic for future research.

3.8 To Share or Not to Share

There is some debate as to whether or not to use a fitness sharing mechanism to derate the fitness of the individuals, or to use a selection mechanism that promotes diversity. This discussion is beyond the scope of this paper, and the fitness sharing mechanism was utilised.

In [5] we reported a potential problem with the fitness sharing mechanism we had previously employed. The function is similar to the sharing function used in [15] and is defined as,

$$m_i = n_j - n_j \bullet \left(\frac{d_{ij}}{2\sigma_{sh_j}} \right)^{\alpha} \quad if\ ind.\, i \in niche\, j \quad (11)$$

Here, d_{ij} is the euclidean distance between individual i and the midpoint of niche j. This sharing function has a problem, however. Individuals at the edge of the niche are penalised by $n_j/2$ as one would expect. But, an individual that lies just outside of the niche is not penalised at all. The effect of this was briefly commented on in [5] and can be seen in Figure 4. Where a niche does not completely cover the whole peak, there can be areas of higher fitness in the uncovered regions of the peak which can lead to the creation of phantom peaks in the fitness landscape. Figure 4 shows a simple function with a single peak and a niche that does not fully encompass the peak. The dotted line shows the modified fitness landscape after sharing has been applied. It is quite evident that there are two phantom peaks at the edges of the niche that have high fitness relative to the top of the peak after sharing has been applied.

In order to solve this problem, the concept of the outer niche radius was introduced. As stated previously in Section 3.1, the outer niche radius is twice the current niche radius and the niche maintains a separate list of references to individuals that lie in this outer radius. If we extend the slopes of the sharing function down to 1, the size of the discontinuity at the edge of a niche can be minimised. So all individuals that lie inside the current niche radius and outer niche radius will have their fitness divided by the sharing function, m_i. The niche population size is the number of individuals that lie inside the current niche radius. Individuals within the outer niche radius are not counted. The resulting sharing function, m_i, is shown in Figure 5. Here, $\sigma_{sh} = 0.1$, and the niche population size = 10.

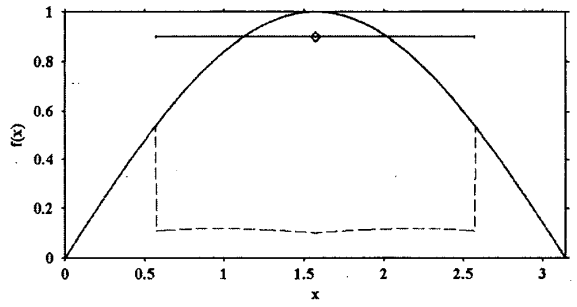

Figure 4. Simple function with modified fitness landscape

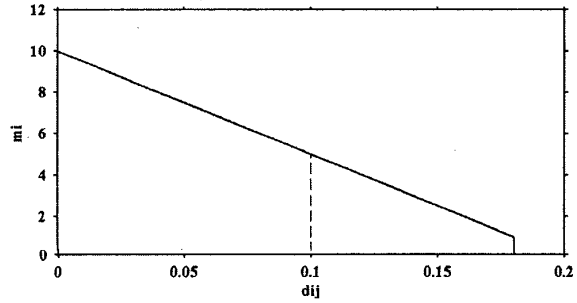

Figure 5. The Triangular Sharing Function

4 Niche Level Operators: Local Elitism

Due to the explicit nature of niches within DNC, it is possible to apply operators at the niche level that act upon the individuals within the niche. This is a powerful concept and is achieved with very little additional computational expense. We shall only look at local elitism within this paper, other niche level operators are left for future work.

If a niche has a population size greater than some proportion of the overall population size, then a proportion of the best individual's within the niche are automatically retained in the next generation. In this way, local elitism can be implemented and the best individuals within the larger niches are always preserved. For this paper, if a niche has a population size greater than 5% of the overall population size, then the top 10% of individuals in the niche are automatically retained in the next generation.

5 The Test Functions

In order to best visualise the results, four 2-dimensional test functions were used. These were the Sines function (equation 12) as described in [13], the Himmelblau function (equation 13) as defined in [1], the Ripple function (equation 14) taken from [6], and a modified version of the Bell function (equation 15) taken from [10].

$$sines(x,y) = 1 + \sin^2(x) + \sin^2(y) - 0.1e^{-x^2-y^2} \quad (12)$$

$$himm(x,y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2 \quad (13)$$

$$ripple(x,y) = e^{-2\log(2)\cdot\left(\frac{x-0.1}{0.8}\right)^2} \cdot \sin^6(5\pi x) + 0.1\cos^2(500\pi x) + e^{-2\log(2)\cdot\left(\frac{y-0.1}{0.8}\right)^2} \cdot \sin^6(5\pi y) + 0.1\cos^2(500\pi y) \quad (14)$$

$$bell(x,r,h) = \begin{cases} h - \frac{2hx^2}{r^2} & if\ x < \frac{r}{2} \\ \frac{2h(x-r)^2}{r^2} & if\ \frac{r}{2} \le x < r \\ 0 & otherwise \end{cases} \quad (15)$$

For the bell function, r is the radius of the cone, h is the height, and x is the euclidean distance from the midpoint of the cone. For the test function, 30 peaks were randomly generated in a 2D space. The midpoints were randomly initialised in the first generation. The radii of the bells were randomly generated with values in the range 0.025 to 0.1, and the bell heights were also randomly generated with values in the range 0.1 to 1. The resulting bells function is shown in Figure 6.

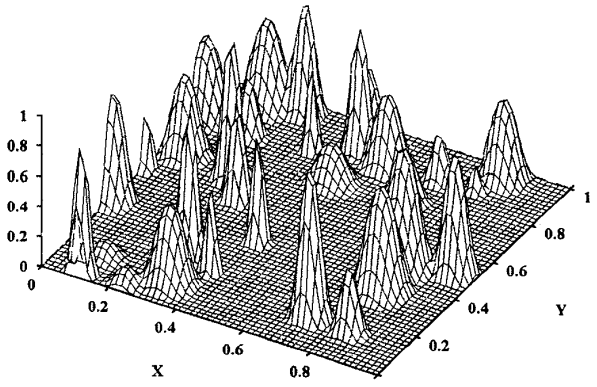

Figure 6. The Bells Function

6 Results

All GA runs were executed for 50 generations, with population sizes of 300, remainder stochastic sampling with replacement, 15 bits per parameter, single point crossover with p_c=*0.7*, G_{gap}=*1.0* and mutation m_p=0.001. All results are averaged over 10 runs. The tests were performed on

basic DNC (as described in [5]), DNC with selective merge/split and speedup functions, and DNC with elitism and selective merge/split and speedup functions, from the same starting populations. In terms of performance for each DNC configuration, we record the ratio of peaks found to actual peaks in the fitness landscape (a peak is considered to be found if the midpoint of a significant niche lies within the top 10% area of the fitness of the peak – a niche is significant if it has a population size greater than 3), the accuracy of the niches (the average RMS error between the top of each peak and the most fit individual within the nearest niche), and the time taken to complete 50 generations.

	Old DNC	Selective DNC	Selective DNC w/elitism
% Peaks	87.8	94	100
Accuracy	0.62	0.56	0.48
Time (ms)	144290	58550	59480

Table 2. Performance Measures against the Sines function

	Old DNC	Selective DNC	Selective DNC w/elitism
% Peaks	100	100	100
Accuracy	0.83	0.86	0.73
Time (ms)	129510	47890	47950

Table 3. Performance Measures against the Himmelblau function

	Old DNC	Selective DNC	Selective DNC w/elitism
% Peaks	88	93.2	98.9
Accuracy	0.087	0.068	0.049
Time (ms)	142040	98420	99460

Table 4. Performance Measures against the Ripple function

	Old DNC	Selective DNC	Selective DNC w/elitism
% Peaks	70.5	92.9	99.5
Accuracy	0.12	0.098	0.072
Time (ms)	116770	41690	42010

Table 5. Performance Measures against the Bells function

It is quite clear from the results in these tables that Selective DNC (the modified technique described in this paper) far outperforms the original DNC scheme described in [4] and [5]. The ability of the algorithm to dynamically increase and decrease the niche radii allows for the more accurate identification of the peaks within the problem space. However, like all stochastic processes the sharing and selection mechanism are subject to drift and it is possible to lose niches in this way. This is especially the case if the niche population sizes are low (e.g. the ideal niche population size for the sines function is 6 with a total population size of 300). The introduction of local elitism at the niche level helps prevent the loss of good niches. Also of particular note is the consistent improvement of the time taken to run the GA. The pre-processing of redundant niches in the initial generation greatly reduces the time complexity of the early stages, which allows the improvement in speed over the original technique. It is also quite evident that the additional costs of employing local elitism are negligible.

7 Summary

In this paper, we have described a number of improvements and modifications to the Dynamic Niche Clustering technique, and have shown empirically that the algorithm outperforms the original technique in terms of speed and accuracy of peak detection. These improvements have been achieved through utilisation of the hill-valley fitness topology function, which allows the local analysis of the fitness landscape. There are a couple of drawbacks to using this approach, however. First, is that hill-valley operates in euclidean space and second, is that it involves many more calculations of the fitness function. This can potentially become a problem in the event of expensive evaluation functions or noisy fitness landscapes. But despite this, it is still a very powerful tool. The improvements can be seen in Figures 7 and 8.

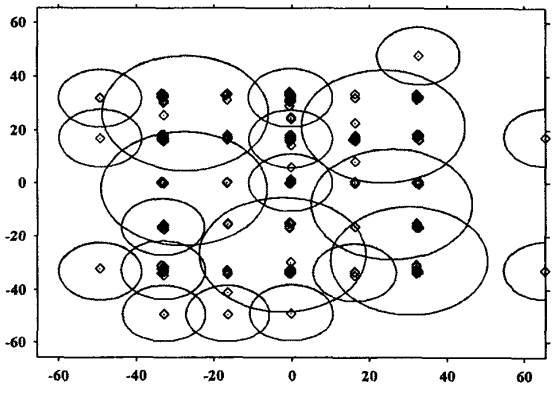

Figure 7. DNC without the split function on DeJong F5

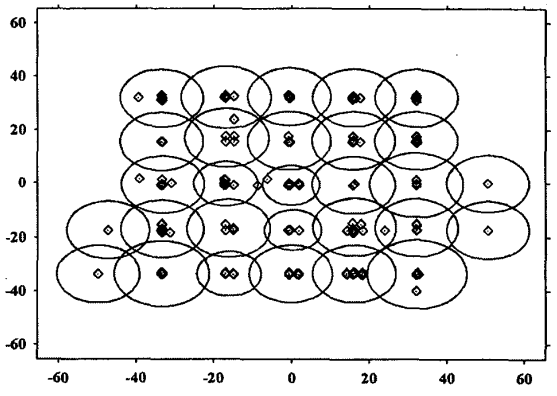

Figure 8. DNC with the split function on DeJong F5

Figure 7 shows the final population and niche spread of DNC without the new split function on De Jong F5, Shekel's Foxholes. It is quite apparent that there are 6 oversize niches that cover more than one peak. Here DNC has not correctly identified the peaks, however it is still

quite interesting to note that the individuals have still populated the tops of all 25 of the peaks. Figure 8 shows the final population and niche spread of DNC with the new split operator. Here, it is evident that the oversize niches have been split up, and the niches are approximately the correct size and in the correct position.

DNC adopts a uniquely different approach to providing a potential solution to the niche radius problem. This approach is based on the concept that with a given population size, there is a limit to the number of niches that can be maintained at any one time. So in this way, less knowledge is required *a priori*; DNC only requires knowledge of the dimensionality and bounds of the search space. The explicit definition of niches as objects and not just as clustering tools allows us to utilise the concept of niche level operators; mechanisms that work at the niche level. These operators can be seen to act as abstract layers above groups of individuals. To describe just a few possibilities of this approach; local elitism has already been described (see Section 4), mating restriction is another obvious candidate as a niche level operator, extinction, and selection mechanisms. These operators can be added to the existing framework with very little additional overhead.

One major assumption made by DNC is that the search space must have bounds of the same magnitude – if they are drastically different, then DNC will choose an inappropriate size of niche radius. A further possible criticism of the technique is that the algorithm itself is quite complicated. But when one considers the nature of the niche radius problem, a solution is required to have independent radii, which fixed radius niching schemes simply cannot achieve.

It should be noted that the emphasis of DNC is towards diversity, rather than the optimal identification of all the peaks. DNC tends to spread the individuals in a niche around the top of a peak, rather than cluster them tightly at the very apex. This means that the average niche fitness is slightly lower when compared to other niche techniques such as classical fitness sharing, but the increases in computation speed are tremendous.

In terms of future work to be undertaken on the technique, the initial population size is an important issue. It is this that determines the initial niche radius and the maximum and minimum bounds of all the niche radii. Dynamically altering the overall population size midway through a GA run could potentially be a very powerful dynamic mechanism for finding any number of peaks within a fitness landscape. But this is a topic for future research.

References

[1] Beasley, D., Bull, D.R. & Martin, R.R.: *A Sequential Niche Technique for Multimodal Function Optimization*, In Evolutionary Computation **1**(2), pp101-125, 1993.

[2] Deb, K.: *Genetic Algorithms in Multimodal Function Optimization*, Masters Thesis, TCGA Report No. 89002, The Uni. of Alabama, Dept. Engineering Mechanics, 1989.

[3] Deb, K. & Goldberg, D.: *An Investigation of Niche and Species Formation in Genetic Optimization*, In proc. of the 3rd Inter. Conf. on Genetic Algorithms, pp42-50, 1989.

[4] Gan, J. & Warwick, K.: *A Genetic Algorithm with Dynamic Niche Clustering for Multimodal Function Optimisation*, In proc. of the 4th Inter. Conf. on Artificial Neural Networks and Genetic Algorithms, pp248-255, Springer Wien New York, 1999.

[5] Gan, J. & Warwick, K.: *A Variable Radius Niching Technique for Speciation in Genetic Algorithms*, In proc. of the Genetic and Evolutionary Computation Conference, pp96-103, Morgan Kaufmann, 2000.

[6] Ghosh, A., Tsutsui, S., Tanaka, H. & Corne, D.: *Genetic Algorithms with Substitution and Re-entry of Individuals*, In inter. Journal of Knowledge-based Intelligent Engineering Systems Vol.4, No.1, pp64-71, Information Sciences, 2000.

[7] Goldberg, D. & Richardson, J.: *Genetic Algorithms with Sharing for Multimodal Function Optimization*, In proc. of the 2nd Inter. Conf. on Genetic Algorithms, pp41-49, 1987.

[8] Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[9] Jelasity, M. & Dombi, J.: GAS, *A Concept on Modelling Species in Genetic Algorithms*, In Artificial Intelligence, 99(1), pp1-19, 1998.

[10] Jelasity, M.: *UEGO, an Abstract Clustering Technique for Multimodal Global Optimization*, In A.E.Eiben, T.Bäck, M.Schoenauer & H.Schwefel (eds), Parallel Problem Solving from Nature – PPSN V. vol 1498 of Lecture Notes in Computational Science, pp378-387, Springer-Verlag, 1998.

[11] Mahfoud, S.W.: *Crowding and Preselection Revisited*, In R.Manner & B.Manderick (eds), Parallel Problem Solving From Nature, 2, pp27-36, Elsevier Science Publishers B.V., 1992.

[12] Press, W.H., Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P.: *Numerical Recipes in C: The Art of Scientific Computing 2nd ed.*, Ch. 8.2, pp332-336, Cambridge Uni. Press, 1992.

[13] Spears, W.M.: *Simple Subpopulation Schemes*, In proc. of 3rd Annual conf. on Evolutionary Programming, pp296-307, World Scientific, 1994.

[14] Tsutsui, S., Suzuri, J. & Ghosh, A.: *Forking GAs: GAs with Search Space Division Schemes*, Evolutionary Computation, Vol.5, No.1, pp61-80, MIT Press, 1997.

[15] Ursem, R.: *Multinational Evolutionary Algorithms*, In proc. of Congress of Evolutionary Computation, vol.3, pp1633-1640, 1999.

[16] Yin, X. & Germay, N.: *A Fast Genetic Algorithm with Sharing Scheme using Cluster Analysis Methods in Multimodal Function Optimization*, In proc. of Inter. Conf. on Artificial Neural Networks and Genetic Algorithms, pp450-457, 1993.