Analysis of the Niching Particle Swarm Optimization Algorithm

Tyler Crane

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Department of Computer Science
Faculty of Mathematics and Science
Brock University
St. Catharines, Ontario

Abstract

Multimodal optimization (MMO) techniques have been researched and developed over the years to track multiple global optima concurrently. MMO algorithms extend traditional unimodal optimization algorithms by using search strategies built around forming niches for multiple possible solutions. NichePSO was one of the first approaches to utilize particle swarm optimization (PSO) for MMO problems, using several small subswarms of agents working concurrently to form niches within the search space. Despite its promising performance NichePSO does suffer from some problems, and very little research has been done to study and improve upon the algorithm over the years. A main goal of this thesis is to analyze the NichePSO algorithm, gaining insight into the strengths and weaknesses of the algorithm. Empirical analyses were performed to study the NichePSO's ability to maintain niches within complex problem domains, as well as methods for improving the overall performance and effectiveness of the algorithm. Two variants of the NichePSO algorithm are proposed, and experimental results show that they both significantly improve the performance of the NichePSO algorithm across several benchmark functions.

Acknowledgements

I would like to thank my supervisor Dr. Ombuki-Berman for her help and guidance in both my graduate and undergraduate education. I would like to thank Dr. Engelbrecht for his invaluable feedback and direction throughout my research. Thank you to my parents for their unending love and support, and to my loving girlfriend for always making me smile.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

When presented with a problem in a real world situation, it is often not enough to simply propose a single solution and hope for the best. Sometimes solutions just don't work out, and when they fail we rely on alternative solutions that we prepare as a backup. Despite this common occurrence, many optimization algorithms are designed to only find a single best solution to a problem. While this behaviour is sufficient in a lot of cases, in the real world we want to consider multiple options to a problem, whether they be multiple strong solutions or simply backup solutions just in case.

Optimization problems are problems that contain a vast set of possible solutions, with a single or small set of global best solutions existing within the set. Finding such global best solutions are often non-trivial, and can be very computationally expensive to find using traditional search algorithms. While finding a solution is easy, it is often extremely difficult to prove whether the found solution is optimal or not. In order to guarantee an optimal solution, all other solutions must be considered and evaluated. This process is not only time consuming, it is impractical for most real world optimization problems we face.

Over the past several decades new approaches have been developed and researched that tackle such complex optimization problems. Since the major complexity of optimization problems stems from proving whether a solution is optimal or not, several metaheuristics have been developed that do not guarantee optimal solutions, but give practical answers in feasible time [3] [9]. While these algorithms cannot guarantee that the solutions returned are optimal, they often present solutions that are sufficient to solve the problems we face.

Traditional optimization algorithms within the domain of artificial intelligence generally deal with finding a single best solution to a given problem, known as uni-

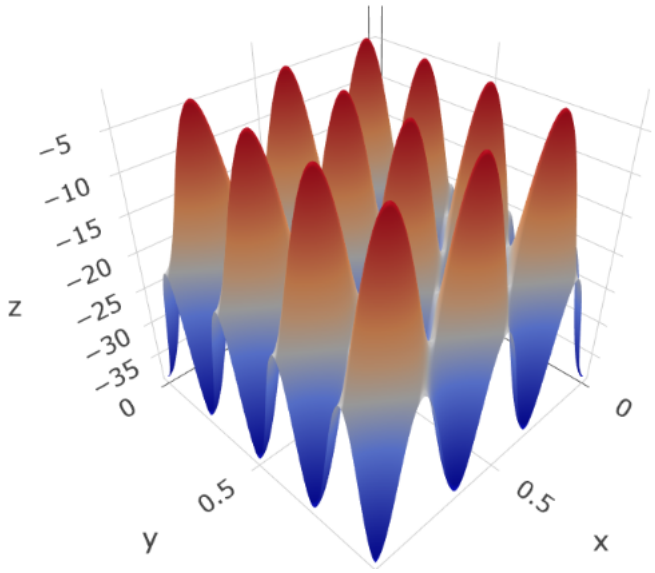

Figure 1.1: The modified Rastrigin function, containing 12 global optima.

modal optimization. While this approach is effective for many different scenarios, for many real world problems finding a single optimum is insufficient [35]. Fields such as engineering face problems with many different unique constraints, and often the optimal solution may not meet all required conditions that are needed to be met [36] [18]. Multimodal functions are those that contain multiple global optima, and instead of a single best solution, these problems contain a set of optimal solutions (see Figure 1.1). A unimodal approach may be able to find one of these global optima, but there may exist many more that should not be ignored.

A trivial solution would be to use a unimodal optimizer and perform multiple runs on the same problem, recording all unique solutions returned. This approach however is very inefficient, as there is no guarantee that a different initialization would lead to a different solution. Multimodal optimization functions take this into consideration, and are designed to return a set of strong solutions while keeping track of other optima found to increase efficiency.

Research into niching methods began back in the 70's and 80's and primarily focused on evolutionary algorithms (EA) for their ability to effectively adapt to complex landscapes [12]. One of the earliest proposed niching techniques was deterministic crowding [16], where offspring are compared to a sample of the current population for diversity and similarity. With this technique, the most similar offspring are replaced with new ones to promote population diversity. Crowding found several applications amongst various algorithms and lead to further research being conducted on addi-

tional niching techniques. These techniques include fitness sharing [13], restricted tournament selection [15], derating [1], clustering [37] and many more.

Around the early 2000's researchers began utilizing alternative meta-heuristic optimization algorithms for multimodal optimization. Several techniques were applied to particle swarm optimization (PSO) due to its unique ability to maintain memory through its use of particles [21]. The traditional PSO algorithm was shown in [10] to be unable to maintain niches, and if one wanted to utilize PSO for multimodal functions then such specialized algorithms and techniques were needed.

The concepts of objective function stretching [28] and derating [1] were applied where the problem landscape is modified to isolate particles that are searching promising areas of the search space. If a particle was to locate a global maximum, then the landscape is stretched so that this location is viewed as a local minimum, making it an unfavourable position for other particles and causing them to search other areas of the search space. This method was a significant improvement over the traditional PSO niching methods, however it was shown in [4] that it had the potential to create false optima. Improvements to this method were proposed in [27] where the concepts of deflection and repulsion were used to mitigate this problem.

Speciation was another idea implemented as the speciation-based PSO (SPSO) [19] [26]. Speciation is where different subswarms are treated as species and clustered together based on distance to their nearest neighbours. Particles that are not nearby any others are isolated and create their own species. A similar method was used in [6] by defining the n-best topological neighbourhoods and clustering particles with their closest neighbours.

The first PSO algorithm to utilize parallel niching methods was the NichePSO algorithm proposed in [5]. The NichePSO algorithm utilizes multiple subswarms of particles that search the problem landscape in parallel, which is more efficient than using the sequential methods previously proposed. The NichePSO algorithm showed promising results, detecting all optima across multiple benchmark functions. A follow up scalability study was performed in [7] which showed strong results for benchmark functions up to four dimensions. Another analysis was done in [11], and multiple subswarm merging and absorption strategies were proposed to mitigate premature convergence and increase landscape exploration.

A common problem prevalent to many niching algorithms is that a radius parameter needs to be defined in order to run the algorithm. The need for such a parameter means that many algorithms rely heavily on a strong parameter values being set, which requires time and resources to properly tune. To overcome this issue

and remove the need for such a parameter, the adaptive niching PSO (ANPSO) was proposed in [2]. The ANPSO uses statistics involving the swarm population in order to adaptively decide the radius size each iteration. A similar approach was utilized by the vector-based PSO [30] [31], where vector operations are applied to the particles to identify niches. Once the niches are defined, particles that lie within each individual niche are clustered and assigned to the same subswarm.

Recent advancements in niching methods for MMO problems include the HillVallEA algorithm [24] [23] which uses hill-valley clustering to create niches, and an evolutionary algorithm for each cluster to perform a search of its defined area. Another technique is proposed in [38], where a PSO utilizing ring neighbourhood topology is described, referred to as a close neighbour mobility optimization algorithm. A hybrid approach combining PSO and evolutionary algorithms is proposed in [22] using nearest best neighbour clustering. Another recent PSO niching algorithm was proposed in [8], where a hybrid ring topology is applied where a sparse topology is used to maximize exploration, and then a dense topology is applied to improve convergence.

1.1 Objectives and Contributions

This thesis proposes several objectives aimed at analyzing and improving the performance of the NichePSO algorithm. Despite its strong results, little research has been done since its proposal to analyze and improve the algorithm. The focus of this work is to study the NichePSO algorithm in detail, and perform several analyses aimed to solve many of the questions still unanswered since its proposal. The goals and objectives of this thesis are as follows:

- Explore and analyze the strengths and weaknesses of the NichePSO algorithm, providing further insight into the inner mechanisms of the algorithm.
- Determine the causes behind the NichePSO's inability to maintain niches, and propose solutions to mitigate the problem.
- Experiment with the NichePSO's sensitivity to the various parameters used to provide further insight into the optimal parameter values.
- Explore alternative velocity update formulas and swarm merging strategies, and analyze the changes in performance using the different techniques.

- Propose a modified version of the NichePSO algorithm with improved performance over the original, and compare the algorithms using several benchmark functions and performance measures.

1.2 Thesis Structure

The structure for this thesis is as follows:

Chapter 2 contains relevant background information on PSO's, and describes the NichePSO algorithm in detail.

Chapter 3 describes in detail the subswarm merging problem, and shows evidence of its existence and prevalence in the NichePSO algorithm. Several solutions are also presented and analyzed to address and overcome this issue.

Chapter 4 presents and analyzes several proposed modifications to the NichePSO algorithm. Modifications are described in detail, and studied for impact on the performance and efficiency of the algorithm.

Chapter 5 Proposes two modified NichePSO algorithms that each utilize different proposed modifications. A detailed description of the modified algorithms are given, as well as a comparison between the proposed algorithms and the original NichePSO algorithm.

Chapter 5 provides concluding remarks and presents possible directions for future work.

Chapter 2

Background Information

This chapter covers the relevant background for this research. An introduction to particle swarm optimization is given, along with a detailed description of the NichePSO and GCPSO algorithms.

2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic population based optimization algorithm. Proposed by Kennedy and Eberhart [17] in 1995, PSO is inspired by the natural flocking of birds as they search for food. We can consider the optimal value of a function some amount of food, and we use particles as agents to mimic the flocking behaviour as they search for the optimal location within the search space. PSO has received lots of interest over the past several decades due to its simplicity in implementation, while also proving to be a highly effective continuous optimization tool [32].

The PSO algorithm revolves around a set of agents known as particles that are initialized randomly throughout the search space. Several iterations are run until some stopping criteria is met, such as a maximum number of iterations or time elapsed. Each iteration, the particles update their current velocity v_i and position x_i using

$$v_i(t+1) = v_i(t) * w + C_1 * rand * (\bar{x}_i - x_i) + C_2 * rand * (\bar{G} - x_i) \tag{2.1}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.2}$$

In the above equations, the terms w, C_1 and C_2 are all parameters to the algorithm.

w is the inertia factor, C_1 is the cognitive term and C_2 is the social term. The two remaining terms are $\bar{x}_i$ and $\bar{G}$. The $\bar{x}_i$ variable represents the personal best position found so far in the run by the individual particle, and the term $C_1 * rand * (\bar{x}_i - x_i)$ causes the particle to be drawn towards its personal best found so far. The $\bar{G}$ variable represents the global best found among all particles in the swarm, and the term $C_2 * rand * (\bar{G} - x_i)$ causes all particles in the swarm be drawn towards the global best position found throughout the search space.

Once the velocity of a particle is calculated, the particle's position is updated using Equation 2.2. If the new position is more favourable than the personal best of the particle, than this position becomes the particle's new personal best. If this position is more favourable than the global best position $\bar{G}$, the global best is updated to this position. This continues over several iterations. Eventually all particles will converge to the global best found throughout the entire run, and this value is returned as the optimum.

```
Randomly initialize particles in search space;
while stopping criteria not met do
    foreach particle in swarm do
        Calculate particles new velocity;
        Update particles position;
        Evaluate the new position of the particle;
    end
end
```

Algorithm 1: The Particle Swarm Optimization Algorithm

2.1.1 Guaranteed Convergence PSO

It has been observed that the using the vanilla PSO velocity update formula can exhibit unwanted behaviour. When all particles in a swarm converge to a single value $\bar{G}$, then the velocity of $v_i(t+1)$ will consist purely of the term $v_i(t) * w$. As particles approach the global best found the velocity of each particle will approach 0, and eventually all particles will converge and stop moving. At this point no more searching will take place, and there is no guarantee that the particles have converged to a global optimum.

To address this behaviour, the guaranteed convergence PSO (GCPSO) was proposed [34] to provide movement even when all particles have converged to a single value. Each particle will use the same velocity update formula as before, except for

the global best particle in the swarm. The global best particle will update its velocity and position using

$$\bar{v}_i(t+1) = -x_i(t) + \bar{G} + \bar{v}_i(t) * w + \rho(t) * (1 - 2 * rand) \tag{2.3}$$

$$x_i(t+1) = \bar{G} + \bar{v}_i(t+1) * w + \rho(t) * (1 - 2 * rand) \tag{2.4}$$

The term $-x_i(t) + \bar{G}$ resets the particle's position to that of the global best found $\bar{G}$. The term $\bar{v}_i(t) * w$ determines the direction of the search, and the term $\rho(t) * (1 - 2 * rand)$ adds a random search to the formula. The function $\rho(t)$ acts as a scaling function to control the magnitude of the random search, and is defined by

$$\rho(t+1) = \begin{cases} 0.5 * \rho(t) & \text{if \#successes} > s_c \\ 2.0 * \rho(t) & \text{if \#failures} > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{2.5}$$

where $\#success$ and $\#failures$ represent the consecutive number of successes or failures for the particle. A failure is counted when $f(\bar{x}_i(t+1)) = f(\bar{x}_i(t))$, and a success is counted otherwise. The terms s_c and f_c are threshold parameters, and optimal values are dependant on the objective function. The values $s_c = 15$ and $f_c = 5$ are used for all experiments, as per the recommendation of [34]. A starting value of $\rho(0) = 1.0$ is also used to match the original paper.

2.2 The NichePSO Algorithm

The NichePSO algorithm proposed in [5] was the first application of particle swarm optimization to apply parallel swarms to multimodal optimization problems. Instead of a single swarm of particles traversing the search space, several subswarms are created dynamically throughout a run. Each subswarm is formed around unique niches, so every time a new niche is detected a new subswarm will be created. Algorithm 2 presents an overview of the NichePSO algorithm, and individual sections of the algorithm are described in greater detail below.

2.2.1 Initialization

The success of the NichePSO algorithm relies on particles being spread evenly throughout the problem space. This is done using a lattice initialization where particles are

```
Initialize particles in main swarm;
while stopping criteria not met do
    Update position of all particles in main swarm;
    Evaluate fitness of particles in main swarm;
    for each subswarm S do
        Update position of all particles within S;
        Evaluate fitness of all particles within S;
        Recalculate the radius of S;
    end
    if any subswarms intersect then
        Merge subswarms;
    end
    for each particle p in the main swarm do
        if p intersects with radius of a subswarm S then
            Particle p is absorbed by subswarm S;
        end
        if partitioning criteria for p is met then
            Create new subswarm from particle p;
        end
    end
end
```

Algorithm 2: The NichePSO Algorithm

spread equidistantly throughout the search space, creating an n-dimensional lattice that spans each dimension.

The velocity for each particle is also initialized randomly, within the range of $v_i \in [-0.5, 0.5], v_i \neq 0$. We force the constraint that $v_i \neq 0$ due to the velocity update for the particles within the main swarm (see Equation 2.6). Since upon initialization $\bar{x_i} = x_i$, the second term of the formula would equal 0, which means that the velocity update for $v_i(1)$ would rely entirely on the value of $v_i(0)$. If $V_i(0) = 0$, then $v_i(1) = 0$, and so on until every particle was partitioned into a subswarm of its own, and a social factor was applied to the velocity update formula.

2.2.2 Main swarm velocity update

The particles in the main swarm are updated using a cognitive velocity update formula, shown with Equations 2.6 and 2.7. The formula is based off the velocity update formula for traditional PSO particles (see Equation 2.1), however no social aspect is used. This allows the particles in the main swarm to do a local search of their immediate area, with the goal of detecting local optima.

$$v_i(t+1) = v_i(t) * w + C_1 * rand * (\bar{x}_i - x_i) \tag{2.6}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.7}$$

2.2.3 Partitioning Criteria

A particle is removed from the main swarm and forms a subswarm when the particle has converged to some value. Each particle has its fitness monitored, and the changes in fitness are tracked each iteration. A particle is considered to be converged when the variance of the changes $\sigma_i < \delta$ calculated over some number of iterations e. In all experiments the values $e = 3$ and $\delta = 0.0001$ were used to match that of [5].

When a particle p_i converges, it is removed from the main swarm and a new subswarm is created. This new subswarm will consist of two particles: p_i, and the closest neighbour to p_i within the main swarm.

2.2.4 Subswarm velocity update

Particles that are no longer in the main swarm, but rather a part of an individual subswarm will use the velocity update formula used by the guaranteed convergence PSO (GCPSO). The formula incorporates both a personal and social aspect, where information about the different particles are shared throughout the individual subswarm.

2.2.5 Subswarm radius calculation

Each individual subswarm maintains a radius to ensure that they are searching a unique area within the search space. The radius of a subswarm is calculated using the max distance between the subswarm best and every particle within the subswarm. The formula for this calculation is

$$R_i = Max\{|S_{x_j,i} - S_{\bar{x},i}|\} \tag{2.8}$$

where $S_{\bar{x},i}$ is the subswarm best.

2.2.6 Subswarm particle absorption

A particle x_j in the main swarm is absorbed by a subswarm if its position falls within the radius of a subswarm S_i, calculated by

$$|x_j - S_{\bar{x},i}| < R_i \tag{2.9}$$

2.2.7 Subswarm intersection

Two subswarms S_i and S_j intersect when the radii of each subswarm intersects, calculated with

$$|S_{\bar{x},i} - S_{\bar{x},j}| < R_i + R_j \tag{2.10}$$

2.3 Experimental Analysis

All experiments performed were done so using several benchmark functions outlined for the Congress of Evolutionary Computation conference in 2013 [20]. These consist of 20 functions ranging from varying degrees of dimensions, complexity and bounds. A detailed description of each of the functions can be found in [20], along with links to source code for function implementation in Matlab, Python, Java and C++. A summary of the functions is shown in Table 2.1.

Among the 20 functions mentioned above, 7 of them were selected to be used for the majority of intermediary experiments discussed within this thesis. These functions are:

- Five uneven peak trap function
- Himmelblau function
- Shubert function (2D)
- Shubert function (3D)
- Composite function CF3 (2D)
- Composite function CF3 (3D)
- Composite function CF3 (5D)

The first two functions selected are small and simple functions with a small number of global optima. The two Shubert functions used are more complex than the first two, and contain a relatively large number of optima. The final three composite functions do not contain many optima, but do contain complex landscapes with many local

Function	# of Dimensions	# of Global Optima	Total Function Evaluations	Global Peak Fitness
Five Uneven Peak Trap	1	2	50K	200
Equal Maxima	1	5	50K	1
Uneven Decreasing Maxima	1	1	50K	1
Himmelblau	2	4	50K	200
Six-Hump Camel Back	2	2	50K	1.031628453
Shubert	2	18	200K	186.7309088
Vincent	2	36	200K	1
Shubert	3	81	400K	2709.093505
Vincent	3	216	400K	1
Modified Rastrigin	2	12	200K	-2
Composite Function 1	2	6	200K	0
Composite Function 2	2	8	200K	0
Composite Function 3	2	6	200K	0
Composite Function 3	3	6	400K	0
Composite Function 4	3	8	400K	0
Composite Function 3	5	6	400K	0
Composite Function 4	5	8	400K	0
Composite Function 3	10	6	400K	0
Composite Function 4	10	8	400K	0
Composite Function 4	20	8	400K	0

Table 2.1: A summary of the 20 benchmark functions used in this thesis.

optima that an algorithm could become trapped in. These 7 functions were chosen as a subset of the original 20 to reduce experimental time and complexity, while still providing a sufficient overview of how the algorithms and modifications perform on different functions of various complexity levels.

The stopping criteria used for each run is a maximum number of function evaluations, which differs per function. Limiting the number of function evaluations creates a fair environment for testing an algorithm or modification opposed to using a maximum time or number of iterations. One could simply add additional resources to a search, which is bound to lead to an increase in performance if not monitored. Limiting the number of function evaluations allows one to better study whether the results obtained are due to the algorithm itself performing well as opposed to performing well due to additional resources.

In order to effectively test whether a global optimum has been detected or not, an accuracy threshold is used. Each benchmark function has a global best score known beforehand, shown in Table 2.1. At the end of a run, a set of solutions is returned, one for each existing subswarm. A global optimum is considered located if the fitness

evaluation of a subswarms best position is greater than the global best score minus the accuracy threshold. A smaller accuracy threshold means that the subswarms will need to be closer to the global best score in order to have the optimum counted and located. For each experiment, the accuracy threshold used will be listed among the other parameters selected.

Chapter 3

The Merging Subswarm Problem

This chapter covers all information regarding the merging subswarm problem and the analyses done to address it. The problem is described in detail, and is shown to have a large impact on the performance of the NichePSO algorithm. Possible causes and solutions are also discussed.

3.1 Analysis on Maintaining Niches

A problem with the NichePSO algorithm that was observed during preliminary testing was that in many instances, only one subswarm exists by the end of the run. Further inspection reveals that many different subswarms are created throughout a run, but they all end up merging together into one large subswarm after several iterations. This behaviour is unwanted as each subswarm should be tracking a separate optimum, and merging two subswarms within different optima will cause one to no longer be tracked.

Table 3.1 shows preliminary performance results of the original NichePSO algorithm on seven benchmark functions. Each run contained 100 iterations using 50 particles, with parameters $w = 0.7$ and $c_1 = c_2 = 1.2$. It can be seen that, for almost every function, the NichePSO algorithm was only able to track a single optimum for each run.

3.2 Analyzing the Problem Further

Two independent runs were selected for analysis, and were analyzed in-depth to try and pinpoint the cause of the merging subswarm problem. The following information was collected for each run:

Function	Dimensions	Global Optima	Budget	Optima Found
Five Uneven Peak Trap	1	2	50K	1.0333
Himmelblau	2	4	50K	1
Shubert	2	18	200K	1
Shubert	3	81	400K	1
Composite function CF3	2	6	200K	1
Composite function CF3	3	6	400K	1
Composite function CF3	5	6	400K	1

Table 3.1: The average number of optima located for the NichePSO algorithm after 100 iterations.

- **Largest radius over time**: This shows changes in the largest radius, as well as patterns where the largest radius quickly grows or shrinks.
- **Number of subswarms over time**: This helps to determine whether several subswarms are being created and merged, or if only a small number of subswarms are created during a run.
- **Largest subswarm over time**: This tracks the subswarm with the largest number of particles, to see if a single subswarm is absorbing all the other particles or if the largest subswarm is staying relatively small.
- **Radius of newly created subswarms**: This checks the initial radii of created subswarms, to see whether subswarms with large radii are being created and potentially causing the issue.

The information discussed was collected and plotted for each run in Figures 3.1 to 3.8. Figures 3.1 to 3.4 were taken from a run done on the Shubert 3D function, and Figures 3.5 to 3.8 were taken from a run done on the CF3 function. The number of particles was reduced from 100 to 50 for each run to help the data to be more readable and manageable.

It can be seen in Figure 3.2 that many subswarms are created throughout the run, which means that the results are not an issue of only one subswarm being created. At iteration 35 the graph peaks at 18 subswarms, and after this the number of subswarms quickly drop down to two subswarms, which is a single subswarm and the main swarm. This pattern shows that many subswarms are being created and are merging together into a single subswarm, confirming that the problem does exist. Further evidence can be seen in Figure 3.3, where at iteration 35 a sharp increase is seen in the size

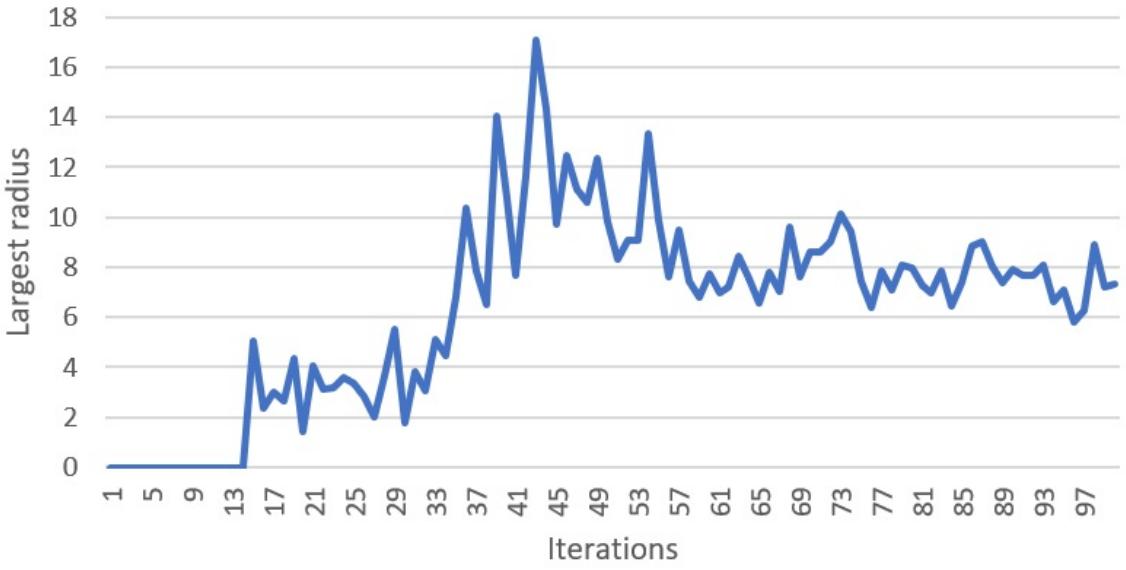

Figure 3.1: The size of the largest radius over time (Shubert 3D, 50 particles).

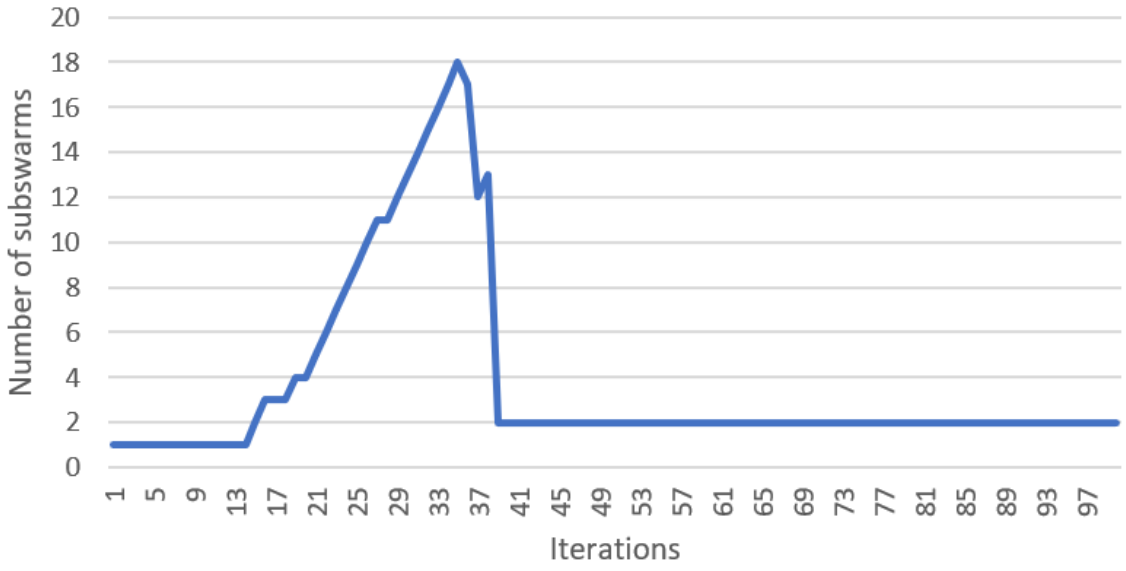

Figure 3.2: The total number of subswarms over time (Shubert 3D, 50 particles).

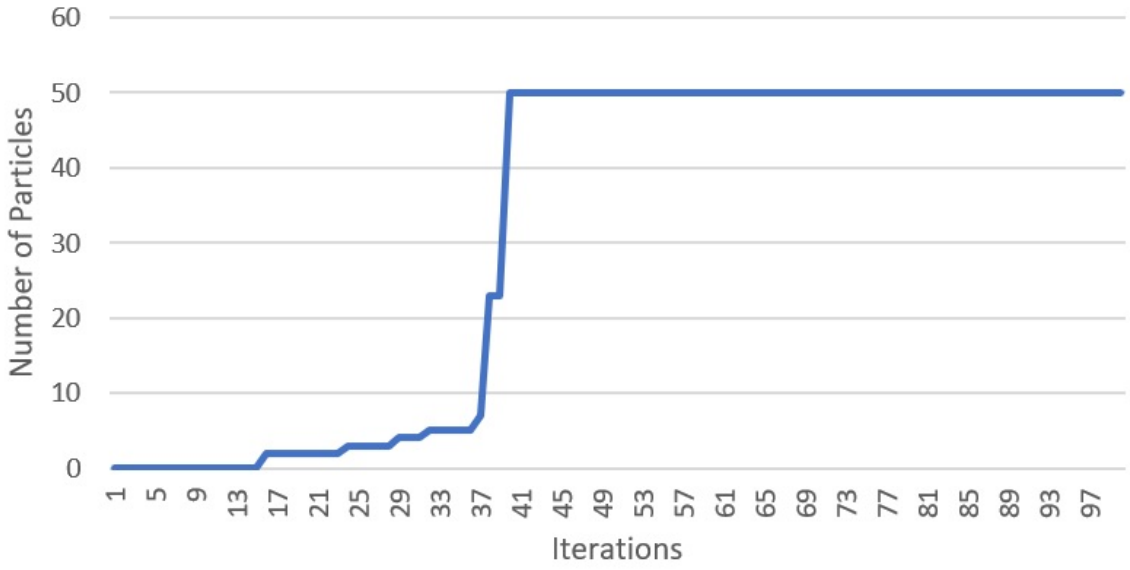

Figure 3.3: The number of particles within the largest subswarm over time (Shubert 3D, 50 particles).

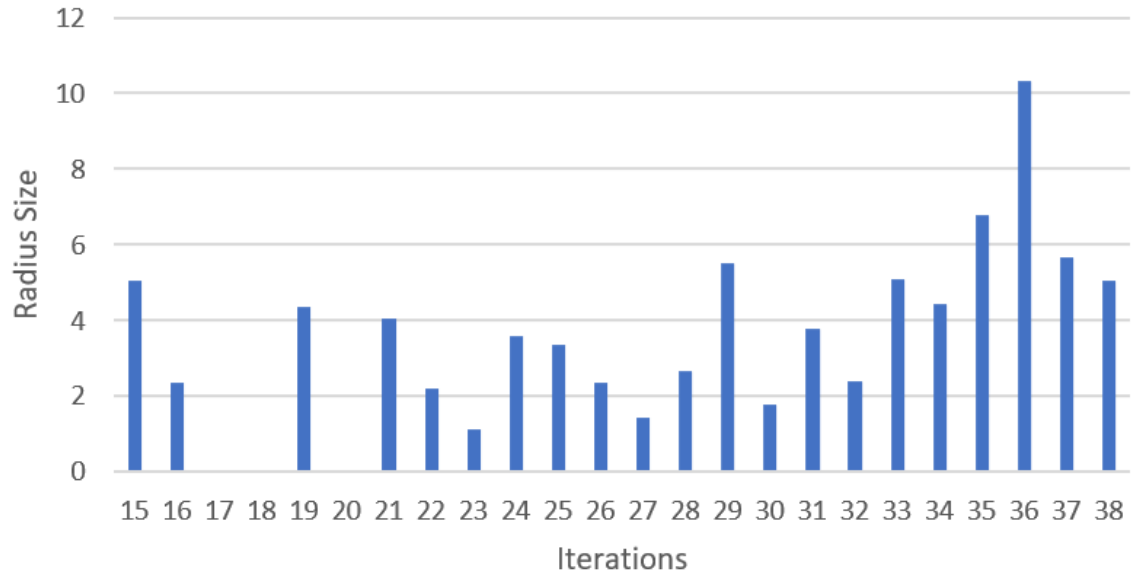

Figure 3.4: The radius size of newly created subswarms throughout a run (Shubert 3D, 50 particles).

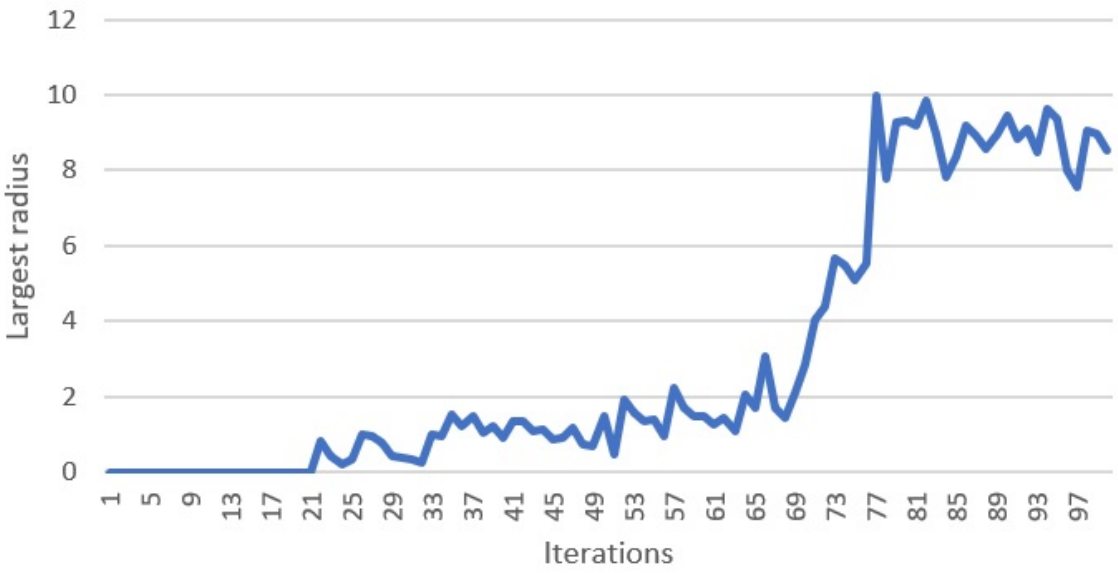

Figure 3.5: The size of the largest radius over time (CF3 3D, 200 particles).

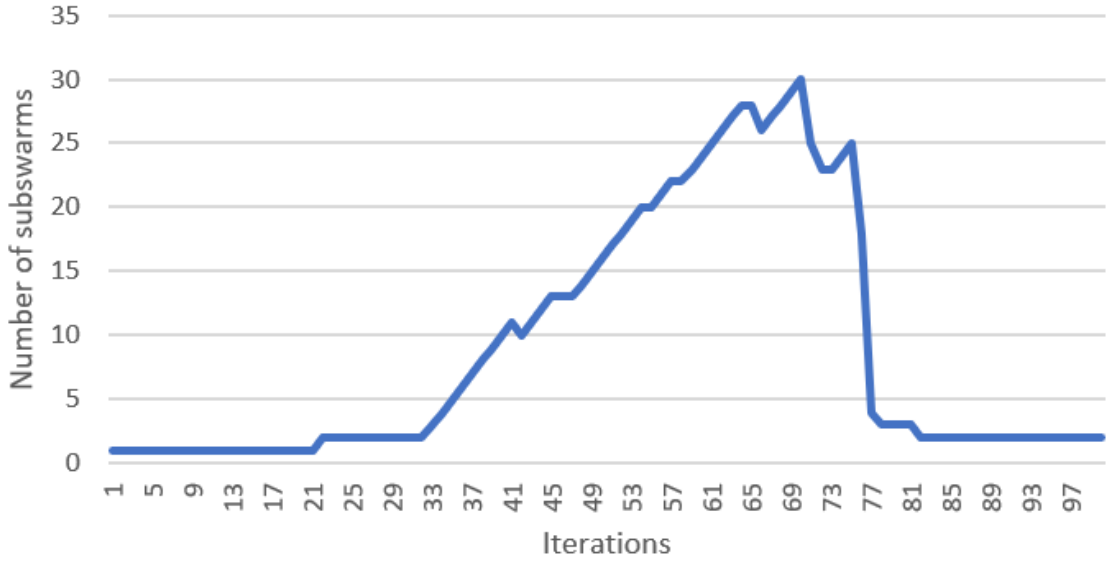

Figure 3.6: The total number of subswarms over time (CF3 3D, 200 particles).

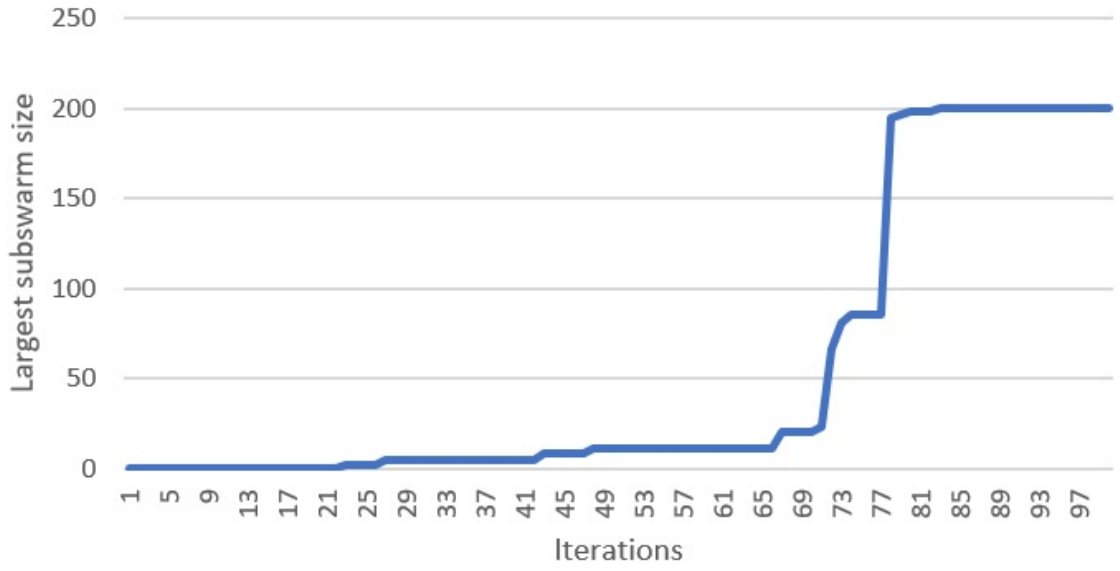

Figure 3.7: The number of particles within the largest subswarm over time (CF3 3D, 200 particles).

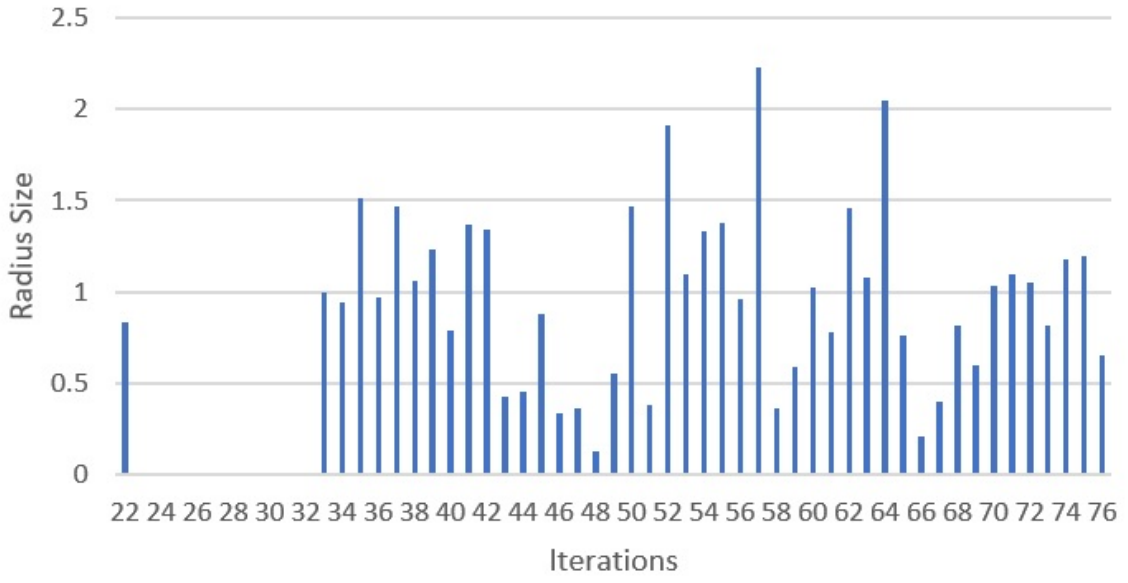

Figure 3.8: The radius size of newly created subswarms throughout a run (CF3 3D, 200 particles).

of the largest subswarm. This shows that as the number of subswarms shrink, the remaining subswarms grow large as expected as they absorb other subswarms. Figure 3.1 shows a similar trend in the maximum radius size, though its pattern is not as clear cut as the other plots.

The patterns seen in Figures 3.2 and 3.3 for the Shubert function are also visible in Figures 3.6 and 3.7 for the CF3 function. While the patterns are not as prominent as the previous Figures, a similar decrease can be seen in Figure 3.6 beginning in iteration 70. At the same time, both the largest subswarm size and largest radius increase in Figures 3.5 and 3.7.

It is clear from these observations that the merging subswarm problem does exist, and is a major issue for the NichePSO algorithm. While there is no guarantee that each subswarm has found an optimum when they merge, an effect of all subswarms merging is the inability to explore new areas of the search space and detect more than a single optimum.

One possible cause for the subswarms merging is that subswarms are being created with large radii around the same time that the merging occurs. When a particle in the main swarm converges, a new subswarm is created that consists of two particles: the particle that converged and its closest neighbour within the main swarm. However, the closest neighbour within the main swarm may not actually be close to the converged particle. Such a situation could happen if there were only two particles remaining in the main swarm, as they both could be far away but still considered the closest neighbours. If this happens, the resulting subswarm created would have a large radius that would intersect with the radii of several other subswarms, thus merging them all into a single subswarm. An illustration of this is shown in Figure 3.9. If one of the orange particles was to converge in Figure 3.9, then the subswarm created would consist of the converged particle and the orange particle on the other side of the search space. The subswarm created would then have a radius that completely covers the existing blue subswarm in between, causing an unwanted intersection.

To determine whether this is the cause of the problem, the size of every subswarm created each iteration was recorded (no more than one subswarm is created per iteration). This data is shown in Figure 3.4 for the Shubert function, and in Figure 3.8 for the CF3 function. Should the hypothesis be true, it is expected that a subswarm with a large radius will be created around the time all the subswarms merge.

Figure 3.4 shows a subswarm created on iteration 36 with a large radius, created just before all the other subswarm start to merge. In order to provide further supporting evidence, the exact subswarm created was analyzed for this run to see if this

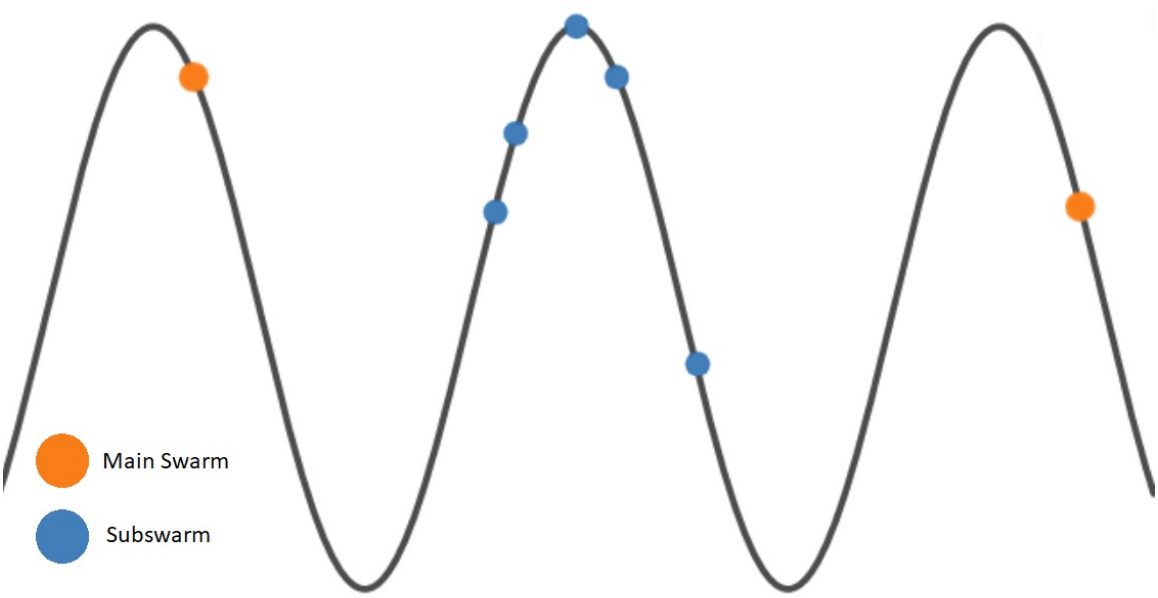

Figure 3.9: If one of the main swarm particles converged, the resulting subswarm would encompass the existing subswarm.

created subswarm absorbed many other subswarms the iteration after it was created. However, when looking at the subswarm in question in greater detail it was found that while it was created with a large radius, in the next iteration it quickly shrunk in size as the two particles moved towards each other. The subswarm in question was created on iteration 36 with a radius of 10.331, and on iteration 37 the radius had shrunk to a size of 3.112. Due to the order of the methods in the algorithm, this takes place before the subswarm is able to absorb any other particles or subswarms.

Figure 3.8 also shows evidence that created subswarms are not the cause of the merging subswarm problem. It can be seen that the size of each created subswarm around iteration 70 is relatively small. A larger subswarm can be seen at iteration 64, but the same behaviour as seen previously emerges where the subswarm shrinks the next iteration before absorbing any particles. The subswarm created on iteration 64 had a radius of 2.046, and on iteration 65 shrunk to a size of 1.703. This provides further supporting evidence that newly created subswarms are not the cause of the rapid merging of the subswarms. Even when subswarms are created with particles far apart, they quickly converge together in the next iteration before absorbing or merging with any other subswarms. This suggests that instead the issue lies within the merging procedure used for the NichePSO algorithm.

3.3 Alternative Merging Strategies

There are two possible areas to modify when considering an alternative merging strategy; How the radii for each subswarm is calculated, and the behaviour for what happens when two subswarms intersect. Several alternative merging approaches were proposed in [11], and were used as a starting point for the experiments.

3.3.1 No Merge Approach

A trivial solution to the merging subswarm problem is to completely remove the NichePSO's ability to merge subswarms. All subswarms cannot merge if there is no merging strategy used. There are some drawbacks to this approach. The NichePSO algorithm loses its ability to prevent multiple subswarms from searching the same area of the problem space. As well, without any subswarms merging, the majority of subswarms will remain relatively small, weakening their ability to perform local searches of the immediate neighbourhood. Despite these drawbacks, the approach is still used to analyze its effect on the merging subswarm problem.

3.3.2 Direction Merge

The direction merge approach was proposed in [11] as a stricter condition for subswarms to merge. Instead of only merging subswarms that intersect, direction merge proposes that the subswarms must intersect and be travelling in the same direction. The idea is that two subswarms can intersect but be travelling towards different niches. Two subswarms intersecting and moving in the same direction are likely to converge to the same optimum, and thus are merged together. The direction of subswarms are compared using the dot product of the two subswarm best particles. If the dot product of the velocity of the subswarm best particles is less than zero, the subswarms are moving towards the same optimum and merge. If the dot product is greater than zero, they are moving in different directions and are not merged.

3.3.3 Diversity Merge

A merging approach was proposed in [11] where subswarms are only merged if the diversity of both swarms is sufficiently low. Within a subswarm, if one particle trails off from the rest of the swarm, then this one particle will greatly increase the radius. This was calculated by computing the average distance each particle is away from

the swarm best, and only merging if this average was smaller than some diversity threshold ϵ.

An undesirable effect of this approach is that it introduces a new parameter to the algorithm. In order to remove the need for a new parameter, a modified diversity merge approach is proposed that factors in the diversity of a subswarm. Instead of calculating the radius by determining the particle in the subswarm that is furthest away from the swarm best, the radius for a subswarm will be equal to the median distance of all the particles in the subswarm. Using the median distance requires no additional parameters, and is much more robust towards particles travelling away from the subswarm. Should one particle travel far away from the rest of the swarm, the radius will not increase as the majority of particles will still be located around the swarm best.

3.3.4 Scatter Merge

The scatter merge appraoch was proposed in [11] as a way to improve the NichePSO's exploration ability. Instead of combining all particles into a single swarm on intersection, the approach is to take the particles of one swarm and reinitialize them back into the main swarm. The subswarm that is reinitialized out of the two is the subswarm with the smaller best fitness. All particles within that subswarm are relocated randomly throughout the search space.

3.3.5 Modified Scatter Merge

During testing an observation was made regarding the scatter merge approach described above. It was observed that runs would often end with a large number of subswarms that contain two particles, the minimum a subswarm can contain. Having many small subswarms can impact the algorithms ability to perform local searches once a subswarm is created, as these subswarms often converge prematurely to suboptimal values. To address this issue, a modified scatter merge approach is proposed. When two subswarms intersect, instead of all particles from one swarm being reinitialized in the main swarm, a single particle from that swarm is added to the other. The remaining particles in the subswarm are reinitialized back into the main swarm. This allows subswarms that have strong personal bests to grow and improve exploitation, while still improving the exploration ability of the original NichePSO algorithm.

Function	Dimensions	Global Optima	Budget	No Merge	Direction	Diversity	Scatter	Modified Scatter	Diversity and Mod Scatter
Five Uneven Peak Trap	1	2	50K	A: 2 S: 0	A: 2 S: 0	A: 2 S: 0	A: 2 S: 0	A: 2 S: 0	A: 2 S: 0
Himmelblau	2	4	50K	A: 4 S: 0	A: 3.4667 S: 1.1366	A: 4 S: 0	A: 4 S: 0	A: 3.9 S: 0.3051	A: 4 S: 0
Shubert	2	18	200K	A: 15.7667 S: 1.2229	A: 4.8 S: 4.0548	A: 15.8 S: 1.3235	A: 16.6 S: 1.9931	A: 6.8333 S: 2.9837	A: 17.7667 S: 0.504
Shubert	3	81	400K	A: 30.6667 S: 4.1384	A: 3.0333 S: 5.3594	A: 35.6667 S: 2.2024	A: 28.1333 S: 5.9291	A: 9.7333 S: 9.8609	A: 38.8667 S: 2.5829
Composite function CF3	2	6	200K	A: 4.7 S: 0.7944	A: 2.5 S: 1.6557	A: 5.1 S: 0.6618	A: 5 S: 0.6948	A: 4.2333 S: 1.2229	A: 5.4667 S: 0.6288
Composite function CF3	3	6	400K	A: 4 S: 0.2626	A: 2.6 S: 1.4527	A: 4 S: 0	A: 4 S: 0	A: 4.0333 S: 0.1826	A: 4 S: 0
Composite function CF3	5	6	400K	A: 3.8667 S: 0.4342	A: 3.2667 S: 1.1725	A: 4 S: 0	A: 3.9 S: 0.4026	A: 4 S: 0	A: 4 S: 0

Table 3.2: Average optima found (A) and standard deviation (S) for the proposed merging strategies.

3.4 Merging Strategies Experiment Results

Each merging strategy discussed was tested for performance using the same benchmark functions as Table 3.1. To recap, the original NichePSO merging strategy was only able to detect on average a single optimum for each tested function. Experimental results are shown in Table 3.2. It should be noted that the goal for these tests are to compare relative performance given the provided parameter set. As such, all parameters will remain static and will not be tuned for each individual function or merging strategy. Each strategy was tested over 30 runs, and used the following parameters:

- Number of particles: 100
- Accuracy threshold: 0.01
- Inertia: 0.7
- C_1 and C_2: 1.2

3.4.1 Strategy Analysis

The motivation behind studying alternative merging strategies is to overcome the subswarm merging problem discussed previously. If a strategy has overcome the issue, then the results will show on average at least two optima being detected for each function. As seen in Table 3.2, each discussed strategy is able to properly track multiple optima on average for each function tested.

No merge

Despite the drawbacks of removing subswarm merging altogether, the no merge strategy shows relatively strong results compared to the other strategies. This provides more evidence that the subswarm merging problem was being caused by the merging strategy, as removing merging from the algorithm shows to no longer suffer from the problem.

Another benefit of the no merge strategy is that there is room to improve further. As discussed previously, removing subswarm merging altogether has some negative effects such as less subswarm diversity and reduced exploitation in each subswarm. Later in the thesis these drawbacks will be addressed, and it is expected that performance will improve even more.

Direction merge

Out of all the alternative merging strategies examined, the direction merge on average performed the worst. The average number of optima found was generally lower than the other strategies, and the standard deviation was also much higher, showing less consistent results. It was also observed that there were several runs using direction merge that still resulted in a single optimum being detected, showing that while it is an improvement over the traditional merging strategy, it still can suffer from the subswarm merging problem.

It is hypothesized that the reason for the weaker performance from the direction merge strategy has to do with the use of the GCPSO velocity update formulas (see Equations 2.3 and 2.4). Using GCPSO, the best particles for each subswarm are updated differently than the rest of the particles. The subswarm best formula gives the particle a random direction each update, which means that the global best particle could be travelling a different direction each iteration irrespective of the direction the remaining particles are travelling. This means that at any given iteration, there is no way of knowing which direction the rest of the subswarm is moving just by checking the direction of the subswarm best particle.

This strategy could be improved in two ways. The first improvement would be if the GCPSO update formulas are replaced with update formulas that do not have the subswarm best particles traveling in random directions. The second improvement would be if an alternative way of calculating subswarm direction was used that better estimates the direction the swarm is travelling. One possibility could be to track the change in subswarm best position as opposed to the velocity at a given point in time.

In other words, when a global best particle moves to a position that is more fit, the direction it moved in that instance is recorded and used to compare whether two subswarms are travelling in the same direction. Further work involving this strategy will not be included in this thesis.

Diversity merge

The diversity merge strategy takes a different approach than the other strategies discussed in that it does not modify the behaviour of what happens when two subswarms intersect, but instead only modifies the radius size of each subswarm. Despite this difference, this strategy shows relatively strong results across all functions used.

The strength of this approach is that it acts as a natural filter for outlier particles. In the traditional NichePSO algorithm, if a single particle within a subswarm is to travel away from the rest of the swarm, the radius would grow to encapsulate this particle. With the diversity merge approach, a swarms radius is unaffected by a single particle travelling far away from the rest of the swarm. Another strength is that this approach can be easily combined with other strategies without causing any conflicts, as shown by combining it with the modified scatter merge approach.

Scatter and modified scatter merge

The scatter merge approach shows slightly weaker results than the other strategies, but still shows to overcome the merging subswarm problem. As mentioned previously, it was observed that the scatter merge approach ended most runs with many small subswarms that were unable to grow during the run. Looking at the modified scatter merge results, alone it fails to obtain the same level of results as the scatter merge approach. However, when combined with the diversity merge strategy, the two approaches work very well together and return the highest results for most functions used.

Moving forward, tests done on the NichePSO algorithm will be done so with the diversity merge modification. This approach is selected because out of all the strategies discussed it modifies the behaviour of the original algorithm the least, while also showing to overcome the subswarm merging problem. Tests involving this approach will therefore not be affected by the subswarm merging problem, allowing the results to more accurately evaluate the performance of the algorithm.

Chapter 4

Modifications and Analysis

This chapter explores several of the procedures and parameters that the NichePSO algorithm utilizes for strengths and weaknesses. Several potential modifications to the algorithm are proposed and analyzed with the goal of improving overall performance and efficiency.

4.1 Impact of Parameter Values

This section looks at several parameters that the NichePSO algorithm utilizes throughout a run. Parameter values are adjusted and analyzed for their impact on a run, and the algorithm's sensitivity to the values is explored.

4.1.1 Analysis of the Inertia Parameter

This section explores how the inertia value used affects the performance of the NichePSO algorithm. Motivation to explore this parameter arose when an observation was made that changing how quickly the inertia decreases throughout a run can greatly change the outcome of a run.

Each iteration, the inertia value decreases by an amount of

$$w(t+1) = w(t) - ((w(0) - 0.2)/Max\_Iterations) \tag{4.1}$$

Whatever the beginning value set is, over the course of the run the inertia is decreased linearly each iteration until it settles on a value of 0.2. This value was chosen as it is the smallest value that would maintain parameter stability between the inertia value, the cognitive term and the social term. Stability of PSO parameters was studied in [14], where it was shown that algorithm stability can be obtained if the

Function	Dimensions	Global Optima	Budget	0.3	0.5	0.7	0.9
Five Uneven Peak Trap	1	2	50K	2	2	2	2
Himmelblau	2	4	50K	4	4	4	3.3667
Shubert	2	18	200K	16.6	16.3667	15.8	8.7667
Shubert	3	81	400K	34.1	35.4667	35.6667	21.6333
Composite function CF3	2	6	200K	4.9333	4.8333	5.1	4.8667
Composite function CF3	3	6	400K	4	4	4	4
Composite function CF3	5	6	400K	4	4	4	4

Table 4.1: The average number of optima located for each inertia value tested with a linear decrease each iteration.

Function	Dimensions	Global Optima	Budget	0.3	0.5	0.7	0.9
Five Uneven Peak Trap	1	2	50K	2	2	2	2
Himmelblau	2	4	50K	4	4	4	2.9667
Shubert	2	18	200K	16.6667	16.6333	15.8667	4.9
Shubert	3	81	400K	35.3333	35	34.1667	20.1333
Composite function CF3	2	6	200K	4.6667	4.7333	5.1667	3.9333
Composite function CF3	3	6	400K	4	4	4	3.8333
Composite function CF3	5	6	400K	4	4	3.9333	3.9667

Table 4.2: The average number of optima located for each inertia value tested with no decrease.

parameters w, C_1 and C_2 satisfy the equation $0 \leq (C_1 + C_2) \leq 2(1 + w)$. If we use a constant $C_1 = C_2 = 1.2$ value for both the cognitive and social terms, than by substituting them into the above equation and solving for w, stability can be achieved with $w \geq 0.2$.

Experimental results are shown in Tables 4.1 and 4.2. Each run recorded the total number of unique global optima tracked, and every recorded value was averaged over 30 runs. The inertia values in Table 4.1 all began at the values shown and decreased linearly to a value of 0.2 throughout each run. Inertia values in Table 4.2 remained the same throughout each run. To mitigate the effect of the merging subswarm problem, all runs were done using the diversity merge modification. Parameters for each run as as follows:

- Number of particles: 100
- Accuracy Threshold: 0.01
- C_1 and C_2: 1.2

A few observations can be made from the data reported. The most notable information is that changing the inertia value from the originally proposed value of 0.7 did not improve performance for almost every run performed. One clear observation is that increasing the inertia to a value of 0.9 showed worse performance across most functions tested. Decreasing the inertia had very little effect, as no changes in performance for 0.3 or 0.5 were statistically significant.

Similar observations can be made when analyzing the effect that removing the linear decrease has on performance. While the averages were close to the original, the fact that none showed any improvement is evidence enough that an inertia of 0.7 is sufficient when paired with a cognitive and social term of 1.2. The use of a linear decrease with the inertia parameter is favourable, but good results can still be found without one. If one removes the linear decrease, then the results suggest that a smaller inertia value is preferred in this scenario.

4.1.2 Particle Absorption Analysis

The next set of tests explore the impact that particle absorption has on the NichePSO algorithm. Particle absorption applies when particles from the main swarm intersect with the radius of a subswarm. The subswarm absorbs the particle, and that particle becomes a part of the subswarm it intersected with. To evaluate the impact of particle absorption, several merging strategies were run without the use of absorption to compare the performance of the runs to the results with the use of absorption. Experiments were averaged over 30 runs, and were done on the Shubert 3D function and the composite function CF3 benchmark function. Results can be found in Table 4.3.

Along with performance results, the number of times that the absorption process takes place during a run is recorded. In order to see how often particles join a subswarm through absorption as opposed to creation, both procedures were counted and recorded as ratios. There are only two ways a particle can leave the main swarm: they can be absorbed by a subswarm, or they can converge and form a new subswarm. Strategies where absorption plays a larger role will see a higher absorption count compared to the creation count, and strategies where absorption plays a smaller role will thus see a smaller ratio. For each run, a ratio of absorption : creation counts was recorded, and these ratios are averaged over the 30 runs. Results are shown in Table 4.4.

As can be seen in Table 4.3, the inclusion of absorption did not have a signifi-

	With Absorption		Without Absorption	
	Shubert	CF3	Shubert	CF3
Vanilla	1	1.1	1	1
No Merge	30.6667	4	35.8333	4
Diversity	35.6667	4	35.2	4
Scatter	28.1333	4	30.4667	4
Modified Scatter	9.7333	4.0333	13.9667	4
Diversity and Modified Scatter	38.8667	4	37.8667	4

Table 4.3: Average number of optima tracked for the different merging strategies with and without the use of particle absorption.

Strategy	Shubert	CF3
Vanilla	27 : 73	40 : 59
No Merge	19 : 80	33 : 66
Diversity	0 : 99	1 : 99
Scatter	67 : 1342	134 : 1257
Modified Scatter	331 : 546	129 : 439
Diversity and Modified Scatter	0 : 130	1 : 208

Table 4.4: Ratios of the number of particles absorbed by subswarms to the number of particles creating new subswarms.

cant impact on the overall performance of any strategy. Removing absorption does show a slightly higher number of optima found on average, but few experiments are statistically significant.

Looking at Table 4.4 it can be seen that aside from the vanilla NichePSO merging strategy, all strategies tested have a much higher creation rate than absorption rate on average. What this suggests is that absorption is used very little throughout a run, and the majority of particles leave the main swarm by converging and creating new subswarms of their own.

Taking the results on these experiments into consideration, it can be concluded that removing particle absorption is beneficial to the NichePSO algorithm. While the increase in performance is not a significant increase, the alternative conclusion is that absorption impacts the algorithm very little. If this conclusion is true, then it is still beneficial to remove absorption to reduce the computational cost of the algorithm. This conclusion is further supported by the results of Table 4.4, which show that for all merging strategies tested, absorption occurs very little during a run. All experiments performed following this conclusion will be done without the use of

particle absorption, unless otherwise stated.

4.1.3 Optima Located Over Time

This subsection will explore the relationship between the number of iterations a run has progressed and the number of unique global optima that has been found. It is clear that allowing a run to progress for more iterations will result in an increased performance, as each subswarm and particle is allowed more time and resources to explore the search space. The purpose of this experiment is to analyze the correlation between iterations and optima found. As more and more optima are located, it becomes more difficult to find the remaining optima and thus more resources are required for a smaller increase in performance.

Experiments were performed using the traditional NichePSO algorithm with the diversity merge modification. The diversity merge is used so that the results are not muddled by the subswarm merging problem. Each result was averaged over 30 runs, and for each run the number of unique global optima found is recorded every 100 iterations. Results can be found in Table 4.5. Parameters for all experiments are as follows:

- Total Iterations: 2000
- Number of Particles: 100
- Inertia: 0.7
- C_1 and C_2: 1.2
- Accuracy Threshold: 0.01

As expected, at the start of a run the number of optima found starts off small and quickly increases. As the run progresses, the speed of increase slows down as the algorithm has less and less optima to locate. It can be observed that for each function, the number of dimensions plays a large role in the number of iterations required for the data to begin to plateau.

To better highlight this observation, the data shown in Table 4.5 was plotted on a stacked area chart shown in Figure 4.1. When interpreting this plot, the magnitude of the y-axis does not matter. What is important is the slope of the lines, as the y-axis plots the relative number of optima that has been located at a given iteration with respect to the total number of optima located by the end of the run.

Function	Dimensions	Global Optima	100	200	300	400	500	600	700	800	900	1000
Five Uneven Peak Trap	1	2	1.7	2	2	2	2	2	2	2	2	2
Himmelblau	2	4	4	4	4	4	4	4	4	4	4	4
Shubert	2	18	3.7333	12.1	13.733	14.767	15.0667	15.733	15.733	15.833	15.833	15.8333
Shubert	3	81	0.0333	1.2	2.7	4.2667	6.0667	8.6	10.9	14.1	16.9	20.2667
Composite Function CF3	2	6	0.7667	3.8333	4.2333	4.5333	4.6333	4.7	4.7333	4.7667	4.7667	4.8
Composite Function CF3	3	6	0.0333	3.5667	3.9	3.9	3.9667	3.9667	4	3.9667	4	4
Composite Function CF3	5	6	0	0.3	0.7	0.8333	1	1.1667	1.2333	1.4667	1.9667	2.3

Function	Dimensions	Global Optima	1100	1200	1300	1400	1500	160	1700	1800	1900	2000
Five Uneven Peak Trap	1	2	2	2	2	2	2	2	2	2	2	2
Himmelblau	2	4	4	4	4	4	4	4	4	4	4	4
Shubert	2	18	15.8333	15.8333	15.8333	15.8333	15.8333	15.8333	15.8333	15.8333	15.8333	15.8333
Shubert	3	81	23.2	26.2	28.2333	29.6333	30.7667	31.9	32.7333	33.5333	34.0667	34.3
Composite Function CF3	2	6	4.8	4.8	4.8	4.8	4.8	4.8333	4.8333	4.8333	4.8333	4.8667
Composite Function CF3	3	6	4	3.9667	4	4	4	4	4	4	4	4
Composite Function CF3	5	6	2.7	3.0333	3.2667	3.5333	3.8333	3.9333	3.9667	3.9667	4	4

Table 4.5: Number of global optima found over a period of 2000 iterations.

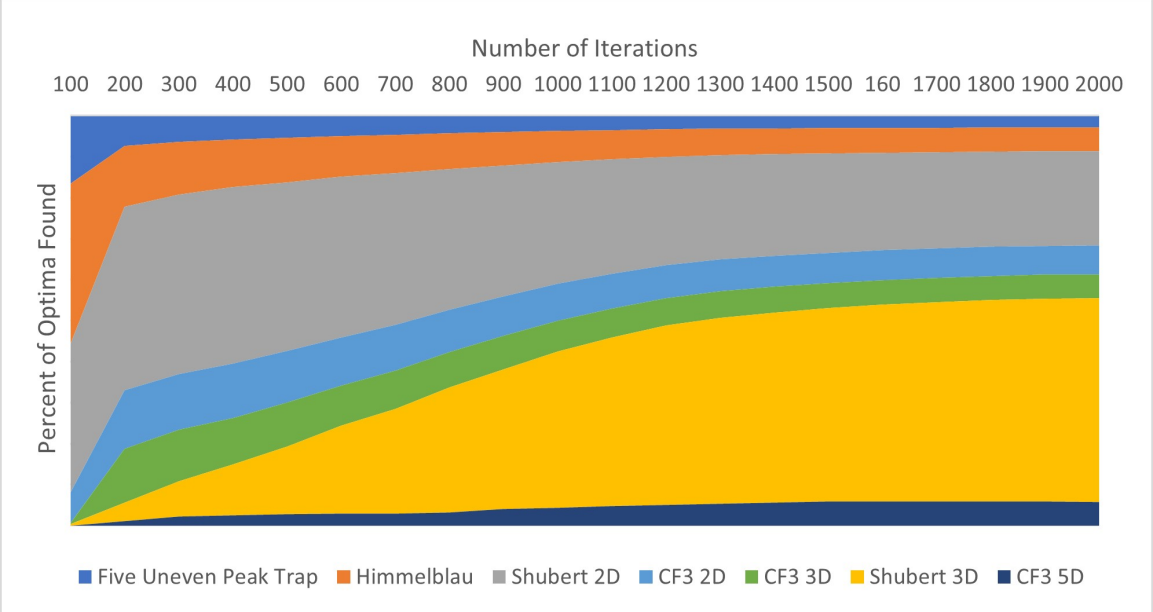

Figure 4.1: Comparison of how many optima are tracked as a run progresses for several benchmark functions.

What can be observed is that the functions that have two dimensions all show a very similar line throughout the 2000 iterations. Likewise, the functions used that have three dimensions also show similar trends as the runs progress. While it is clear that increasing the number of iterations will lead to an increase in performance, this plot shows relatively how many iterations are needed for functions of various dimensions. This information can be used to determine a strong value for the max number of iterations that balances out performance and computational cost.

4.1.4 Analysis of the Number of Particles

The final parameter that will be studied is the number of particles that the main swarm should be initialized with at the start of a run. It is clear that increasing the number of particles will lead to more optima being located and tracked. This however is not a feasible course of action, as more particles being used means an increase in computational cost and a decrease in algorithm efficiency. As well, as the number of optima located grow larger and larger, it will require exponentially more particles to continue to increase performance at the same rate.

The goal of the first experiment performed is to analyze the correlation between the number of particles and number of optima tracked. Runs were averaged over 30 runs on several benchmark functions with a varying number of optima and dimensions. All experiments used the diversity merge modification to mitigate the impact of the merging subswarm problem. Results are shown in Table 4.6. Parameters used for each run are as follows:

- Number of iterations: 1000
- Inertia: 0.7
- C_1 and C_2: 1.2
- Accuracy threshold: 0.01

As expected, as the number of particles increase the number of optima tracked also increases. As well, the results provide evidence that as the number of optima tracked approaches the maximum number that a function contains, more and more resources are needed to maintain the improvement. When looking at the Shubert 3D function, the difference between using 100 particles and 250 particles resulted in an increase in roughly 17 global optima tracked. The difference between using 250 particles and 500 particles is also roughly 17 global optima tracked. The increase

in performance remains the same for these three data points, but the increase in particles from 100 to 250 is only 150 particles, whereas the increase from 250 to 500 is a 250 particle increase. If total resources is not a factor to consider, then it is clear that more particles results is better performance. However this is rarely the case, and one must find a balance between resources allocated and performance.

The goal of the next experiment is to try and find an optimal balance between cost and performance. Instead of using a maximum number of iterations for each run, a maximum number of function evaluations will be allowed. The same functions and parameters used in the previous experiment will be used again. The goal of this second experiment is to try and find a balance between number of particles and computational cost that would maximize performance. Results are shown in Table 4.7.

A visualization of Table 4.7 is shown in Figure 4.2. When interpreting this graph, the magnitude of the y-axis is not important between the functions. Each line is relative to the maximum number of optima found per function, and roughly shows how the performance changes as the number of particles increase.

Figure 4.2 shows a similar trend among each function used, however the impact is much more noticeable with the Shubert 3D function. This makes sense as this function contains the highest number of optima out of the functions used, and the gaps between each number of particles used is much greater. What the results show is that the optimal number of particles to use that would balance out performance and computational cost is around 250. If less particles are used, then less optima are able to be tracked and less of the search space can be explored. As more particles are added, the amount of resources allocated for each particle decreases, which leads to subswarms being unable to effectively search their local neighbourhoods and find the global peaks. This correlation is less noticeable for functions with a smaller number of global optima, and one can use less particles then to reduce the computational

Function	Dimensions	Global Optima	20	50	100	250	500	1000
Five Uneven Peak Trap	1	2	1.6333	2	2	2	2	2
Himmelblau	2	4	4	4	4	4	4	4
Shubert	2	18	8.1	14.0333	16.6333	17.5333	17.8333	17.6667
Shubert	3	81	5.2	15.9	26.6	43.0333	60.5667	69.3
Composite Function CF3	2	6	3.3	4.1333	4.6	5.6667	5.9333	5.9333
Composite Function CF3	3	6	3.5667	4	4	4	4	4
Composite Function CF3	5	6	1.9	3.0667	3.8	3.9667	4	4

Table 4.6: The average number of optima located for various number of particles initialized over 1000 iterations.

Function	Dimensions	Global Optima	Budget	20	50	100	250	500	1000
Five Uneven Peak Trap	1	2	50K	1.6	2	2	2	2	1.1
Himmelblau	2	4	50K	5	4	3.9667	4	4	3.7667
Shubert	2	18	200K	7.9	13.3	15.6667	14.8333	13.4667	11.2
Shubert	3	81	400K	8.3667	20.7	35.3	55.3667	49.9667	13.6667
Composite Function CF3	2	6	200K	3.5333	4.5	4.8667	5.4	5.7333	4.4
Composite Function CF3	3	6	400K	3.7333	4	4	4	4	3.7333
Composite Function CF3	5	6	400K	3.3	4	4	4	3.9667	1.2667

Table 4.7: The average number of optima located using diversity merge for various number of particles initialized using function evaluations as the stopping criteria.

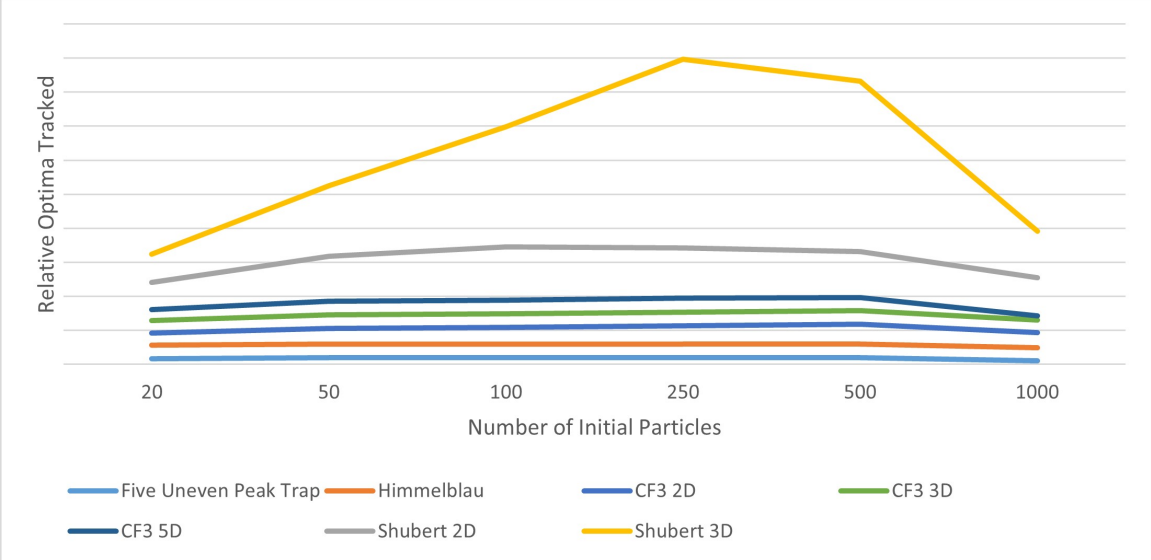

Figure 4.2: Trend lines showing relative number of optima tracked with respect to the number of particles used.

cost.

4.2 Subswarm Modifications

This section proposes several possible modifications to how subswarms are utilized during a run. This includes modifying how they are created, how they interact with each other, and how their radii are calculated. Experiments are performed that analyze each modification against the traditional NichePSO algorithm using the diversity merge modification for impact on performance.

4.2.1 Alternative Subswarm Creation Method

An observation was made when exploring possible causes of the merging subswarm problem in Chapter 3. It was hypothesized that the way subswarms are created could be causing undesirable effects within the algorithm. To recap, when a particle within the main swarm converges it creates a new subswarm by combining with the closest neighbouring particle within the main swarm. This behaviour was shown to not be the cause of the merging subswarm problem, but the issues it creates are still prevalent and should be addressed.

The first major issue is that when a particle converges and forms a subswarm, there is no guarantee that the closest neighbouring particle is actually nearby. If the closest neighbour happens to be far away in the search space, then the resulting subswarm will have a very large radius and poorly define the area it is currently searching (see Figure 3.9). As well, the particle far away may be tracking a unique optimum of its own, but by merging it into the created subswarm it then loses this optimum and instead converges to an optimum already located by another particle.

To address these problems that the traditional subswarm creation method poses, a new creation method is proposed that is designed to mitigate these issues. When a particle in the main swarm converges, κ particles are created at that moment and positioned near the converged particle. Particles are positioned at the location of the converged particle, and then each dimension is slightly modified at random to position the new particles nearby. The converged particle and these created particles then form the new subswarm.

The main strength of this approach is that it allows particles in the main swarm to search their own unique areas of the search space without being affected by other particles. As well, since the created particles are positioned near the original con-

verged particle, created subswarms will not have a large radius defined that might intersect with other subswarms searching their own unique optimum.

A downside to this approach is that the maximum number of particles used during a run increases to a maximum of κ multiplied by the number of initial particles in the main swarm. These additional particles will increase the computational cost, and if one is running the algorithm with a maximum number of function evaluations then each created particle lowers the number of function evaluations allotted for the others. Therefore it is important to select a κ value that balances out performance and computational cost.

Experiments were run to study how the alternative creation method compares to the traditional closest neighbour creation method. The parameter κ is also modified to try and determine the optimal value to maximize performance with respect to function computational cost. Each result was averaged over 30 runs, and all runs used the diversity merge modification to mitigate the merging subswarm problem. Results are shown in Table 4.8. Constant parameters used for each run are as follows:

- Number of particles: 100
- Inertia: 0.7
- C_1 and C_2: 1.2
- Accuracy threshold: 0.01

The first observation to be made is that using a smaller number of particles per subswarm results on average in better performance. The most notable increase in performance happens on the Shubert functions that contains a relatively large number of optima. This increase in performance despite the smaller number of particles is most likely attributed to an increase in total function evaluations per particle.

Function	Dimensions	Global Optima	Budget	Closest Neighbour	Alternative (Created Particles)			
					1	2	3	4
Five Uneven Peak Trap	1	2	50K	2	2	2	2	2
Himmelblau	2	4	50K	3.9667	4	4	3.9667	3.8667
Shubert	2	18	200K	15.6667	17.9667	15.6333	12.2	11.2
Shubert	3	81	400K	35.3	52.6	46.8	38.8	34.0333
Composite function CF3	2	6	200K	4.8667	5.1333	5.3667	5.2333	5.3
Composite function CF3	3	6	400K	4	4	4	3.9667	4
Composite function CF3	5	6	400K	4	4	4	4	4

Table 4.8: The average number of optima found for the alternative creation method with varying values for κ.

Creating only a single particle per subswarm allows each particle to have roughly twice as many function evaluations as creating two particles per subswarm. This provides evidence that when selecting parameters for the NichePSO algorithm, one should consider using less particles to allow them more time to converge, as opposed to using many particles with less time to converge.

Another observation that can be made is that using this alternative method to create subswarms on average results in a better performance than the original closest neighbour method. This difference in performance is a lot more clear in the Shubert functions used with many more optima than the other functions. Even though creating new particles for each subswarm causes an increase in the total number of particles used throughout a run, by limiting the number of function evaluations allowed this bias is mitigated. While the results show that using less particles with more function evaluations is preferred, even with this increase in particles, an general increase in performance is observed.

4.2.2 Radius Out of Bounds Approach

When exploring alternative merging strategies earlier in the thesis, it was observed that removing subswarm merging completely can lead to strong results despite the few issues created. Subswarms are merged when they intersect as a response to two subswarms exploring the same area of the search space. By removing subswarm merging, the NichePSO algorithm has no methods for ensuring population diversity among the subswarms. The only thing that would prevent subswarms from all converging to the same location is the initial location of the particles selected when the main swarm is initialized.

To prevent subswarms from searching the same areas of the search space, a modification to the no merge strategy is implemented. Each subswarm has a radius that defines the area that it is currently searching. The modification treats these areas as an out of bounds region for every particle not within the subswarm. When a particle travels out of bounds in a PSO, it is not allowed to update its personal best or the global best position found. Essentially, out of bounds regions are treated as unfavourable positions for the particles to be in, and the particle will naturally travel back within the search space as it converges towards its personal best position and the global best position.

This method of treating a subswarms radius as an out of bound area is referred to in this thesis as the radius out of bounds method (radiusOOB). This technique

Function	Dimensions	Global Optima	Budget	Diversity	No Merge	RadiusOOB	Alt Creation	RadiusOOB with alt creation
Five Uneven Peak Trap	1	2	50K	2	2	2	2	2
Himmelblau	2	4	50K	3.9667	4	4	4	4
Shubert	2	18	200K	15.6667	17.3	17.1333	17.9667	17.9333
Shubert	3	81	400K	35.3	35.8333	35.6333	52.6	53.1667
Composite function CF3	2	6	200K	4.8667	4.8	4.6667	5.1333	4.93333333
Composite function CF3	3	6	400K	4	4	4	4	4
Composite function CF3	5	6	400K	4	4	4	4	4

Table 4.9: Performance results comparing previously viewed methods to the RadiusOOB method.

prevents subswarms from converging to the same location by treating areas of the search space already being searched as unfavourable areas in terms of fitness. When a particle then enters such an area, it will leave naturally as it converges to the better positions already located.

The advantage of this approach is that when two subswarms intersect, there is no immediate event that takes place. The subswarm merging approaches looked at earlier in the thesis all force some event to happen when an intersection occurs, where the action depends on the strategy currently being used. These events are completely artificial, which takes away from the nature inspired algorithm that PSO is. By removing such an event and instead subtly modifying the behaviour of the particles, it allows the particles to behave much more naturally.

Experiments were performed to compare how implementing the radiusOOB method impacted the performance compare to the no merge method previously discussed. The method was also paired with the alternative subswarm creation method previously described to investigate how the two modifications work together. For the alternative creation method, one particle is created for each subswarm that is created. All results were averaged over 30 runs, and for each run the total number of optima located was recorded. Results are shown in Table 4.9. Parameters used for each run are as follows:

- Number of particles: 100
- Inertia: 0.7
- C_1 and C_2: 1.2
- Accuracy threshold: 0.01

It can be seen in Table 4.9 that applying the radiusOOB method shows little change in performance. This lack of impact is most likely attributed to particles and subswarms intersecting very little throughout a run. Since the radius of a subswarm

is calculated using the distance of the particles within the subswarm, as a subswarm converges towards a single point, the radius of the subswarm converges towards 0. The impact that this has is that nearby subswarms may search very close to the subswarm without intersecting, as their radii are relatively small and would not overlap.

To address this problem and try to improve the impact that the radiusOOB method has on the algorithm, experiments were done to modify the subswarm radius formula to prevent them from shrinking to a very small size. Earlier experiments proposed the diversity merge strategy, which limited the growth of a radius with regards to outlier particles. The following experiments aim to limit convergence of a radius so that the area of effect for a subswarm is maximized without becoming too large.

Static Radius

A trivial approach to this problem is to set the radii of all subswarms to the same static size for each iteration. A constant radius would not shrink to a relatively small size as the particles converge, thus decreasing the possibility of multiple subswarms tracking the same niche within the search space. The downside of this method is that it adds a new parameter that requires tuning for each specific function to optimize.

Minimum Sized Radius

Another simple approach is to use a dynamic radius calculation method, but enforce that every radii has a minimum size that must be maintained. Like the static approach, this method is a trivial implementation and would still address the issue of converging radii. This method also allows subswarms with a larger radius to grow in size without constricting it like the static approach would. The downside of this method remains the same, where a new parameter is added that requires tuning for each problem.

Experimental results are shown in Table 4.10. Each result was averaged over 30 runs, and used the radiusOOB method with the alternative subswarm creation method modification. Parameters used for each run are as follows:

- Number of particles: 100
- Particles created per subswarm: 1
- Inertia: 0.7

Function	Dimensions	Global Optima	Budget	RadiusOOB	Static radius of 0.1	Static radius of 0.5	Minimum radius of 0.1	Minimum radius of 0.5
Five Uneven Peak Trap	1	2	50K	2	2	2	2	2
Himmelblau	2	4	50K	4	4	4	4	4
Shubert	2	18	200K	17.9333	17.8	17.4667	17.5333	17.2333
Shubert	3	81	400K	53.1667	50.0333	50.1667	51.2	50.9
Composite function CF3	2	6	200K	4.9333	4.0667	4.1	4.1	4.1667
Composite function CF3	3	6	400K	4	4	4	4	4
Composite function CF3	5	6	400K	4	4	3.9667	4	3.9333

Table 4.10: Performance results comparing different methods for calculating a subswarms radius.

- C_1 and C_2: 1.2
- Accuracy threshold: 0.01

It can be seen in Table 4.10 that for most functions, not using a minimum sized or static radii results in on average higher performance. While the changes in performance are not large, the small decrease in performance provides enough evidence to suggest that these simple approaches are not the correct approach to overcoming the issues with the radiusOOB method.

The decrease in performance for these approaches is likely caused by subswarms that are located in sub-optimal locations. These subswarms would maintain a radius along with the other subswarms, but if they converge to a weak location in the search space then nearby positions that are stronger would be overlapped by the radius. If this is the case, then the method for calculating a subswarms radius would need to both prevent the subswarms radius from converging and also limit the size and impact that weaker subswarms have on stronger subswarms.

Ideal Radius Calculation Formula

The goal of a subswarm's radius is to denote an area of the search space that the subswarm is currently searching. If two radii intersect, an assumption is made that these two subswarms are searching and converging towards the same optimum. In this situation, action is taken to prevent this through the use of a merging strategy. It is ideal that within the bounds of any radius, only a single optimum exists (perhaps yet to be located). Should a radius contain multiple optimum, only a single one would be tracked by the end of the run. Another consideration is that it is undesirable for a radius to be relatively small. If a radius is too small, then it is easy for another subswarm to exist close by and converge towards the same location. One final consideration is that if one treats the radius as off limits to other subswarms, then this would create "false optima" located around the edges of the radius. A subswarm

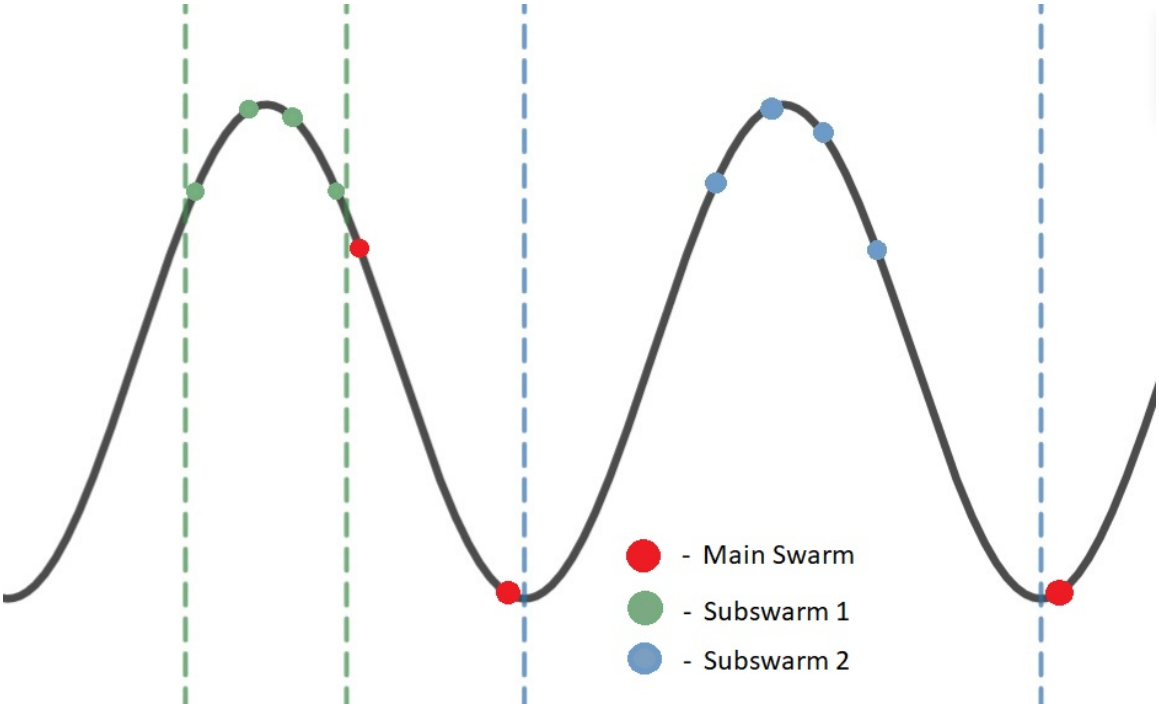

Figure 4.3: A depiction of a false optimum shown with subswarm 1, and a depiction of an ideal radius shown with subswarm 2.

could converge to a location just outside of the radius of another subswarm, as the converging subswarm may not have been able to find a stronger position elsewhere in the function (see Figure 4.3).

Taking these factors into consideration, an ideal radius size would be the size of the current hill that the subswarm is converging within. This guarantees that only a single optimum would exist within the radius, being the peak of the hill. As well, it mitigates the unwanted consequence of creating false optima around the edges of the radius. If these false optima are located close to a local minimum in the problem space, then it is likely that a nearby subswarm would converge to the neighbouring hill instead of the suboptimal location.

Another attribute of an ideal radius would be that it is asymmetric depending on the location of the subswarm and shape of the problem landscape. One could calculate the size of a hill, and the radius of a subswarm could be set to that value and not change. This however would not work entirely, as the radius is centered around the subswarm global best. If the global best is not located near the peak of the hill it is searching, then the radius would not cover the entire hill and would cover an area outside of the current hill. Covering an area outside of the current hill should be avoided, as it is possible that optima may exist within the area and the algorithm

would be unable to locate it.

By allowing a radius to be asymmetric, for each dimension the distance from the subswarm best to the nearest minimum peak can be stored and used as a much more accurate way of assigning an area in the search space to be searched. This approach would overcome the issues previously described, where radii either cover too much area and overlap other peaks, or cover too little and form false optima within the hills. A formula to calculate a subswarm radius that would meet all these requirements is unknown at this time, and the development of one is left as future work.

4.3 Alternative Velocity Update Formulas

The NichePSO algorithm uses a cognitive velocity update formula to update the velocity and position of all particles in the main swarm. Particles that have been assigned to a subswarm have their velocities and positions updated using the GCPSO update formulas. This section looks at several alternative velocity update formulas for both the main swarm particles and those within a subswarm. Details describing each modified formula are provided, along with experiments and results comparing each modified formula to the original formulas.

4.3.1 Vanilla PSO

The first tests done are to test whether the use of the GCPSO social update formula is beneficial to the NichePSO algorithm. This is tested by removing the updates for the global best particles and instead using the tradition PSO update formula for every particle. A description of the vanilla PSO algorithm and formulas can be found in Chapter 2.

4.3.2 Fast Convergence PSO

The Fast Convergence PSO (FCPSO) was proposed in [29], and is experimented with to try and improve the convergence speed of the subswarms. The FCPSO formula uses the traditional PSO social update velocity formula, and adds a new term referred to as the particle mean dimension (Pmd) of the subswarm particles. Particles within a subswarm S are updated using

$$v_i(t+1) = v_i(t)*W+C_1*rand*(\bar{x}_i-x_i)+C_2*rand*(\bar{G}-x_i)+C_3*rand*(Pmd_S(t)-x_i) \quad (4.2)$$

In the above equation, the Pmd of a subswarm is calculated by averaging for each dimension the position of every particle within the subswarm using $Pmd_S = (x_{i1} + x_{i2} + ... + xiD)/D$, where D is the number of dimensions in the problem space. For all experiments performed with this update formula, a value of $c_3 = 1.6$ was used to satisfy the equation $c_1 + c_2 + c_3 \geq 4$ given in the original paper [29].

4.3.3 Predator-Prey PSO

The Predator-Prey PSO (PPPSO) was proposed in [33], and is experimented with to improve particle diversity and exploration in the main swarm. The PPPSO is based on the interactions in nature between predators and prey, where prey will explore and search for food while simultaneously avoiding predators searching for food of their own. Simulating this scenario causes prey to explore new areas that they otherwise would not have searched to both avoid predators and find food or shelter.

In implementation, the PPPSO algorithm is designed to promote swarm diversity during a run and help particles break free of local optima when trapped. During a run, prey particles are updated using the traditional PSO velocity and position update formulas described in Chapter 2. The difference is that each particle is influenced by a fear factor P_f, which is the probability of the particles direction being adjusted when influenced by a predator. If a particles fear probability is met, then the particles velocity and position are updated using

$$v_i(t+1) = v_i(t) * W + C_1 * rand * (\bar{x}_i - x_i) + C_2 * rand * (\bar{G} - x_i) + C_3 D(d) \quad (4.3)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (4.4)$$

In the above equations, the term $D(d)$ calculates how strongly the predator effects the particle in the current dimension. The variable d is the distance between the particle and the predator, and the function $D(x)$ is an exponential decreasing function $D(x) = ae^{-bx}$. The values of a and b are changeable parameters, where a represents the maximum strength of the predators influence on a particle, and b represents the distance between the predator and particle that the effect is significant. Finally, predator particles are updated using the equation

$$v_i(t+1) = c_4(\bar{G} - \bar{x}_i) \tag{4.5}$$

The PPPSO algorithm can be integrated with the NichePSO algorithm by treating the main swarm as a set of prey, and the subswarm best particles are treated as predators. The update formula for the prey particles is modified by removing the social term from the equation so that particles in the main swarm do a local search instead of converging to an already discovered location. As well, preliminary experiments showed that removing the predator update formula shown in Equation 4.5 and instead updating predator particles using the GCPSO formula improved results.

In all PPPSO runs, the parameter values of a and b are set to match the values used in the original paper [33]. The values set are $a = 0.1X_{max}$, $b = 10/X_{max}$, and a fear factor of $P_f = 0.01$. The value of X_{max} refers to the upper bound of the current function being optimized. Finally, a value of $C_3 = 2$ is used for the fear term to match the original paper.

4.3.4 Experimental Results

Experiments were conducted to compare the impact on performance between the various velocity update formulas discussed. For each test, the total number of optima located was recorded, and each result was averaged over 30 iterations. Velocity specific parameters were set to match the values mentioned previously in their respective subsections. For each run, the NichePSO algorithm used the radiusOOB modification and the alternative subswarm creation method modification. Results are shown in Table 4.11. Constant parameters used for each run are as follows:

- Number of particles: 100
- Particles created per subswarm: 1
- Inertia: 0.7
- C_1 and C_2: 1.2
- Accuracy threshold: 0.01

Looking at Table 4.11, the first observation to be made is that using the vanilla PSO update formula results in significantly worse performance, with the algorithm failing to find a single optimum for several of the problems. This decrease in performance is likely due to subswarms converging to suboptimal values, and then they

Function	Dimensions	Global Optima	Budget	GCPSO	Vanilla	FCPSO	PPPSO
Five Uneven Peak Trap	1	2	50K	2	1.3	2	2
Himmelblau	2	4	50K	4	0.2333	4	4
Shubert	2	18	200K	17.9667	0.0667	17.8667	17.8667
Shubert	3	81	400K	52.6	0	53	53.6667
Composite function CF3	2	6	200K	5.1333	0.0333	5.3667	5.3
Composite function CF3	3	6	400K	4	0	4	4
Composite function CF3	5	6	400K	4	0	4	4

Table 4.11: The average number of optima located for each velocity update formula tested.

are unable to continue searching without the use of the GCPSO formulas. One could argue that this decrease in performance is due to each subswarm only containing two particles, which would greatly impact the vanilla formulas as the subswarms would converge very quickly. However, follow up tests found that even with each subswarm containing five particles each, there is no significant change in performance with the vanilla PSO formulas.

When looking at the results of the FCPSO and PPPSO formulas, there are slight variations in results compared to the original GCPSO formula. Since the particles within a subswarm continue to use the GCPSO update formula with these modifications, the use of these formulas only impacts the exploration and convergence speed of the subswarms. For the FCPSO formula it is likely that even though subswarms are converging faster, whether a subswarm is detected or not comes down to the GCPSO formula thoroughly searching the local neighbourhood. For the PPPSO formula, the results suggest that the increase in exploration is not enough to make a significant difference. Additional tests with the PPPSO formula showed a similar lack of results when the parameters were modified to try and increase its effect on the particles. While neither of these two formulas performed poorly, the downside is that they have a slight increase in computational cost and complexity, which one should consider when trying to decide which formula to proceed with.

4.4 Reinitialization

A drawback of the NichePSO algorithm is that there is a natural limitation to the number of optima that can be tracked during a single run. In the traditional NichePSO algorithm, a subswarm is created by merging a converged particle with its closest neighbour in the main swarm. This means that each time a subswarm is

created, a minimum of two particles are removed from the main swarm. With that in mind, if a run begins with 100 particles in the main swarm, the most amount of optima that could be tracked in a perfect scenario would be 50. Even if the alternative subswarm creation method is used, the algorithm is still limited to one optimum tracked per initial particle. This is also assuming every subswarm is able to detect a unique optimum and sufficiently search the local neighbourhood and converge to the peak, which is difficult to ensure.

In order to improve performance of the NichePSO algorithm in such a scenario, there are two options. The first is that the number of particles used in the main swarm could be increased. While this is a simple solution to the issue, it is not an efficient or scalable solution. As the number of particles increase the computational speed decreases, and it becomes more and more difficult to prevent the subswarms from searching the same areas of the search space.

The second option is to increase the number of optima that can be tracked by a single subswarm. This is done by using a reinitialization method, where subswarms are reintroduced back into the main swarm once the subswarm has fully converged. This approach has the benefit of overcoming the limitations of the traditional NichePSO algorithm without needing to increase the number of particles used.

4.4.1 Implementation

For a reinitialization method to work, a method of determining whether a subswarm has converged or not is needed. It is important that this method allows subswarms a sufficient amount of time to search their local neighbourhood before reinitialization. Once a subswarm is reintroduced back into the main swarm, no more searching will occur in that neighbourhood.

To achieve this, the partitioning criteria procedure from the traditional NichePSO algorithm is leveraged. To recap, a particle in the main swarm is said to have converged if the variance of its fitness value $\sigma_i < \delta$ over three iterations. This procedure is utilized again by checking whether the particles within a subswarm have a fitness variance smaller than δ. If every particle in a subswarm meets this criteria, then the subswarm is considered converged. At this point, the subswarms best location is recorded and the particles are reintroduced back into the main swarm at a random location. The radius of the subswarm remains in the same location, that way other particles and subswarms in the area are discouraged from searching the same location multiple times.

4.4.2 Improving Reintroduction

A problem that can be observed with relocating particles randomly is that these particles may be located within an area of the search space that has already been searched. As a run progresses this becomes more and more likely as the search space is further explored. To maximize the potential of a reinitialization method, particles should be placed in areas of the search space that have not been searched yet. This would maximize exploration and reduce the chance of particles converging to an optimum already located.

To address this problem a modification is proposed that places particles between existing subswarms. Two random subswarms are selected (including subswarms that have already converged), and the middle location for each dimension is calculated. A particle is then placed in a location close to this determined point. By placing particles in the locations between subswarms, the chance of a particle being reintroduced into an already searched area is reduced.

A negative aspect to this approach is that particles are unable to reinitialize into a location in the search space outside of every existing subswarm towards the edges of the search space. To increase exploration under this method, particles are reinitialized in between subswarms 50% of the time, and the other 50% are reinitialized randomly.

4.4.3 Modifying the Convergence Test

It was observed in preliminary experiments that many subswarms were reinitialized before fully converging to the peak of the optimum they were tracking. If a subswarm has not fully explored the local neighbourhood they are searching when they are considered converged, then there is no way for the algorithm to return to this neighbourhood later on and finish this search. Since the radii of converged subswarms remain in place after reinitialization, any particles that try to search this area later on will be unable to explore the area.

This issue was found to be caused by the alternative subswarm creation method used. As per the results of previous experiments, the alternative creation method performs better when less particles are created for each subswarm. This observation however has a negative effect when combined with reinitialization. If only a single particle is created for each subswarm, then the two particles within each subswarm will converge quickly. The convergence test will then determine that the subswarm has converged and reinitialize the particles before they are finished exploring the neighbourhood. Evidence of this can be seen in Table 4.12.

Function	Dimensions	Global Optima	Budget	Particles created			
				1	2	3	4
Five Uneven Peak Trap	1	2	50K	2	2	2	2
Himmelblau	2	4	50K	4	4	4	4
Shubert	2	18	200K	16.1333	18	18	17.9667
Shubert	3	81	400K	1	11	33.8333	49.5
Composite function CF3	2	6	200K	4	4.1	5.2	5.2667
Composite function CF3	3	6	400K	2.4	4	4	4
Composite function CF3	5	6	400K	0.0333	0	0.5333	3.6333

Table 4.12: A performance comparison of combining the alternative creation method with reinitialization.

Along with the original convergence test proposed, a stricter convergence test is tested alongside the original convergence test. Instead of utilizing the partitioning criteria test used for particles in the main swarm, a more specialized convergence test will be used. A subswarm is considered converged if the distance between the subswarm best location and every particle within the subswarm is less than some distance value $d = 0.001$. By using distance instead of fitness variance, it can be ensured that all particles are located in almost identical locations before a subswarm is reinitialized.

4.4.4 Initial Experiments

The first experiments performed compare the results of the original reinitialization method, a reinitialization method using the stricter position based convergence test, and a reinitialization method utilizing the pseudo-random positioning technique that positions particles between subswarms. All techniques were run 30 times, and each run recorded the total number of unique global optima detected for the various benchmark functions. Every run also uses the radiusOOB modification and alternative subswarm creation method both proposed previously. Using the results of Table 4.12, each reinitialization method uses four created particles per subswarm. The original method without reinitialization uses one created particle per subswarm. Results are shown in Table 4.13. Parameters for each run are as follows:

- Number of particles: 100
- Inertia: 0.7
- C_1 and C_2: 1.2
- Accuracy threshold: 0.01

Function	Dimensions	Global Optima	Budget	No Reinitialize	Basic Reinitialize	Middle Reinitialize	Position Reinitialize
Five Uneven Peak Trap	1	2	50K	2	2	2	2
Himmelblau	2	4	50K	4	4	4	4
Shubert	2	18	200K	17.9667	18	18	18
Shubert	3	81	400K	52.6	52.2667	52.3	48.8
Composite function CF3	2	6	200K	5.1333	5.8333	5.8333	4.0667
Composite function CF3	3	6	400K	4	4	4	4
Composite function CF3	5	6	400K	4	3.1333	2.9333	3.5667

Table 4.13: A comparison of performance for the different reinitialization methods presented.

Function	No Reinitialization	Using Reinitialization
Five Uneven Peak Trap	100	100.6667
Himmelblau	100	100.8
Shubert	100	274.8
Shubert	100	760.2667
Composite function CF3	100	136.3333
Composite function CF3	100	170
Composite function CF3	100	313.9667

Table 4.14: Total number of subswarms created with and without the use of reinitialization.

The first observation is that incorporating reinitialization seemed to have changed the performance very little. One might see these results and think that this is because very few subswarms are reinitializing, thus acting similarly to the original algorithm without the use of reinitialization. A quick experiment was done to test this idea, where the total number of subswarms created were counted. Results are shown in Table 4.14.

It can be seen in Table 4.14 that reinitialization does create on average more subswarms than without using reinitialization. This means that the similarities in performance are likely due to coincidence as opposed to the reinitialization approach not having an impact on the algorithm. Functions with a larger number of global optima are shown to be impacted a lot more by reinitialization, as seen with the Shubert functions used.

The use of the middle reinitialize method and the position convergence test does not have a positive impact on the reinitialization approach. The pseudorandom middle method shows results suggesting that using a completely random location for new particles is just as effective. The position convergence test shows a decrease in performance for multiple functions used, suggesting that the original test proposed works

better with the NichePSO algorithm and the modifications utilized.

4.4.5 A Trivial Convergence Test

As previously discussed, the use of a convergence test relies on subswarms containing multiple particles, as using fewer particles per subswarm causes the subswarm to be reinitialized prematurely. It would be beneficial to instead use a convergence test that would work if subswarms used only a couple particles each. Previous experiments showed that when using the alternative creation method, creating less particles correlated to a stronger performance. If a convergence test was implemented that was able to combine the strong results of using less particles with the enhanced exploration abilities of reinitialization, then the resulting algorithm would likely perform better than its counterparts.

While experimenting with different convergence tests, a trivial test was implemented to test the theory. Instead of testing for convergence each iteration, a subswarm is allowed to exist for only a certain number of iterations ι before being reinitialized. Empirical testing and observations made from Table 4.5 (correlation between performance and number of iterations) has determined that $\iota =$ number of dimensions $* 300$ iterations for each subswarm provides sufficient time for subswarms to converge fully.

A downside of this method is that a lot more iterations are required to provide the subswarms a sufficient amount of time to form, converge and reinitialize successfully. While this approach does allow the number of particles used to remain small, the number of function evaluations needed to complete a run remains roughly the same as increasing the number of particles and shortening the number of iterations.

Experimental results comparing this existence method to the standard reinitialization methods are shown in Table 4.15. Parameters all remain the same as the previous experiment, except that for the existence reinitialization method only a single particle is created per subswarm as opposed to four.

It can be seen in Table 4.15 that allowing subswarms to exist for some number of iterations does show signs of improved performance over the use of a convergence test. While this reinitialization approach does show a worse performance for the 2D composite function, all other functions show either an increase or even levels of performance despite the simple nature of the existence approach. If this approach is replaced by a convergence test that successfully allows all subswarms to converge sufficiently, while also reinitializing subswarms that converge to their peak quickly,

Function	Dimensions	Global Optima	Budget	No Reinitialize	Basic Reinitialize	Existence Reinitialize
Five Uneven Peak Trap	1	2	50K	2	2	2
Himmelblau	2	4	50K	4	4	4
Shubert	2	18	200K	17.9667	18	18
Shubert	3	81	400K	52.6	52.2667	59.2667
Composite function CF3	2	6	200K	5.1333	5.8333	4.3333
Composite function CF3	3	6	400K	4	4	4
Composite function CF3	5	6	400K	4	3.1333	4

Table 4.15: A comparison of performance for the existence reinitialization method.

one would expect an even greater increase in performance. Finding such a convergence test is left as future work for the NichePSO algorithm.

Chapter 5

Modified NichePSO Algorithms

This chapter proposes two modified NichePSO algorithms utilizing the modifications and experiments performed in the previous chapters. Experimental results are included that compare these modified algorithms to each other and the original NichePSO algorithm. Each of the proposed algorithms is described in complete detail below.

5.1 NichePSO-R Algorithm

The NichePSO-R algorithm is similar to the original NichePSO algorithm with only a few modifications to its behaviour. It utilizes the no merge and radiusOOB modifications to overcome the merging subswarm problem, while also modifying the alternative subswarm creation strategy to boost performance. A large number of particles are used which are initialized across the search space, and each one forms its own subswarm and converges to the nearest optimum. The NichePSO-R algorithm is described in Algorithm 3.

Particles are initialized in a lattice formation across the search space to spread out each particles area of effect. Initial velocities are initialized randomly within the range of $v_i \in [-0.5, 0.5], v_i \neq 0$.

Particles in the main swarm are updated using a cognitive only update formula, described in Chapter 2 Equations 2.6 and 2.7. Particles within a subswarm are updated using the GCPSO update formulas described in Chapter 2 Equations 2.3 and 2.4. The radius of each subswarm is calculated by taking the maximum distance between the subswarm best and each particle within the subswarm, using Equation 2.8.

If a particle intersects with the radius of a subswarm, and that particle is not a

```
Initialize particles in main swarm;
while stopping criteria not met do
    Update position of all particles in main swarm;
    Evaluate fitness of particles in main swarm;
    foreach subswarm S do
        Update position of all particles within S;
        Evaluate fitness of all particles within S;
        Recalculate the radius of S;
    end
    foreach particle P do
        foreach subswarm S do
            if P intersects with S & P is not a subset of S then
                Flag P as out of bounds;
            end
        end
        if P is not within any subswarm radius then
            Remove out of bounds flag from P;
        end
    end
    foreach particle P in the main swarm do
        if partitioning criteria for P is met then
            Create new subswarm from particle P;
        end
    end
end
```

Algorithm 3: The NichePSO-R Algorithm

member of the subswarm, then a flag is set for that particle. If a particle has this flag set, then it is considered to be travelling in an out of bounds area of the search space, and is unable to update its personal best fitness. If a particle has this flag set and is no longer intersecting with the radius of another subswarm, then the flag is removed and the particle is free to update its personal best fitness again. This behaviour applies to all particles, whether they be in the main swarm or within a created subswarm.

A particle in the main swarm is considered converged if after $e = 3$ iterations, its variance in fitness $\sigma < \delta = 0.0001$. Values for e and δ may be changed, but for the experiments performed in this thesis these values were set to match the original NichePSO algorithm [5]. Once a particle in the main swarm converges, it is removed from the main swarm and creates its own subswarm. κ new particles are created and positioned near the converged particle, and a subswarm is formed with the original particle that converged and these created particles. For the experiments performed in this chapter, a value of $\kappa = 1$ is used.

5.2 NichePSO-S Algorithm

The NichePSO-S algorithm uses a smaller number of particles that are frequently reinitialized randomly throughout the search space. This approach gives the algorithm a high exploration ability, while also allowing particles that find favourable positions sufficient time to explore their local neighbourhood before being moved to a new area of the search space. The NichePSO-S algorithm is described in Algorithm 4.

Particles are initialized in a lattice formation across the search space to spread out each particles area of effect. Initial velocities are initialized randomly within the range of $v_i \in [-0.5, 0.5], v_i \neq 0$.

Particles in the main swarm are updated using a cognitive only update formula, described in Chapter 2 Equations 2.6 and 2.7. Particles within a subswarm are updated using the GCPSO update formulas described in Chapter 2 Equations 2.3 and 2.4. The radius of each subswarm is calculated by taking the median distance between the subswarm best and each particle within the subswarm, using Equation 5.1.

$$R_i = Median\{|S_{x_j,i} - S_{\bar{x},i}|\} \tag{5.1}$$

A particle in the main swarm is considered converged if after $e = 3$ iterations,

```
Initialize particles in main swarm;
while stopping criteria not met do
    Update position of all particles in main swarm;
    Evaluate fitness of particles in main swarm;
    foreach subswarm S do
        Update position of all particles within S;
        Evaluate fitness of all particles within S;
        Recalculate the radius of S;
    end
    foreach subswarm S do
        if subswarm S meets convergence test then
            Reinitialize particles in subswarm S;
        end
        foreach subswarm T do
            if S and T intersect then
                Reinitialize particles in weaker subswarm;
            end
        end
    end
    foreach particle P in the main swarm do
        if partitioning criteria for P is met then
            Create new subswarm from particle P;
        end
    end
end
```

Algorithm 4: The NichePSO-S Algorithm

its variance in fitness $\sigma < \delta = 0.0001$. Values for e and δ may be changed, but for the experiments performed in this thesis these values were set to match the original NichePSO algorithm [5]. If a particle in the main swarm converges, it is removed from the main swarm and creates its own subswarm. κ new particles are created, and a subswarm is formed with the original particle that converged and these created particles. For the experiments performed in this chapter, a value of $\kappa = 1$ is used.

A subswarm is considered converged once it has existed for ι iterations. For the experiments in this chapter, the value ι = number of dimensions$*300$ was used. When a subswarm has existed for ι iterations, its best location is recorded and its original particle is reinitialized back into the main swarm with a new random location. The additional κ particles created from the creation strategy are removed, so that more and more particles are not created exponentially as the run progresses.

When two subswarms intersect, both subswarms are checked for which subswarm best has a stronger fitness. The original particle that created this subswarm with the weaker fitness is reinitialized back into the main swarm with a new random location. The additional κ particles created when the subswarm was created are removed.

5.3 Experimental Setup

To thoroughly compare the results of the original NichePSO, the NichePSO-R and the NichePSO-S algorithms, all 20 benchmark functions from the CEC'2013 niching competition are used [20]. As well, the smallest accuracy threshold of 0.0001 that the competition uses is also used. This setup compares the performance of each proposed NichePSO algorithms along with the original in a strict competition environment. All results are averaged over 30 individual runs.

Results for the original NichePSO algorithm were collected using the diversity merge modification described in Chapter 3. This is done to keep the algorithm as close to the original as possible, while also mitigating the effect of the merging subswarm problem. The NichePSO and NichePSO-R algorithms use 250 particles for each run. This number was chosen based off of experimental results shown in Table 4.6. The NichePSO-S algorithm uses 80 particles per run, as its use of reinitialization allows it to perform well using less particles than the others. The inertia value for each run is set to $W = 0.7$, and the cognitive and social components are set to $c_1 = c_2 = 1.2$.

5.3.1 Performance Measures

To better compare each proposed algorithm, multiple performance measures are used based off of measures described in [25]. The first measure is the average number of global optima located. This measure has been used regularly throughout this thesis, and will serve as a strong baseline for measuring the performance for each algorithm.

Along with the average global optima found, the average peak ratio for each algorithm is recorded for each function. The peak ratio is calculated by global optima found / total global optima, and represents the percentage of global optima located for each run.

The next performance measure is the success rate of each run. A success is recorded if all global optima are located within a single run. The success rate is calculated by # of successes / # of runs, and represents the percentage of individual runs that are able to locate all global optima within the function. An algorithm may outperform another for average number of optima found, but the success rate of the algorithm may be lower. If one prioritizes finding all optima as opposed to finding the most on average, then this measure will show which algorithm is a better choice for the task.

Finally for each run the total amount of time needed to perform the run is recorded. This measure compares the computational time of each algorithm, and compares whether one algorithm takes significantly longer to run than another. One algorithm may outperform another across all functions, however if the same algorithm takes significantly longer to complete for only a small increase in performance, one may consider using the weaker algorithm to save time and resources. All tests are performed on the same machine environment. Since execution time is dependant on the machine environment that the tests are being run on, all time results will be normalized and displayed as relative speeds as well as absolute times.

The original NichePSO algorithm is given a relative speed of 1. The NichePSO-R and NichePSO-S algorithms are given a unit speed that is relative to the run time of the original algorithm. For example, a relative speed of 2 means that the run took twice as long to complete as the original algorithm. A relative speed of 0.5 means that the run completed in half the time as the original algorithm. This relative speed measurement allows the time results to be valid across all machine environments, and provides an easy measure for comparing additional algorithms to the three discussed.

5.4 Experimental Results

Results comparing the average number of global optima detected are shown in Table 5.1. As well, the peak ratio for each function is shown in Table 5.2. For most functions used, the NichePSO-R and NichePSO-S algorithms found as many or more optima than the original NichePSO algorithm. The only functions where the proposed algorithms lost were composite function 3 (2D) and composite function 4 (3D). For all other functions, the NichePSO-R and NichePSO-S showed either an equal or greater performance.

To validate the results shown in Tables 5.1 and 5.2, Mann-Whitney U tests were performed to compare each individual performance across all functions. The p-values for each test are shown in Table 5.3. All p-values that are considered statistically significant with a confidence level of $p < 0.05$ are shown in bold.

The NichePSO-R algorithm outperformed the original NichePSO algorithm on nine out of the twenty benchmark functions used. Out of those nine functions, eight performances were statistically significant improvements over the original algorithm. The original NichePSO algorithm outperformed the NichePSO-R algorithm on two benchmark functions, but only one was statistically significant. All remaining results for both algorithms are either equal, or too similar to be considered a significant difference.

The NichePSO-S algorithm outperformed the original NichePSO algorithm on eleven out of twenty benchmark functions. Out of these eleven functions, nine showed an improvement that was statistically significant. The original NichePSO algorithm outperformed the NichePSO-S algorithm on three functions, two of which were statistically significant. All remaining results are either equal, or too similar to be considered a significant difference.

Overall both the NichePSO-R and NichePSO-S algorithm show a considerable improvement over the NichePSO algorithm. This improvement becomes more pronounced when one considers that the NichePSO algorithm being tested uses the diversity merge modification to mitigate the impact of the merging subswarm problem. Compared to the vanilla NichePSO algorithm proposed in [5], both modified versions greatly outperform the original.

The differences between the NichePSO-R and NichePSO-S algorithms are much less noticeable. Out of the twenty functions used, the NichePSO-R algorithm performed better on six than the NichePSO-S. Out of these six performances, four were a statistically significant improvement. The NichePSO-S algorithm outperformed the

Function	Dimensions	Global Optima	Budget	NichePSO	NichePSO-R	NichePSO-S
Five Uneven Peak Trap	1	2	50K	A: 2 S: 0	A: 2 S: 0	A: 2 S: 0
Equal Maxima	1	5	50K	A: 4.3 S: 1.1492	A: 5 S: 0	A: 5 S: 0
Uneven Decreasing Maxima	1	1	50K	A: 1 S: 0	A: 1 S: 0	A: 1 S: 0
Himmelblau	2	4	50K	A: 4 S: 0	A: 4 S: 0	A: 4 S: 0
Six-Hump Camel Back	2	2	50K	A: 1.1333 S: 0.3457	A: 2 S: 0	A: 2 S: 0
Shubert	2	18	200K	A: 14.8333 S: 1.3412	A: 18 S: 0	A: 18 S: 0
Vincent	2	36	200K	A: 20.0333 S: 1.8286	A: 24.4 S: 1.5222	A: 30.5 S: 1.3834
Shubert	3	81	400K	A: 52.5667 S: 2.7877	A: 71.7 S: 2.8905	A: 67.3667 S: 3.1237
Vincent	3	216	400K	A: 43.7333 S: 2.7156	A: 59.8 S: 3.4481	A: 72.9333 S: 3.8946
Modified Rastrigin	2	12	200K	A: 5.9667 S: 3.2108	A: 12 S: 0	A: 12 S: 0
Composite Function 1	2	6	200K	A: 5.9667 S: 0.1826	A: 5.9667 S: 0.1826	A: 4.5333 S: 0.6288
Composite Function 2	2	8	200K	A: 6.1 S: 1.3983	A: 7.8667 S: 0.3457	A: 6.8 S: 0.7611
Composite Function 3	2	6	200K	A: 5.3667 S: 0.8087	A: 4.6 S: 0.6215	A: 4.0667 S: 0.2537
Composite Function 3	3	6	400K	A: 4 S: 0	A: 4 S: 0	A: 4 S: 0
Composite Function 4	3	8	400K	A: 5.0333 S: 0.7649	A: 4.6 S: 0.6215	A: 4.0667 S: 0.2537
Composite Function 3	5	6	400K	A: 4 S: 0	A: 4 S: 0	A: 4 S: 0
Composite Function 4	5	8	400K	A: 2.4 S: 0.5632	A: 3.3333 S: 0.7112	A: 3.2 S: 0.5509
Composite Function 3	10	6	400K	A: 0 S: 0	A: 0 S: 0	A: 2.3 S: 0.8367
Composite Function 4	10	8	400K	A: 0 S: 0	A: 0 S: 0	A: 0.1 S: 0.3051
Composite Function 4	20	8	400K	A: 0 S: 0	A: 0 S: 0	A: 0 S: 0

Table 5.1: The average (A) and standard deviation (S) comparing the NichePSO algorithm to the modified versions proposed.

Function	Dimensions	Global Optima	Budget	NichePSO	NichePSO-R	NichePSO-S
Five-Uneven-Peak Trap	1	2	50K	1	1	1
Equal Maxima	1	5	50K	0.86	1	1
Uneven Decreasing Maxima	1	1	50K	1	1	1
Himmelblau	2	4	50K	1	1	1
Six-Hump Camel Back	2	2	50K	0.5667	1	1
Shubert	2	18	200K	0.8241	1	1
Vincent	2	36	200K	0.5565	0.6778	0.8472
Shubert	3	81	400K	0.649	0.8852	0.8317
Vincent	3	216	400K	0.2025	0.2769	0.3377
Modified Rastrigin	2	12	200K	0.4972	1	1
Composite Function 1	2	6	200K	0.9944	0.9944	0.7556
Composite Function 2	2	8	200K	0.7625	0.9833	0.85
Composite Function 3	2	6	200K	0.8944	0.7667	0.6778
Composite Function 3	3	6	400K	0.6667	0.6667	0.6667
Composite Function 4	3	8	400K	0.6292	0.6583	0.6417
Composite Function 3	5	6	400K	0.6667	0.6667	0.6667
Composite Function 4	5	8	400K	0.3	0.4167	0.4
Composite Function 3	10	6	400K	0	0	0.3833
Composite Function 4	10	8	400K	0	0	0.0125
Composite Function 4	20	8	400K	0	0	0

Table 5.2: The peak ratio of each algorithm across all benchmark functions.

NichePSO-R algorithm on four of the twenty functions, three of which were statistically significant. All other results were either even or too similar to be considered a significant difference.

Another aspect to consider when comparing the algorithm is the success rate for each function, shown in Table 5.4. Of the three algorithms, the NichePSO-R shows on average to have the highest success rate for the most functions. The original NichePSO algorithm had the highest success rate for the composite function 3 (2D), but aside from that function the NichePSO-R algorithm has a success rate that is either higher or equal to the other algorithms for all remaining functions.

The final performance measure to compare the algorithms is computational execution time. For every run, the time it took to perform the search is recorded, averaged and reported in Table 5.5. A ratio comparing each recorded time to the original algorithm is shown in Table 5.6.

For the first 10 functions used, in general the original NichePSO algorithm has the fastest execution times. However, a lot of these functions are relatively small, and the actual differences in time for these functions is also small. For example, for the Five Uneven Peak Trap function, the NichePSO-S took around 13.6 times longer to execute than the original NichePSO algorithm. However, the actual difference in time when looking at Table 5.5 is about 700ms, or less than a second difference.

Function	NichePSO to NichePSO-R	NichePSO to NichePSO-S	NichePSO-R to NichePSO-S
Five Uneven Peak Trap	0.9999	0.9999	0.9999
Equal Maxima	**0.0271**	**0.0271**	0.9999
Uneven Decreasing Maxima	0.9999	0.9999	0.9999
Himmelblau	0.9999	0.9999	0.9999
Six-Hump Camel Back	**0.0001**	**0.0001**	0.9999
Shubert	**0.0001**	**0.0001**	0.9999
Vincent	**0.0001**	**0.0001**	**0.0001**
Shubert	**0.0001**	**0.0001**	**0.0001**
Vincent	**0.0001**	**0.0001**	**0.0001**
Modified Rastrigin	**0.0001**	**0.0001**	0.9999
Composite Function 1	0.9999	**0.0001**	**0.0001**
Composite Function 2	**0.0001**	0.05	**0.0001**
Composite Function 3	**0.0003**	**0.0001**	**0.0017**
Composite Function 3	0.9999	0.9999	0.9999
Composite Function 4	0.2757	0.6384	0.5419
Composite Function 3	0.9999	0.9999	0.9999
Composite Function 4	**0.0001**	**0.0001**	0.6031
Composite Function 3	0.9999	**0.0001**	**0.0001**
Composite Function 4	0.9999	0.5093	0.5093
Composite Function 4	0.9999	0.9999	0.9999

Table 5.3: P-values comparing each algorithms results using Mann-Whitney U Tests.

Function	Dimensions	Global Optima	Budget	NichePSO	NichePSO-R	NichePSO-S
Five-Uneven-Peak Trap	1	2	50K	1	1	1
Equal Maxima	1	5	50K	0.6667	1	1
Uneven Decreasing Maxima	1	1	50K	1	1	1
Himmelblau	2	4	50K	1	1	1
Six-Hump Camel Back	2	2	50K	0.1333	1	1
Shubert	2	18	200K	0.0333	1	1
Vincent	2	36	200K	0	0	0
Shubert	3	81	400K	0	0	0
Vincent	3	216	400K	0	0	0
Modified Rastrigin	2	12	200K	0	1	1
Composite Function 1	2	6	200K	0.9667	0.9667	0.0667
Composite Function 2	2	8	200K	0.1667	0.8667	0.1667
Composite Function 3	2	6	200K	0.5333	0.0667	0
Composite Function 3	3	6	400K	0	0	0
Composite Function 4	3	8	400K	0	0	0
Composite Function 3	5	6	400K	0	0	0
Composite Function 4	5	8	400K	0	0	0
Composite Function 3	10	6	400K	0	0	0
Composite Function 4	10	8	400K	0	0	0
Composite Function 4	20	8	400K	0	0	0

Table 5.4: The success rate of each algorithm across all benchmark functions.

Function	Dimensions	Global Optima	Budget	NichePSO	NichePSO-R	NichePSO-S
Five Uneven Peak Trap	1	2	50K	55.1	114.5333	748.5
Equal Maxima	1	5	50K	97.9	46.8	753.7667
Uneven Decreasing Maxima	1	1	50K	148.1	61.2	806.5
Himmelblau	2	4	50K	69.3	116.1333	358.0667
Six-Hump Camel Back	2	2	50K	98.2	72.7333	380.4333
Shubert	2	18	200K	321.633	503.3333	544.5667
Vincent	2	36	200K	162.767	292.8333	421.9667
Shubert	3	81	400K	573.8	726.5333	609.7333
Vincent	3	216	400K	314.4	518.9667	410.3
Modified Rastrigin	2	12	200K	118.5	79.9333	388.6667
Composite Function 1	2	6	200K	16448.7	10431.3	16077.2
Composite Function 2	2	8	200K	16432.6	11148.133	16148.63
Composite Function 3	2	6	200K	17112.4	11837.4	16791.23
Composite Function 3	3	6	400K	22179.6	17994.9	21882.77
Composite Function 4	3	8	400K	23226	19877	22537.63
Composite Function 3	5	6	400K	32669	31898.5	31604
Composite Function 4	5	8	400K	34295.7	33824.4	33311.3
Composite Function 3	10	6	400K	61339.8	56294.6	55740.8
Composite Function 4	10	8	400K	60252.9	60184.6	59282.1
Composite Function 4	20	8	400K	112178	109355.27	108358.8

Table 5.5: The average run time in milliseconds for each algorithm across all benchmark functions.

Function	Dimensions	Global Optima	Budget	NichePSO	NichePSO-R	NichePSO-S
Five Uneven Peak Trap	1	2	50K	1	2.0786	13.5844
Equal Maxima	1	5	50K	1	0.4780	7.6994
Uneven Decreasing Maxima	1	1	50K	1	0.4132	5.4456
Himmelblau	2	4	50K	1	1.6758	5.1669
Six-Hump Camel Back	2	2	50K	1	0.7407	3.8741
Shubert	2	18	200K	1	1.5649	1.6931
Vincent	2	36	200K	1	1.7991	2.5925
Shubert	3	81	400K	1	1.2662	1.0626
Vincent	3	216	400K	1	1.6507	1.3050
Modified Rastrigin	2	12	200K	1	0.6745	3.2799
Composite Function 1	2	6	200K	1	0.6342	0.9774
Composite Function 2	2	8	200K	1	0.6784	0.9827
Composite Function 3	2	6	200K	1	0.6917	0.9812
Composite Function 3	3	6	400K	1	0.8113	0.9866
Composite Function 4	3	8	400K	1	0.8558	0.9704
Composite Function 3	5	6	400K	1	0.9764	0.9674
Composite Function 4	5	8	400K	1	0.9863	0.9713
Composite Function 3	10	6	400K	1	0.9177	0.9087
Composite Function 4	10	8	400K	1	0.9989	0.9839
Composite Function 4	20	8	400K	1	0.9748	0.966

Table 5.6: The relative time values of each algorithm across all benchmark functions.

When comparing the times for the composite functions used, for a lot of functions the runtimes are very similar. On average the NichePSO algorithm is the slowest for these functions, however once again the differences are only at most a few seconds overall. What this shows is that all three algorithms have a very similar execution time, and that one does not need to worry about any algorithm taking significantly longer to run. If one considers this conclusion with the fact that the performance of the NichePSO-R and NichePSO-S is significantly higher than the NichePSO function, then it provides more supporting evidence that the proposed modified NichePSO algorithms outperform the original.

When comparing the NichePSO-R and NichePSO-S algorithms to each other, both have their own strengths and merits. The NichePSO-R algorithm performs the best consistently and on the most function used, while also having the best success rate and even a fast computation time relative to the other algorithms. The NichePSO-S algorithm also performs well, and is the only algorithm to be able to locate any optima on the functions with 10 dimensions. Overall, both the NichePSO-R and NichePSO-S algorithms show a considerable improvement over the original NichePSO algorithm.

Chapter 6

Conclusions and Future Work

The goal of this thesis is to analyze the NichePSO algorithm in detail and gain an understanding of its strengths and flaws.

This thesis shows that the merging subswarm problem is a prevalent issue with the traditional NichePSO algorithm, and its impact is analyzed in-depth. Several alternative subswarm merging strategies were discussed and are shown to overcome the merging subswarm problem, greatly increasing the performance and stability of the algorithm.

Several individual components of the NichePSO algorithm are studied and modified, with the goal of improving performance and reducing computational cost. Several parameters that are utilized are tested for their impact and strong parameter values are suggested.

An alternative subswarm creation method is proposed where new particles are created and added to the search. This new method is shown to greatly outperform the vanilla nearest neighbour creation method that the NichePSO algorithm previously utilized.

The radiusOOB approach to particle and subswarm intersection is proposed, which treats the radii of subswarms as out of bounds regions for particles not within the subswarms. Several methods for calculating the radius of a subswarm are looked at to work with this method, and an optimal approach to sizing a subswarms radius is discussed in detail.

Reinitialization methods are proposed and analyzed, where subswarms that converge can be randomly placed back into the main swarm to improve exploration. This approach is shown to provide comparable results using a significantly less number of particles than previously discussed methods.

All of the experiments and analyses performed lead to the proposal of the NichePSO-

R and NichePSO-S algorithms. Both of these algorithms were tested against the original in a competitive environment, and both show a significant performance increase over the original NichePSO algorithm across several benchmark functions.

Future work involves further studies to gain additional insight into the NichePSO algorithm and its modified versions. This includes deeper analyses using standard benchmark functions, as well as applying the algorithms to other problem types such as dynamic environments. Studying the NichePSO algorithm's ability to optimize large and dynamic data sets will better analyze its applicability to dynamic real world problems.

The NichePSO-R algorithm can be further improved by developing a new method for calculating the radius of a subswarm, as discussed in detail in Chapter 4. This new method will use a dynamic, asymmetric radius for each subswarm that adapts to the current hills and valleys in the problem search space. Such a radius will maximize the impact of the radiusOOB method proposed, and minimize the areas in the search space that are searched by multiple subswarms.

The NichePSO-S algorithm can be further improved by developing a new convergence test that works with subswarms that use a small number of particles. The current existence method used in the NichePSO-S algorithm works but is trivial. A specific test that provides subswarms enough time to fully explore their neighbourhood while also limiting the number of iterations subswarms exist after converging will improve efficiency and maximize exploration.

The scalability of the NichePSO-R and NichePSO-S algorithms should be explored in further detail. Both algorithms struggle to perform on functions with a high number of dimensions, and studies to analyze the weaknesses and limitations of the algorithm in higher dimensional problems will be a great benefit. As well, further work is needed to compare the NichePSO variants to current state-of-the-art MMO algorithms, and improve the algorithms to make them more competitive.

Bibliography

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

[2] S. Bird and X. Li. Adaptively choosing niching parameters in a pso. *GECCO '06: Genetic and Evolutionary Computation Conference*, pages 3–10, 2006.

[3] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013. Prediction, Control and Diagnosis using Advanced Neural Computations.

[4] R. Brits, A. P. Engelbrecht, and F. van den Bergh. Locating multiple optima using particle swarm optimization. *Applied Mathematics and Computation*, 189(2):1859–1883, 2007.

[5] R. Brits, A.P. Engelbrecht, and F. van den Bergh. A niching particle swarm optimizer. *Proc. 4th Asia–Pac. Conf. Simulat. Evol. Learn. (SEAL)*, pages 692–696, 2002.

[6] R. Brits, A.P. Engelbrecht, and F. van den Bergh. Solving systems of unconstrained equations using particle swarm optimization. *IEEE Conference on Systems, Man, and Cybernetics*, pages vol. 3, page 6, 2002.

[7] R. Brits, A.P. Engelbrecht, and F. van den Bergh. Scalability of niche pso. *Swarm Intelligence Symposium (SIS)*, pages 228–234, 2003.

[8] Z. G. Chen, Z. H. Zhan, D. Liu, S. Kwong, and J. Zhang. Particle swarm optimization with hybrid ring topology for multimodal optimization problems. *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2044–2049, 2020.

[9] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137:106040, 2019.

[10] A. P. Engelbrecht, B. S. Masiye, and G. Pampard. Niching ability of basic particle swarm optimization algorithms. *IEEE Swarm Intelligence Symposium, (SIS)*, pages 397–400, 2005.

[11] A.P. Engelbrecht and L.N.H. van Loggerenberg. Enhancing the nichepso. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, pages 2297–2302, 2007.

[12] D.E. Goldberg. Genetic algorithms in search, optimization and machine learning. *Addison-Wesley Longman Publishing Co. Inc.*, 1989.

[13] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. *Proc. 2nd Int. Conf. Genet. Algorithms*, pages 41–49, 1987.

[14] A. Gopal, M. M. Sultani, and J. C. Bansal. On stability analysis of particle swarm optimization algorithm. *Arabian Journal for Science and Engineering*, 45:2385–2394, 2019.

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

[16] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, USA, 1975.

[17] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks.*, 4:1942–1948, 1995.

[18] K.D. Koper, M.E. Wysession, and D.A. Wiens. Multimodal function optimization with a niching genetic algorithm: A seismological example. *Bulletin of the Seismological Society of America*, 89:978–988, 1999.

[19] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. *Genetic and Evolutionary Computation Conference*, pages 105–116, 2004.

[20] X. Li, A.P. Engelbrecht, and M. Epitropakis. Benchmark functions for cec'2013 special session and competition on niching methods for multimodal function optimization. *Evol. Comput. Mach. Learn. Group*, 2013.

[21] X. Li, M. G. Epitropakis, K. Deb, and A. P. Engelbrecht. Seeking multiple solutions: An updated survey on niching methods and their applications. *IEEE Transactions on Evolutionary Computation*, 21(4):518–538, 2017.

[22] W. Luo, Y. Qiao, X. Lin, P. Xu, and M. Preuss. Hybridizing niching, particle swarm optimization, and evolution strategy for multimodal optimization. *IEEE Transactions on Cybernetics*, 57:495–503, 2020.

[23] S. C. Maree, T. Alderliesten, D. Thierens, and P. A. N. Bosman. Benchmarking the hill-valley evolutionary algorithm for the gecco 2018 competition on niching methods multimodal optimization. *GECCO '18: Genetic and Evolutionary Computation Conference*, 2018.

[24] S. C. Maree, T. Alderliesten, D. Thierens, and P. A. N. Bosman. Real-valued evolutionary multi-modal optimization driven by hill-valley clustering. *GECCO '18: Genetic and Evolutionary Computation Conference*, pages 857–864, 2018.

[25] J. Mwaura, A. P. Engelbrecht, and F. V. Nepocumeno. Performance measures for niching algorithms. *IEEE Congress on Evolutionary Computation (CEC)*, pages 4775–4784, 2016.

[26] D. Parrott and X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10(4):440–458, 2006.

[27] K. E. Parsopoulos and M. N. Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, 2004.

[28] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas, and M.N. Vrahitis. Stretching technique for obtaining global minimizers through particle swarm optimization. *Proceedings of the Particle Swarm Optimization Workshop*, pages 22–29, 2001.

[29] A. Sahu, S. K. Panigrahi, and S. Pattnaik. Fast convergence particle swarm optimization for functions optimization. *Procedia Technology*, 4:319–324, 2012.

[30] I. L. Schoeman and A. P. Engelbrecht. Using vector operations to identify niches for particle swarm optimization. *Cybernetics and Intelligent Systems*, pages 361–366, 2004.

[31] I. L. Schoeman and A. P. Engelbrecht. A parallel vector-based particle swarm optimizer. *Adaptive and Natural Computing Algorithms*, pages 268–271, 2005.

[32] Y. Shi and R. Eberhart. An empirical study of particle swarm optimization. *Proceedings of the 1999 Conference on Evolutionary Computation*, 3:1945–1950, 1999.

[33] A. Silva, A. Neves, and E. Costa. An empirical comparison of particle swarm and predator prey optimisation. *Proceedings of the 13th Irish International Conference on Artificial Intelligence and Cognitive Science*, pages 103–110, 2002.

[34] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, 2002.

[35] A. Ward, J. K. Liker, J. J. Cristiano, and D. K. Sobek. The second toyota paradox: How delaying decisions can make better cars faster. *Sloan Manag. Rev*, pages vol. 4, page 129, 1995.

[36] K.C. Wong, K.S. Leung, and M. H. Wong. Protein structure prediction on a lattice model via multimodal optimization techniques. *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 155–162, 2010.

[37] X. Yin and N. Germay. A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. *Artificial Neural Nets and Genetic Algorithms.*, pages 450–457, 1993.

[38] J. Zou, Qi. Deng, J. Zheng, and S. Yang. A close neighbor mobility method using particle swarm optimizer for solving multimodal optimization problems. *Information Sciences*, 519:332–347, 2020.