New Methods of Increasing the Effectiveness of Particle Swarm Optimization

July 2008

Kevin Jack Binkley

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
at the
Graduate School of Science and Technology of
Keio University

Abstract

In this dissertation, two new methods for increasing the effectiveness of particle swarm optimization (PSO) are proposed.

First, to combat premature convergence of the swarm, velocity-based reinitialization (VBR) is developed. Through studies of swarm simulations, we observed that before the swarm stagnates, the median velocity drops significantly and can be used as an effective trigger for reinitialization. With VBR, the problem of premature convergence is alleviated; and the optimization process is able to focus on one minimum at a time.

In experiments on benchmark functions from the literature, we apply VBR to enhance the global best, local best, and von Neumann neighborhood PSO algorithms. We observe that the VBR enhanced PSO algorithms achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal functions.

Second, we develop the stop and go particle swarm optimization algorithm, a new method to dynamically adapt the effective PSO population size. Stop and go (SG) PSO takes advantage of the fact that in practical problems there is a limit to the required accuracy of the optimization result. In SG-PSO, particles are stopped when they have approximately reached the required accuracy. Stopped particles do not consume valuable function evaluations. Still, the information contained in the stopped particles' state is not lost, but rather as the swarm evolves, the particles may become active again, behaving as a new memory for the swarm. The SG method is a straightforward modification to the standard PSO algorithm.

In experiments on benchmark functions from the literature, we apply SG to enhance the global best, local best, and von Neumann neighborhood PSO algorithms. We observe that both the SG enhanced standard global best PSO algorithm and the SG enhanced von Neumann neighborhood PSO algorithms achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal benchmark functions. Furthermore, we observed that the SG enhanced standard local best PSO algorithm achieves improved results on both unimodal and multimodal benchmark functions.

Acknowledgments

First, I would like to thank my advisor Professor Masafumi Hagiwara, whose advice and guidance while at Keio University have proved invaluable to me during my studies at Keio University.

Second, I would like to thank the members of my thesis review committee, Professor Akito Sakurai, Professor Hideo Saito, and Professor Yoshikazu Yamamoto. Their helpful comments and suggestions, have greatly improved this thesis.

Third, I would like to thank the many anonymous reviewers of the papers I have submitted. Their valuable comments have often helped greatly improve the quality of the published papers, and indirectly this thesis.

And finally, I thank my family and friends who have provided valuable support throughout my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Particle swarm optimization (PSO) is a population-based evolutionary computation algorithm introduced by Kennedy and Eberhart in 1995 [32]. The PSO algorithm has proven both straightforward to implement and effective as a global optimization tool. Since its introduction, it has been widely studied and widely applied. In this section, we look some of the many applications of PSO and then give an outline of this thesis.

1.1 Applications of Particle Swarm Optimization

Over 10 years have passed since the introduction of PSO, and research concerning PSO is continuing to grow steadily. A search by topic on the Thomson ISI Web of Knowledge database [1] yielded 1007 articles from 2004 to 2008. A breakdown by field of study (see Table 1.1) shows that PSO is widely applied in physics, telecommunications, and engineering applications. By publication year, there were 82, 173, 350, 301, and 101 articles published in the years 2004, 2005, 2006, 2007 and 2008 (through March) respectively.

PSO is certainly not the only global optimization tool. It is natural to ask: what are the unique advantages of PSO?

First of all, PSO is a very simple, easy to understand, and natural algorithm. PSO has comparatively few easy to set parameters, there being only two key equations to PSO optimization (see Section 2.5.2 for the details). Both are straightforward and derive from a natural analogy of birds swarming or fish schooling. Furthermore, these equations are both easy to code in software.

In terms of applications, particle swarm can be applied to almost any real-valued optimization problem. PSO requires a fitness function to evaluate positions in the domain space of the function being optimized. Unlike gradient-based optimization methods, PSO does not need gradient information about the function being optimized, and thus PSO is applicable to a wider range of problems. PSO is also less susceptible to getting stuck in a local minimum when compared to gradient-based methods.

As an example, let us consider optimizing the weights of a neural network (NN). This is the application in the original paper introducing PSO in 1995 by Kennedy and Eberhart [32]. In this paper, PSO was compared the gradient-based backpropagation method. It was found that PSO and backpropagation performed equally well.

Since PSO can be applied to learn the weights of a NN, the applications of PSO are numerous. Some applications of PSO that have a appeared in the literature are games, control, and image processing.

Table 1.1: Number of PSO publications by field study (Search performed in April 2008, using the keywords "Particle Swarm Optimization").

Field of Study	Number of Publications
Engineering, Electrical and Electronic	162
Computer Science, Artificial Intelligence	83
Computer Science, Theory and Methods	53
Applied Mathematics	51
Computer Science, Interdisciplinary Applications	50
Telecommunications	45
Applied Physics	33
Automation and Control Systems	32
Operations Research and Management Science	25
Multidisciplinary Engineering	22

In the games domain, PSO has been applied to checkers [18], tic tac toe [40], and the iterated prisoner's dilemma (IPD) [19]. For checkers and tic tac toe, PSO was applied to learn the NN weights for a position evaluation function. For the IPD [19], two variants of PSO were applied. In one, the standard PSO algorithm was applied to learn the weights of a NN for evaluation the IPD position, the output indicating whether to cooperate or defect. In the other, a binary version of PSO [33] was applied where each particle represented the IPD strategy itself as a 64-bit vector.

PSO has also been applied to control applications. One example is the parameter selection of internal combustion engines. In [43], PSO was applied to find the optimal engine power optimization parameters.

Examples of PSO applied in the image processing domain are image enhancement and noise cancellation. In [7], PSO was applied to enhance images by maximizing the number of pixels on the edges. PSO was found to perform better than genetic algorithms both in terms of computational time and objective evaluation. In [47], PSO was applied to optimize the templates for a discrete time cellular neural network for the application to image noise cancellation.

1.2 Contribution of this Dissertation

In applying PSO, one chooses a population size, and the three PSO parameters, w, c_1, and c_2(see Section 2.5.2). The three PSO parameters may be chosen from [11] to guarantee convergence, and then the only parameter remaining to determine is the population size. In addition, fixing the population size at 40 has been shown to work well for a wide variety of problems. However, there are still problems where the standard PSO algorithm converges too slowly or too fast.

Converging to a solution too fast is known as premature convergence. Premature convergence is a frequent problem when emphasizing local search (exploitation) over global search (exploration). The standard PSO algorithm offers a good balance between local and global search. However, with some modifications the standard algorithm can be improved to offer an even better balance between local and global search. In this thesis, we propose and study two methods of enhancement.

First, we propose a new method to detect premature convergence and restart the swarm, velocity-based reinitialization (VBR). With VBR, a local search can be conducted without getting stuck in a local minimum. After the local search, reinitialization occurs, resuming global search. Due to VBR, the problem of premature convergence is alleviated; and the optimization process is able to focus on one minimum at a time.

Second, we propose a new method to dynamically adapt the PSO population size, stop and go (SG). SG-PSO takes advantage of the fact that in practical problems there is a limit to the required accuracy of the optimization result. In SG-PSO, particles are stopped when they have approximately reached the required accuracy. Stopped particles do not consume valuable function evaluations. Still, the information contained in the stopped particles' state is not lost, but rather as the swarm evolves, the particles may become active again, behaving as a memory for the swarm. SG-PSO, like VBR-PSO, applies reinitialization to abort local search and continue global search. In SG-PSO, reinitialization occurs when all particles have stopped.

1.3 Outline of this Dissertation

In Chapter 2, an overview of popular evolutionary computation (EC) methods is given. The overview is followed by a detailed discussion of particle swarm optimization, beginning with an discussion of the history and development of the PSO algorithm, and concluding with a discussion of some of the many notable PSO methods that have been applied in the literature.

Chapter 3 gives the details of the velocity based reinitialization (VBR) method for PSO developed in this thesis. Experiments are conducted demonstrating the benefits of the VBR method when applied to standard global best PSO, the local best PSO, and the von Neumann neighborhood PSO algorithms.

Chapter 4 gives the details of the stop and go (SG) method for PSO developed in this thesis. Experiments are conducted demonstrating the benefits of the SG method when applied to standard global best PSO, the local best PSO, and the von Neumann neighborhood PSO algorithms.

Chapter 5 concludes this thesis with a summary of our findings and our view concerning the future directions of PSO research.

Chapter 2

Evolutionary Computation Methods

2.1 Introduction

Particle swarm optimization (PSO) is certainly not the only evolutionary algorithm (EA) for real-valued global function optimization. There are many methods of evolutionary computation that have been developed for real function optimization. The common evolutionary algorithms (EA) include simulated annealing (SA) [35], genetic algorithms (GA) [20], genetic programming (GP) [36], evolution strategies (ES) [3], evolutionary programming (EP) [52], differential evolution (DE) [46], and particle swarm optimization (PSO) [32].

Of these methods, for real function optimization, the most commonly applied are ES, EP, DE, and PSO. In the following sections, we will take a brief look at these optimization methods, followed by a detailed look at PSO and its history. We will review the no free lunch theorem and then look at how PSO has been compared to the other evolutionary algorithms in the literature.

2.2 Evolutionary Strategy

The evolutionary strategy (ES) algorithm is a population based optimization algorithm. The basic ES consists of a population of individuals representing possible solutions to the problem at hand. Each individual is a vector in the domain space of the function being optimized. Similar to a GA, the individuals are evolved through mutation and recombination.

ES algorithms are described with the notation $(\mu/\rho, \lambda)$ or $(\mu/\rho + \lambda)$, where μ is the size of the population, λ is the number of children to generate through recombination and mutation, and ρ is the number of parents recombined to produce a child. The comma or plus indicate whether selection for the next generation includes the previous parents or not (the plus indicating the inclusion of the previous parents).

Pseudocode for a typical ES algorithm is given in Figure 2.1. The population is initialized randomly, and then the following optimization process is performed until a stopping criterion is meant (generally a required minimum is achieved, or a maximum number of generations is exhausted). First, the children are created using recombination and then mutation. Second, selection of the next generation is performed by taking the best λ individuals (from either the children only or parents plus children depending on the selection strategy: plus or comma).

```
1: initialize population randomly
2: while stopping condition not reached do
3:    // generate λ children
4:    for i = 1 to λ  do
5:       pick ρ parents with replacement
6:       children[i] = perform recombination using parents
7:       children[i] = mutate(children[i])
8:    end for
9:    // select μ parents for next generation based on the type of selection algorithm
10:   if  ES is of type (μ/ρ, λ) then
11:      // select from the λ children
12:      population = selection(children)
13:   else // ES is of type (μ/ρ + λ)
14:      // select from the union of the current μ parents and λ children
15:      population = selection(parents + children)
16:   end if
17: end while
```

Figure 2.1: The basic ES algorithm.

There are two different types of recombination methods commonly applied. For the first method, the child vector is the centroid of the ρ parents, a straightforward average. A second method is to choose each component of the child vector from a random parent vector.

For mutation, a random Gaussian mutation is commonly applied to the child vector. Each component of the child vector should be mutated on a different scale $\sigma_i N(0,1)$, where σ_i is a standard deviation, i represents a component of the child vector, and $N(0,1)$ is a normally distributed random variable. In a more general case, mutation can be performed by adding a random vector from a general normal distribution in n-dimensions, in which case the correlation matrix must be specified.

For an n-dimensional domain space, there are thus n parameters (σ_i) describing how mutation is performed. These parameters are called strategy parameters and are attached to the individual and adjusted throughout the evolution.

Why? Consider optimizing the one dimensional function $f(x) = x^2$, hoping to get to an accuracy of 10^{-6}. If an individual starts at $x = 100$, the first σ setting should be 100 for fast convergence. However, with $\sigma = 100$, it is very unlikely the individual will land within 10^{-6} of the minimum 0. The step size needs to be decreased as the minimum is approached.

In practice, the distance to the minimum is not known. The ES solution is to attach the strategy parameters to the individual, they evolve too, undergoing recombination and mutation. Recombination is performed just as described above for the position vector. Mutation is performed by

$$\sigma_i(t+1) = \sigma_i(t) * exp(\tau N(0,1)) \tag{2.1}$$

where τ is $1/\sqrt{n}$, where n is the number of dimensions in the domain space.

2.3 Evolutionary Programming

Evolutionary programming (EP) a population based EA that has proven effective at the task of optimizing real-valued functions in high dimensional spaces [52, 17]. Let there be *npop* individuals in the population.

```
1: // an individual consists of the pair of position and strategy components: (x_i, σ_i)
2: initialize population, (x_i, σ_i) randomly
3: while stopping condition not reached do
4:   for i = 1 to npop do // Create the children
5:     r_1 = N(0,1) // normally distributed random variable with μ = 0 and σ = 1
6:     for d = 1 to number of dimensions do
7:       r_2 = N(0,1)
8:       r_3 = N(0,1)
9:       x'_id = x_id + σ_id r_2
10:      σ'_id = σ_id exp(τ' r_1 + τ r_3)
11:    end for
12:  end for
13:  evaluate fitness of children // f(x'_i) for each child
14:  // the next generation is selected from a competition between parents and children
15:  (x_i, σ_i) = tournament_selection(union of the parents and children, npop)
16: end while
```

Figure 2.2: The basic EP algorithm.

In EP, an individual is represented by two vectors in the domain space of the function being optimized, its position $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ and its strategy parameters, $\vec{\sigma}_i = (\sigma_{i1}, \sigma_{i2}, \ldots, \sigma_{in})$, where i is the individual index and n is the number of dimensions of the domain space.

Each generation, each parent creates a child according to the following equations:

$$x'_{id} = x_{id} + \sigma_{id} N_d(0,1) \tag{2.2}$$

$$\sigma'_{id} = \sigma_{id} exp(\tau' N(0,1) + \tau N_d(0,1)) \tag{2.3}$$

Where i is the index of the individual being updated, d is the coordinate of the individual being updated, $N_j(0,1)$ is a normally distributed random variable generated for each coordinate, and $N(0,1)$ is generated only once each generation for each individual. The variables τ and τ' are set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$ respectively [52, 17].

Pseudocode for the EP algorithm is given in Figure 2.2. The population is initialized randomly, and then the following optimization process is performed until a stopping criterion is meant (generally a required minimum is achieved, or a maximum number of generations is exhausted). First, the children are created and evaluated with the fitness function. Second, tournament selection is then performed with the union of the parents and children to select *npop* individuals to become the new parents of the next generation.

Tournament selection is defined as follows. A tournament size, t, is assumed to be given. For each participant, p, pick t competitors uniformly at random from the incoming population (in this case, union of parents and children) and compare p's fitness to each competitor's fitness. p receives a point for each win (better fitness), the total wins being p's score. After all competitions have been held, sort the individuals by number of points received and return the best *npop* individuals as the result.

2.4 Differential Evolution

Differential evolution (DE) [46] is a population based generational EA. In DE, an individual is an n-dimensional vector in the domain space of the function to be optimized (fitness function). To begin, a

population of *npop* individuals are initialized randomly. At each generation, each of the *npop* individuals undergoes mutation and crossover to produce a child. The fitness of the child is compared to the fitness of the parent and the better of the two proceeds to the next generation.

More specifically, let the current population be given by the vectors $\vec{x}_i$, where $i \in [1, npop]$.

For each parent, mutation is applied to produce an intermediate vector $\vec{v}_i$. Mutation proceeds as follows for parent i. Three unique individuals are drawn randomly from the population not including i. These indices will be called a, b, and c. The mutation vector $\vec{v}_i$ is generated by equation 2.4.

$$\vec{v}_i = \vec{x}_a + F(\vec{x}_b - \vec{x}_c) \tag{2.4}$$

where the parameter F is constant amplification factor, normally $F \in [0, 2]$.

The next step, crossover, occurs between the parent vector, $\vec{x}_i$, and its corresponding mutation vector, $\vec{v}_i$. The components of the child vector, $\vec{u}_i$, are chosen randomly from either $\vec{x}_i$ or $\vec{v}_i$ in a manner similar to a GA's uniform crossover operator. The one difference being that one component is guaranteed to be taken from the mutation vector. Let this component be determined by the index k and choose k randomly $\in [1, n]$. The child vector, $\vec{u}_i$, is generated from equation 2.5.

$$\vec{u}_{ij} = \begin{cases} \vec{v}_{ij} & \text{if } rand() < CR \text{ or } j = k \\ \vec{x}_{ij} & \text{otherwise} \end{cases} \tag{2.5}$$

$CR \in [0, 1]$ is the crossover rate. The function $rand()$ generates a random number $\in [0, 1)$ and is generated freshly for each component j.

The final step, selection, is straightforward. The fitness of the child, $\vec{u}_i$ is compared to the parent, $\vec{x}_i$ and the better of the two goes forward to the next generation. Pseudocode for the DE algorithm is given in Figure 2.3.

There are several variants of DE, which are described with the notation: $DE/x/y/z$. In this notation, x specifies the vector to be mutated ("rand" or "best"); y is the number of difference vectors used; and z specifies the type of crossover to perform. The main DE variant described here is $DE/rand/1/bin$. The reader is referred to the DE literature for other variants [46].

According to [46], DE's control variables, *npop*, F, and CR, are not difficult to choose. Good initial recommended settings are: $npop \in [5n, 10n]$, $F = 0.5$, and $CR = 0.9$.

2.5 Particle Swarm Optimization

2.5.1 Introduction

Particle swarm optimization (PSO) is an evolutionary optimization algorithm originally introduced by Kennedy and Eberhart in 1995 [32]. It was observed that in many processes of nature such as birds flocking and fish schooling, the individuals could be simulated by rather simple rules; and furthermore, that the idea applied to function optimization.

This section is outlined as follows. First, we take a look at what we will call the standard PSO algorithm. Then, we study the history and development of the PSO algorithm. Next, we look some of the neighborhood methods that have been developed for PSO. And finally, we take a glance at the variety of PSO algorithms proposed in the literature.

```
1: initialize population, $\vec{x}_i$, randomly
2: while stopping condition not reached do
3:   for $i = 1$ to $npop$ do
4:     // perform DE mutation
5:     // $rand(N)$ produces integer $\in [1, N]$
6:     $a = rand(npop)$; while $(a == i)$ $a = rand(npop)$;
7:     $b = rand(npop)$; while $(b == i$ or $b == a)$ $b = rand(npop)$;
8:     $c = rand(npop)$; while $(c == i$ or $c == a$ or $c == b)$ $c = rand(npop)$;
9:     $\vec{v}_i = \vec{x}_a + F(\vec{x}_b - \vec{x}_c)$ // $F$ is the DE amplification factor, not a function
10:    // perform DE crossover
11:    $k = rand(n)$; // make sure the $k$ index goes to the child
12:    for $j = 1$ to $n$ do
13:      // $rand()$ produces real number $\in [0, 1)$
14:      if $rand() < CR$ or $j == k$ then
15:        $\vec{u}_{ij} = \vec{v}_{ij}$
16:      else
17:        $\vec{u}_{ij} = \vec{x}_{ij}$
18:      end if
19:    end for
20:    // perform DE selection
21:    // select the better of parent and child for next generation
22:    if $fitness(\vec{u}_i) < fitness(\vec{x}_i)$ then
23:      $\vec{x}_i = \vec{u}_i$
24:    end if
25:  end for
26: end while
```

Figure 2.3: The basic DE algorithm.

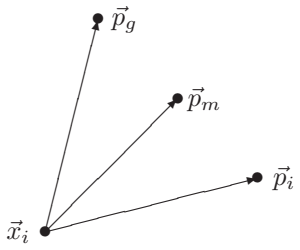

Figure 2.4: PSO Particle Dynamics. When the momentum is zero and $c_1 = c_2$, the particle, i, will move between its personal best position and the global best position, in the direction of the vector $\vec{p}_m = 0.5(\vec{p_i} + \vec{p_g})$.

2.5.2 The Standard PSO Algorithm

In the standard PSO algorithm, a population of particles is initialized randomly in the domain space of the function to be optimized. Each particle is given a random position and velocity in the n-dimensional domain space. As time progresses, the particles fly through domain space, updating their positions, velocities, and keeping track of their own personal best-ever position.

The state of a particle is represented by its position $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, velocity $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$, and its personal best position $\vec{p}_i = (p_{i1}, p_{i2}, \ldots, p_{in})$, where i is the particle index and n is the number of dimensions of the domain space. The state of the whole swarm consists of state of all the particles plus a global best position $\vec{p}_g = (p_{g1}, p_{g2}, \ldots, p_{gn})$.

The particles head toward both their own personal best and the best position observed by the whole swarm. This tendency is controlled by the following PSO equations.

$$v_{id} = wv_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id}) \tag{2.6}$$

$$x_{id} = x_{id} + v_{id} \tag{2.7}$$

where r_1 and r_2 are random numbers $\in [0, 1)$, i is the particle index, and d is the coordinate being updated. Note that the velocity and position vectors are updated on a coordinate by coordinate basis. The pseudocode for the standard global best PSO algorithm is presented in Figure 2.5.

The three key parameters to PSO are in the velocity update equation 2.6. The first component is the momentum component where the inertial constant w controls how much the particle remembers its previous velocity. The second component is the cognitive component, the acceleration constant c_1 controls the how much the particle heads toward its personal best position. The third component, referred to as the social component, draws the particle toward swarm's best ever position; the acceleration constant c_2 controls this tendency.

There are many possible PSO parameter settings. Perhaps the most popular are what we will call the standard Clerc settings [11, 8, 15] ($w = 0.729$ and $c1 = c2 = 1.49455$). These settings provide a good balance between exploration and exploitation.

For a better understanding of PSO, it is helpful to visualize the PSO particle dynamics. Let us temporarily remove the random factors by setting $r_1 = r_2 = 1$, let $c_1 = c_2$, and assume that the momentum is zero. The velocity update equation is now simplified to

$$v_{id} = c_1[(p_{id} - x_{id}) + (p_{gd} - x_{id})] \tag{2.8}$$

```
1: initialize population() // set p⃗_g to x⃗_0
2: while stopping condition not reached do
3:   for i = 1 to population size do
4:     for d = 1 to number of dimensions do
5:       r_1 = rand() // between 0 and 1
6:       r_2 = rand()
7:       v_id = w v_id + c_1 r_1 (p_id − x_id) + c_2 r_2 (p_gd − x_id)
8:       x_id = x_id + v_id
9:     end for
10:    if f(x⃗_i) < f(p⃗_i) then
11:      p⃗_i = x⃗_i
12:    end if
13:    if f(x⃗_i) < f(p⃗_g) then
14:      p⃗_g = x⃗_i
15:    end if
16:  end for
17: end while
```

Figure 2.5: The standard PSO algorithm. The optimization result is $\vec{p}_g$.

In terms of the vectors, $\vec{p}_i$ and $\vec{p}_g$, the particle $\vec{x}_i$ moves equally toward both the personal best and the global best, scaled by the factor c_1. As shown in Figure 2.4, the particle moves along the line between its current position and the average of its personal best and the global best.

In the general case where c_1 and c_2 may be different, and r_1 and r_2 are random, the particle moves between $\vec{p_i}$ and $\vec{p_g}$, determined by a weighted average of r_1c_1 and r_2c_2. Due to the randomness overshoot and undershoot occur, causing the particle to bounce around $\vec{p}_m = 0.5(\vec{p_i} + \vec{p_g})$.

In order to understand the PSO dynamics better, the distribution of the points visited around $\vec{p}_m$ was analyzed in [29] and found to be Gaussian-like but with longer tails. Further efforts were made to simply sample a probability distribution around $\vec{p}_m$, instead of applying the traditional velocity update equation [29, 31]. Results were good, but the traditional velocity update equation still performed the best. In addition to the longer tails, the following possible explanation was offered. In traditional PSO, a particle will overshoot in one direction, then return to overshoot in the other direction. However, when sampling from a probability distribution, the probability of a sample being on a particular side of $\vec{p}_m$, is independent of the side the particle is currently on. The "zig-zag" overshoot dynamics are not observed.

2.5.3 The History and Development of the PSO Algorithm

In this section, we take a walk through the history and evolution of the PSO algorithm. Some of the major developments of the PSO algorithm are given in Table 2.1. In the following sections, we will discuss each development in detail.

The First PSO Algorithm

The debut of the PSO algorithm took place in 1995 with the first paper by Kennedy and Eberhart [32]. The idea of PSO came from natural simulations such as birds flocking or fish schooling. In nature, after one bird finds food, somehow, others follow and soon the whole flock is on the food. The flock is effectively sharing information and has solved the "food-finding" optimization task.

Table 2.1: Major developments in the history of the PSO.

Date	Authors	Idea Introduced
1995	Kennedy and Eberhart [32]	The first global best PSO algorithm introduced
1995	Eberhart and Kennedy [13]	The local best PSO algorithm introduced
1999	Shi and Eberhart [44]	The inertial weight introduced
2002	Clerc and Kennedy [11]	Constriction coefficient introduced and theoretic analysis showed that with appropriate choice of parameters convergence can be guaranteed

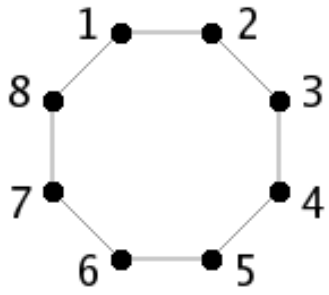

Figure 2.6: The LBEST neighborhood. For a neighborhood of size two, the particles are laid out in a circle and each particle's neighborhood is the particle itself and its two nearest neighbors. For example, particle 1's neighborhood consists of particles, 1, 8, and 2.

From observations like this, the first global best (GBEST) PSO algorithm was developed. In the first GBEST PSO algorithm, the population of particles are flown through the n-dimensional domain space of the function to be optimized. As they fly, each particle keeps track of its personal best position. The global best position is calculated and shared among all particles.

Using the notation from Section 2.5.2, the first PSO velocity update equation was

$$v_{id} = v_{id} + 2r_1(p_{id} - x_{id}) + 2r_2(p_{gd} - x_{id}) \tag{2.9}$$

In [32], this PSO algorithm was applied to training a neural network (NN) and compared to the back-propagation training method. PSO was found to train the NN just as efficiently as backpropagation. The PSO algorithm was also compared to elementary genetic algorithms and found to achieve approximately the equivalent results.

The Local Best PSO Algorithm

Shortly after introducing the GBEST PSO algorithm (see Section 2.5.3), Eberhart and Kennedy [13] extended it to include local neighborhoods, creating the LBEST family of PSO algorithms.

Recall, in the GBEST PSO algorithm, $\vec{p}_g$ is the best position observed by the whole swarm. In LBEST, $\vec{p}_g$ becomes the best position ever observed by the neighborhood. LBEST neighborhoods are defined by placing the particles a circle. A neighborhood consists of the particle and its nearest neighbors (Figure 2.6). For a neighborhood of two neighbors, particle i's neighborhood consists of particles $i-1$ and $i+1$. Larger neighborhoods are constructed in a similar manner. In LBEST, $\vec{p}_g$ is thus different for

each particle in the swarm. Neighborhoods are discussed in detail in Section 2.5.4.

A modified velocity update equation was applied

$$v_{id} = v_{id} + cr_1(p_{id} - x_{id}) + cr_2(p_{gd} - x_{id}) \tag{2.10}$$

where $\vec{p}_g$ is the neighborhood best. In addition to the LBEST neighborhood, the parameter $c \in [0.5, 2]$, and a cap, V_{max}, on the maximum allowed velocity were also studied.

The PSO algorithm was applied to train a neural network the XOR problem. PSO was found to perform as well as backpropagation. It was also found to be competitive with the elementary genetic algorithm. Overall, for these tasks, the LBEST algorithm (neighborhoods with 2 and 6 neighbors tested) did not perform as well as the GBEST version.

The Social and Cognitive Components

In [27], Kennedy introduced and studied the social and cognitive components of the PSO. The general PSO velocity update equation was given as

$$v_{id} = v_{id} + c_1r_1(p_{id} - x_{id}) + c_2r_2(p_{gd} - x_{id}) \tag{2.11}$$

In this equation, we see separate c_1 and c_2 components. Kennedy draws interesting analogies from social science.

The vector $(\vec{p}_i - \vec{x}_i)$ represents the distance of the individual i from its personal best. Thus in a social science context the term, $c_1r_1(p_{id} - x_{id})$, of the velocity update can be considered a cognitive term. It represents the tendency of the individual to return to its own previous best.

In a similar manner, the vector $(\vec{p}_g - \vec{x}_i)$ represents the distance of the individual i from its neighborhood best. And in a social science context the term, $c_2r_2(p_{gd} - x_{id})$, of the velocity update can be considered a social term. It represents the tendency of the individual to head toward the neighborhood best.

The constants, c_1 and c_2, are now referred to as the cognitive and social acceleration constants.

The Inertial Weight

In 1998, Shi and Eberhart [44] introduced the inertial weight to the PSO velocity update equation. The PSO velocity update equation was given as

$$v_{id} = wv_{id} + c_1r_1(p_{id} - x_{id}) + c_2r_2(p_{gd} - x_{id}) \tag{2.12}$$

where w is the newly introduced inertial constant. It is pointed out that by adjusting w, one can fine-tune the balance between local and global search. Experiments suggested that $w \in [0.9, 1.2]$ yields the best performance.

Further studies [45], suggested an inertial weight setting of $w = 0.8$ yielded best results. Also, it was suggested that the maximum velocity be clamped at the maximum range of each coordinate ($V_{max} = X_{max}$). Experiments with a linearly time-decreasing inertial weight were also performed (w going from 0.9 to 0.4).

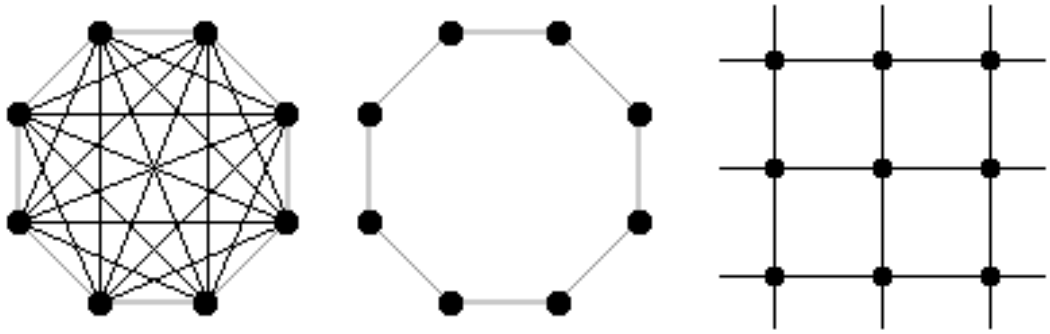

Figure 2.7: The GBEST (left), LBEST (middle), and von Neumann (right) neighborhoods.

Constriction and Guaranteed Convergence

In [11, 8], it is shown theoretically that with the appropriate parameter choices, convergence can be guaranteed. The velocity equation 2.6 can be rewritten in the following form.

$$v_{id} = \chi[v_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id})] \tag{2.13}$$

When written in this form, if χ, c_1, and c_2 are chosen as follows

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}, \text{ where } \quad \phi = c_1 + c_2, \phi > 4 \tag{2.14}$$

convergence is assured.

Using this formula, a popular choice for the PSO parameters in equation 2.6 is $w = 0.729$, $c_1 = c_2 = 1.49445$. This set of parameters, called the Clerc parameters, comes form setting $\phi = 4.1$ and choosing $c_1 = c_2$.

In [14], the Clerc parameters were investigated. Experiments were performed both with and without a velocity cap. Despite theoretically guaranteed convergence, it was concluded that a velocity cap was beneficial to prevent the particles from wasting function evaluations outside the search range.

2.5.4 Neighborhood Methods

When the standard global best PSO algorithm does not provide enough exploration (global search), neighborhoods methods are often applied [13, 28, 34, 39]. With the neighborhood methods, the particles are only aware of their neighbors as defined by the particular neighborhood method being applied. Information transfer concerning the global best can be delayed resulting in more exploration. Neighborhood methods are an important tool to balance local and global search.

Implementation is straightforward, the PSO velocity equation 2.6 is modified so that the global best $\vec{p}_g$, becomes a "neighborhood best" (including oneself).

In the most commonly applied neighborhoods, the particle's neighborhood is fixed at the beginning of the PSO algorithm. The neighborhood can be represented as a graph, where the particles are the vertices and the neighbors are connected by the edges of the graph.

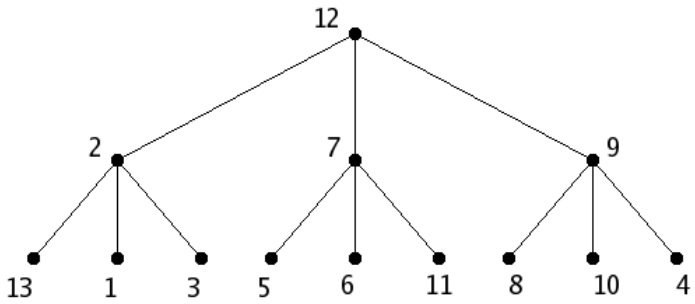

Figure 2.8: An example of 13 particles of a Hierarchical Particle Swarm in a tree of branching degree 3. The neighborhood of a particle is the particle itself and the particle above it in the hierarchy. For example, the neighborhood of particle 7 is particles 7 and 12. Similarly, the neighborhood of particle 6 is particles 6 and 7. The hierarchy, and thus the neighborhood is dynamic, being updated every time step causing the better particles to move to the top of the hierarchy.

Common Neighborhoods

Graphs showing the GBEST, LBEST, and von Neumann neighborhoods are given in Figure 2.7. The most applied neighborhood structure is the global best (GBEST) neighborhood (see Section 2.5.2). In graph form, it is the fully connected graph. With GBEST neighborhood, information about the global best is known to all particles immediately. Of all the possible neighborhoods, this neighborhood places the most emphasis on exploitation (local search).

Another commonly applied neighborhood is the local best (LBEST) neighborhood. Here the particles are placed in a circle and each particle has only two neighbors. Information about a newly discovered global best will travel around the circle slowly allowing for more exploration before the swarm finally converges.

A more connected, but not fully connected neighborhood is the von Neumann neighborhood. In this neighborhood, the particles are placed on a toroidal grid and the neighborhood is the particle and its four nearest neighbors. It has been reported that the von Neumann neighborhood performed best overall on a standard set of test functions [34].

The Hierarchical Particle Swarm

In [24, 25, 26], the hierarchical particle swarm optimizer (H-PSO) is proposed. In the H-PSO, the particles are placed in a tree of branching degree d. The neighborhood of a particle is defined as itself and the particle above it in the hierarchy. With only two particles, this neighborhood is smaller than LBEST (see Figure 2.8).

An interesting element of the H-PSO is that each generation the particles move up and down in the tree. Thus unlike fixed structure neighborhoods, GBEST and LBEST, the H-PSO applies a dynamic neighborhood.

Adjusting the balance between local and global search can be done by adjusting the branching degree, d. In the extreme, setting $d = npop - 1$ gives the H-PSO behavior similar to the GBEST neighborhood. Setting $d = 1$ (a very tall tree with no branches), results in behavior similar to LBEST.

The adaptive H-PSO (AH-PSO) algorithm [26] is proposed to change the search emphasis from global search to local search. In the AH-PSO, the branching degree is decreased with time, resulting in a change

to emphasis local search as the optimization progresses. Interestingly, this is the opposite the change from LBEST to GBEST proposed in [48]. In [48], it was suggested that the neighborhood start with the particle itself (a fully "cognitive" swarm), and gradually increase the the GBEST neighborhood.

The authors find that the H-PSO algorithm performed well on both unimodal and multimodal test functions. In addition, it was found that the AH-PSO algorithm could improve the performance of the H-PSO algorithm.

The Fully Informed Particle Swarm

In the fully informed particle swarm (FIPS) [39, 38], the whole idea of personal best and global best is thrown out. Recall that in the standard PSO (Section 2.5.2) the particles head toward a weighted average of their personal best and neighborhood best positions (controlled by the c_1 and c_2 acceleration constants). With FIPS, it is proposed instead that the particles head toward the weighted average of the personal best positions of the particles in the neighborhood. With the appropriate weights, the LBEST and GBEST PSO algorithms are clearly a subset of the FIPS algorithm.

The theory behind FIPS is as follows [39]. The PSO velocity update equation

$$v_{id} = \chi[v_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id})] \tag{2.15}$$

can be rewritten as:

$$v_{id} = \chi[v_{id} + \phi(p_{md} - x_{id}))] \tag{2.16}$$

where $\phi = \phi_1 + \phi_2$, $\phi_1 = c_1 r_1$, $\phi_2 = r_2 c_2$, and $p_{md} = (\phi_1 * p_{id} + \phi_2 * p_{gd})/(\phi_1 + \phi_2)$. Note that if $\chi = 0.7298$, and $c_1 = c_2 = 2.05$, then we have the standard Clerc settings.

It clear here that $\vec{p}_m$ is a weighted average between a particle's personal best and the neighborhood best. In [39], it is shown that $\vec{p}_m$ may be extended to become a weighted average of the whole neighborhood, and that the convergence and explosion characteristics of the swarm discussed in [11] are preserved. The general weighted FIPS equations become

$$\phi_k = r_k \frac{\phi_{max}}{|N|} \tag{2.17}$$

$$p_{md} = \frac{\sum_{k \in N} W_k \phi_{kd} p_{kd}}{\sum_{k \in N} W_k \phi_{kd}} \tag{2.18}$$

where r_k is a random number $\in [0, 1)$, $\phi_{max} = 4.1$, and N is the set of neighbors.

In [39], experiments show that FIPs produces better results than the canonical PSO algorithm.

2.5.5 Modified Particle Swarms

Many extensions of the PSO algorithm have been proposed and studied in the literature. The proposed extensions in the current literature are too numerous to cover in full. Instead, in this section we have chosen a few of our favorites to discuss.

Hierarchical PSO with Time-varying Acceleration Coefficients

In [43], the hierarchical particle swarm (HPSO) with time-varying acceleration coefficients (TVAC) was introduced and studied. TVAC is an interesting method of controlling the balance between local and

global search. In particular, as the search begins, it might be desirable to emphasis global search relative to local search. Recalling the velocity update equation

$$v_{id} = wv_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id}) \tag{2.19}$$

we observe that emphasizing global search corresponds to emphasizing the social component relative to the cognitive component. In TVAC, this is achieved by having c_1 and c_2 vary with time in opposite directions. c_1 varies from 2.5 to 0.5 and c_2 varies from 0.5 to 2.5. In addition, the inertial coefficient, w, is varied from 0.9 to 0.4.

In the HPSO, the inertial coefficient, w, is clamped to 0 in the velocity update equation. And anytime a velocity component, v_{id}, reaches 0, it is reinitialized to the maximum allowed velocity.

In this study, the mutation PSO (MPSO) was also proposed. In the MPSO, when the global best is not improving at the desired rate, with mutation probability p_m, the velocity components of the particles receive a random mutation based on the maximum allowed velocity.

Experiments are performed with the HPSO, MPSO, TVAC, and combinations with TVAC. Overall the HPSO-TVAC is found to perform the best.

The Gregarious PSO

The gregarious PSO (GPSO) [42] is a social only swarm. The PSO velocity update equation is modified by zeroing the cognitive term. As the particles lose momentum, particle restarts are applied. Also, the social acceleration coefficient is adapted during the evolution of the swarm.

The GPSO velocity update is performed as follows

$$v_{id} = \gamma r(p_{gd} - x_{id}) \tag{2.20}$$

where $r \in [0, 1]$ is a random real value, and γ is an adaptive scaling factor. γ decreases or increases each time period based upon whether the the global best improved or not. The update rule for γ is

$$\gamma = \begin{cases} max(\gamma - \delta, \gamma_{min}) & \text{if global best improved last time period} \\ min(\gamma + \delta, \gamma_{max}) & \text{otherwise} \end{cases} \tag{2.21}$$

Through the dynamic adaptation of γ, the GPSO is given some self control over the degree of global versus local search. While the global best is improving, γ will approach γ_{min} particle steps will get smaller allowing for improved local search. When the global best fails to improve, search steps will get larger and global search will be emphasized.

Particle restarts are performed when the particle position gets within a predetermined distance, ϵ, from the global best position. A restart consists of completely reinitializing the particle's velocity vector.

One challenge may be choosing the parameters concerning γ. In [42], γ was initialized to 3, and $\gamma_{min} = 2$ and $\gamma_{max} = 4$. These parameter settings were shown to work well on a standard set of benchmark functions.

Division of Labor PSO

In [50], the division of labor (DoL) PSO is introduced. The idea behind the DoLPSO, is to have the individuals dynamically switch between local and global search as needed. An individual is in either of two states: global search or local search. Individuals begin in the global search state (defined as applying the normal PSO algorithm). Every time step an individual has a probability of switching roles. The probability of switching to local search, P_{local}, is governed by the following equation

$$P_{local} = \frac{s_i^n}{\theta_i^n + s_i^n} \tag{2.22}$$

where s_i is the number of time steps since an improvement in the individual's personal best last occurred, and θ_i is the individual's current propensity for local search (θ_i varies with time). The probability of switching from local to global search is a predetermined constant p. When switching from global to local search, the particle is simply placed at the global best position, and its velocity is reinitialized based on the current velocity of the global best particle. For further details, the reader is referred to [50].

The DoLPSO was compared to GA, SA, and the basic PSO methods. DoLPSO was found to perform better or at least equally well compared to a basic PSO algorithm.

Other PSO Algorithms

There are many other interesting PSO algorithms that have been developed. Unfortunately, we cannot discuss them all in detail. Here we mention a few, in hopes that the interested reader will consult the literature.

In [49], the Cooperative PSO (CPSO) method is proposed. In the CPSO, there are n 1-dimensional swarms, one for each component of the position vector. The final position vector (n-dimensional) is built by taking one component from each of the n swarms. The problems involving the choices in construction of an n-dimensional solution vector, assignment of credit are addressed in [49]

PSO has been applied to dynamic systems to successfully track moving minima. In [16], the basic PSO equation was modified to use a random inertial weight to track a dynamic parabolic system. In [21], change of the location of the global minimum was monitored by watching for changes in the global best position. When a change of the global minimum was observed, several different methods of swarm reinitialization were proposed and studied. More advanced methods are discussed in [6].

The PSO algorithm is not to be constrained to real problem domains. For problems discrete nature, a discrete binary version of the particle swarm has been developed [33]. A swarm for permutation problems is developed in [23]. In this study, the position vector is a permutation, to update the position, the particle velocity is normalized between 0 and 1 to represent the probability of the particle heading toward the best permutation in swarm memory. A permutation-based mutation operator is applied to preserve diversity.

Multiobjective optimization occurs where there is not just one solution to the problem, but possibly many different solutions where any one of these solutions cannot be said to be better than any other. Such solutions are said to be Pareto optimal. PSO has been applied to multiobjective optimization in [41, 22, 12].

2.6 The No Free Lunch Theorem

Before we can discuss the comparison of optimization algorithms, it is important to point out and consider the no free lunch theorem(NFL) [51]. The no free lunch theorem tells us that with respect to the set of all possible functions, there is no optimization algorithm that will perform better than any other.

As an example, the NFL says that if we consider the hill climbing optimization algorithm in the context of all possible functions, it is no better than the hill descending optimization algorithm. Furthermore, random search is also no better or worse than hill climbing, either ascending or descending. Sequential search is equally fruitful. The key here is that we are in the context of all possible functions.

However, it is not time to give up and apply random search. In real engineering problems, the functions that the algorithms we design face are not randomly chosen from the set of all possible functions. On the contrary, they are far from random and tend to have some (possibly difficult to understand) structure. For example, the functions are often continuous. Consider drawing a random function out of the set of all functions. What is the probability that this function belongs to the class of continuous functions? Very small (my guess is 0).

The fact that in real engineering problems we only have to deal with a very special subset of the set of all functions gives us the ability to design optimization algorithms that are useful, and work well in practice.

When comparing different optimization algorithms, we will naturally find that the best algorithm and parameter settings for that algorithm will depend on the function or class of functions being optimized. In this thesis, we search for optimization algorithms that work well on a wide variety of benchmark problems intended to reflect the types of functions that we need to optimize in real engineering problems.

2.7 Comparisons

Since PSO was introduced [32], researchers have been comparing PSO to other optimization algorithms. Initial studies compared PSO to a GA and to neural network backpropagation [32, 13]. Results showed the PSO performed equally or better than backpropagation (PSO is much simpler to implement), and as well as a GA.

A comparison of EP and PSO is offered in [2]. Standard benchmark functions, are used for comparison in 10, 20 and 30 dimensions. The two algorithms where found to be “competitive”.

In [50], a real-valued GA, an SA, PSO, and the division of labor PSO (DoLPSO) algorithms are compared with three different benchmark functions in 10 to 50 dimensions. Results are mixed with the DoLPSO outperforming the other methods on most benchmark functions. The authors conclude that “in general the DoLPSO performs better or at least equally well compared to the bPSO”, where bPSO is a basic PSO algorithm.

Each optimization algorithm has its own strengths and weaknesses. In order to understand these, in [37], a GP is used to evolve functions that are difficult to optimize for each particular optimization algorithm. PSO, DE, the Newton-Raphson hill-climber, and the covariance matrix adaptation-ES were compared. The GP succeeded in finding functions difficult for each optimization method, and functions such that any method outperformed any other method.

Chapter 3

Velocity-based Reinitialization Particle Swarm Optimization

3.1 Introduction

For many optimization problems, the standard PSO algorithm suffers from premature convergence. In order to increase the global search capabilities, the standard PSO algorithm has frequently been extended in the literature with neighborhood methods [28, 34, 39]. All common neighborhood methods (other than global best), increase global search (exploration) at the expense of slowing or delaying local search (exploitation).

We ask: what if we could exploit and then go back and explore? It is with this goal in mind that we propose velocity-based reinitialization (VBR) [4]. VBR monitors the velocities of the swarm particles to determine if the swarm has stagnated. If the swarm has stagnated then further function evaluations are not likely to be fruitful. In order to more effectively apply function evaluations, the current result is saved and the swarm is restarted.

In addition to being straightforward to incorporate into a PSO algorithm, experiments show that VBR-PSO outperforms the standard global best PSO algorithm on multimodal benchmark functions, while achieving equivalent results on the unimodal benchmark functions. Since the neighborhood methods local best (LBEST) [28] and von Neumann (VonNeu) [34] are known to improve the standard PSO algorithm on multimodal function optimization tasks, we also applied VBR to enhance these neighborhood PSO algorithms. Both the VBR-LBEST and VBR-VonNeu PSO enhanced algorithms exhibited improvements over their non-enhanced counterparts on the multimodal benchmark functions while yielding equivalent performance on unimodal benchmark functions.

3.2 Related Research

As its name implies, critical components of the velocity-based reinitialization (VBR) approach, are the detection of swarm stagnation and reinitialization. VBR uses particle velocity as an indicator of stagnation and upon stagnation, the whole swarm is reinitialized. There are, however, other ways of determining stagnation and taking action. Here are some of the approaches seen in the PSO literature.

In [8], a "no-hope" criterion based on the spread of particles in space is introduced and applied to determine when to restart the particles. In contrast to VBR, this method places more emphasis on local search in that the particles are restarted around the previous best particle.

Complete swarm reinitialization has been proposed for use with dynamic environments [16, 21]. In [21], the environment is monitored (by for example, checking the global best to see if the evaluation changed) and when a change is detected the whole swarm is reinitialized.

Re-initialization of velocity components was applied in the "self-organizing hierarchical particle swarm optimizer" (HPSO) [43]. The HPSO method is interesting in that it keeps the particle's previous velocity fixed at zero (essentially setting $w = 0$). With $w = 0$, the particle will tend to move between the global best and its personal best until its velocity approaches zero. The HPSO algorithm checks the particle's velocity on a component by component basis and reinitializes any component velocity if it reaches zero.

In the Gregarious PSO (G-PSO) [42], a social component only version of PSO is proposed (the personal best component of the velocity update equation is not used, $c_1 = 0$). In the G-PSO algorithm, the velocity vector is completely reinitialized when the particle reaches a certain distance, ϵ, from the global best.

In [10], stagnation is determined by the number of time steps without an improvement in a particle's local best. In this study, random neighborhoods are initially generated and on stagnation, the neighborhoods are reinitialized.

VBR differs from the approaches observed in the literature in that it offers a unique velocity-based method to determine the need for complete swarm reinitialization.

3.3 Velocity-based Reinitialization Particle Swarm Optimization

We have observed many particle swarm evolutions using the standard PSO settings. One common characteristic is that the particles often find a local minimum and due to constriction continue to get closer and closer to this minimum, always improving the global best position slightly. This is fine if the minimum is a global minimum. However, on multimodal functions, there are almost always many other better minimums. We propose that rather than the slight improvements offered by continuing local search, it is often better to make note of the result found and reinitialize the swarm completely, performing global search again.

The problem left is when to restart the search. The width of the search area as determined by either the particles' position (or personal best) spread is a possibility [8]. However, we observed that often a few particles out of the whole population would be spread out and the rest would be "wasting" valuable function evaluations while waiting for the remaining particles to converge to the swarm's global best. In many cases, these few remaining particles might never get lucky enough to land near enough to the swarm's global best, improve their personal best and converge.

With these observations, we developed velocity-based reinitialization (VBR). With VBR, we estimate when most of the swarm's function evaluations are not being applied effectively and restart the swarm. In VBR, the median velocity of the swarm particles is monitored at each time step during the evolution. When the median velocity drops below a predetermined threshold, α, we declare the swarm stagnant and initiate a restart. We define stagnation very loosely as "further function evaluations are not likely to be

very productive". Specifically, stagnation is determined by the following steps.

1. calculate the Euclidean norm of each velocity vector, $n_i = ||\vec{v}_i|| = \sqrt{\vec{v}_i \bullet \vec{v}_i}$.
2. sort the norms.
3. compare the median of the velocity vector norms to the stagnation threshold, α. When the median velocity vector norm has dropped below α, the swarm is considered stagnate.

A more general method than the median is to use the n^{th} norm, where the first norm is the smallest. The n^{th} norm can be used to adjust the amount of reinitialization. In some preliminary experiments, we found that VBR was not too sensitive which norm was used. We settled on the median for this study. However, we note that the first norm does not work as well since there will be cases where one particle's velocity is very small but the swarm is still productive and reinitialization would be premature. It is actually natural for the global best particle's velocity get small quickly since its personal best is the global best. If a new global best is not found, the particle will lose momentum as it homes in on the global best (for the usual case where $w < 1$). The average of the velocity vector norms can also be considered. However, the median was chosen to be able to declare a swarm stagnate when the majority of the particles have stagnated.

The modifications to the standard PSO algorithm given in Section 2.5.2 are quite small and simple to implement. With VBR, a list of all global bests is kept, and at the beginning of each time step a check for stagnation is performed. If stagnation is detected the swarm's global best is saved and the swarm is reinitialized. When a predetermined termination condition is reached, the best of the union of the current swarm's global best and the global bests stored in the global best list is returned as the optimal value found. The pseudocode for the VBR enhanced PSO algorithm is given in Figure 3.1.

3.4 Experiments

In this section, first we discuss the common experimental settings and benchmarks to be applied during all simulations, conduct experiments to study the PSO effectiveness with respect to variation in population size, and study the performance of common PSO algorithms from the literature.

Next, we analyze the performance of VBR enhanced PSO and study how the threshold parameter, α, affects VBR. Guidelines for determining appropriate threshold parameter settings are proposed.

We show that VBR is effective at enhancing three different PSO algorithms: the standard global best, the local best neighborhood, and the von Neumann neighborhood. We observe that the VBR enhanced PSO algorithms achieve improved results on the multimodal benchmark functions with equivalent performance on the unimodal functions.

3.4.1 Common Experimental Settings

We compare the PSO algorithms using five benchmark functions commonly found in the PSO literature. The functions given in Table 3.1. The Sphere, and Rosenbrock functions are unimodal functions. The Rastrigrin, Griewank, and Schaffer's f6 functions are multimodal functions. For the curious, graphs showing these functions in two dimensions are given in Figures 3.2 - 3.6. The simulation difficulty is adjusted by varying the number of dimensions of the search domain space. Except for the Schaffer's f6

initialize population() // set $\vec{p}_g$ to $\vec{x}_0$
set global best list to empty
while stopping condition not reached **do**
  if swarm has stagnated **then**
    add swarm's global best to global best list
    reinitialize swarm
  else
    for $i = 1$ to population size **do**
      for $d = 1$ to number of dimensions **do**
        $r_1 =$ rand() // between 0 and 1
        $r_2 =$ rand()
        $v_{id} = wv_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id})$
        $x_{id} = x_{id} + v_{id}$
      end for
      if $f(\vec{x}_i) < f(\vec{p}_i)$ **then**
        $\vec{p}_i = \vec{x}_i$
      end if
      if $f(\vec{x}_i) < f(\vec{p}_g)$ **then**
        $\vec{p}_g = \vec{x}_i$
      end if
    end for
  end if
end while
add swarm's global best to global best list
return the best of the global best list

Figure 3.1: The VBR-PSO algorithm.

Table 3.1: Benchmark functions.

Name	Function
Sphere	$f_1(x) = \sum_{i=1}^{n} x_i^2$
Rosenbrock	$f_2(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
Rastrigrin	$f_3(x) = \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i) + 10]$
Griewank	$f_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(x_i/\sqrt{i}) + 1$
Schaffer's f6	$f_5(x) = 0.5 + \frac{(\sin\sqrt{x_1^2+x_2^2})^2 - 0.5}{(1.0+0.001(x_1^2+x_2^2))^2}$

Table 3.2: Search range, maximum velocity, and asymmetric initialization range.

Function	Search Range	V_{max}	Init. Range
$f_1(x)$	$(-100, 100)^n$	100	$(50, 100)^n$
$f_2(x)$	$(-100, 100)^n$	100	$(50, 100)^n$
$f_3(x)$	$(-10, 10)^n$	10	$(2.56, 5.12)^n$
$f_4(x)$	$(-600, 600)^n$	600	$(300, 600)^n$
$f_5(x)$	$(-100, 100)^n$	100	$(15, 30)^n$

function which is two dimensional, simulations were performed for all functions in 10, 20, 30, 50 and 100 dimensions, resulting in a total of 21 benchmark functions of various difficulties.

There are many simulation parameters to determine such as the maximum number of function evaluations allowed for each simulation, the size of the search domain, the stopping criterion for each benchmark function. We based our simulation settings on [43, 4, 5].

The simulations were stopped when a predetermined stopping criterion was reached or if the maximum number of permitted function evaluations was exhausted. The stopping criterion is based on function difficulty. For all functions, the stopping criterion was set to 0.01, except for the Schaffer's f6 function where the stopping criterion was set to 0.00001. The maximum number of function evaluations was set at 400000 for all simulations. All simulations were run 50 times.

In PSO, the particle positions, are normally restricted to a predetermined search range, and the velocity is restricted to a fraction of the search range [14, 15]. The search range and maximum velocities are specific to each benchmark function and are given in Table 3.2. In a manner similar to the PSO velocity update process, the position and velocity restrictions are enforced on a component by component basis. If a particle's position component exceeds the maximum range, the component is randomly placed back in the range, and the corresponding velocity component is set to V_{max}. If a particle's velocity component exceeds the maximum velocity, the component is set to the maximum velocity.

Since the global minimum of all the common PSO benchmark functions is near the origin, uniform initialization over the search domain will tend to surround the minimum. To avoid this bias, asymmetric initialization was proposed in [2]. The asymmetric initialization ranges used in this study are given in Table 3.2. The initial particle velocities are randomly chosen from between minus and plus the maximum velocity.

Unless otherwise stated, the following standard PSO algorithm parameter settings are applied. The standard PSO algorithm population is fixed at 40 particles. For the parameters, w, c_1, and c_2, we use the common Clerc settings [11, 8, 15] ($w = 0.729$ and $c_1 = c_2 = 1.49455$).

3.4.2 Choosing the Population Size

A key parameter to PSO optimization is the population size. Frequently applied population sizes observed are from 20 to 100. It has been observed in the PSO literature that the appropriate size for the particle swarm is problem dependent [9]. In general, when computational resources are fixed, the smaller populations emphasize local search, and are better for unimodal functions; whereas larger populations emphasize global search, and are needed for multimodal functions. In this section, we conduct experiments using the standard PSO settings (Section 3.4.1) varying the population from 10 to 100.

Table 3.3 and Figure 3.7, give the results. Populations of 10 and 20 observed problems with premature convergence on the 100-dimension sphere problem. Furthermore, populations of 10 and 20 performed worse than the populations of 40 or more on all of the multimodal benchmark functions. However, the smaller populations are not completely without benefit, in terms of the average number of function evaluations applied to reach the stopping criterion, a population size of 10 was by far the the fastest (Figure 3.7). Furthermore, a population of 20, performed best overall on the unimodal Rosenbrock function in 30 and 50-dimensions. Overall, we conclude that for these benchmark functions, populations of 10 and 20 and not sufficient.

Let us compare the population of 40 and the population of 100 in detail.

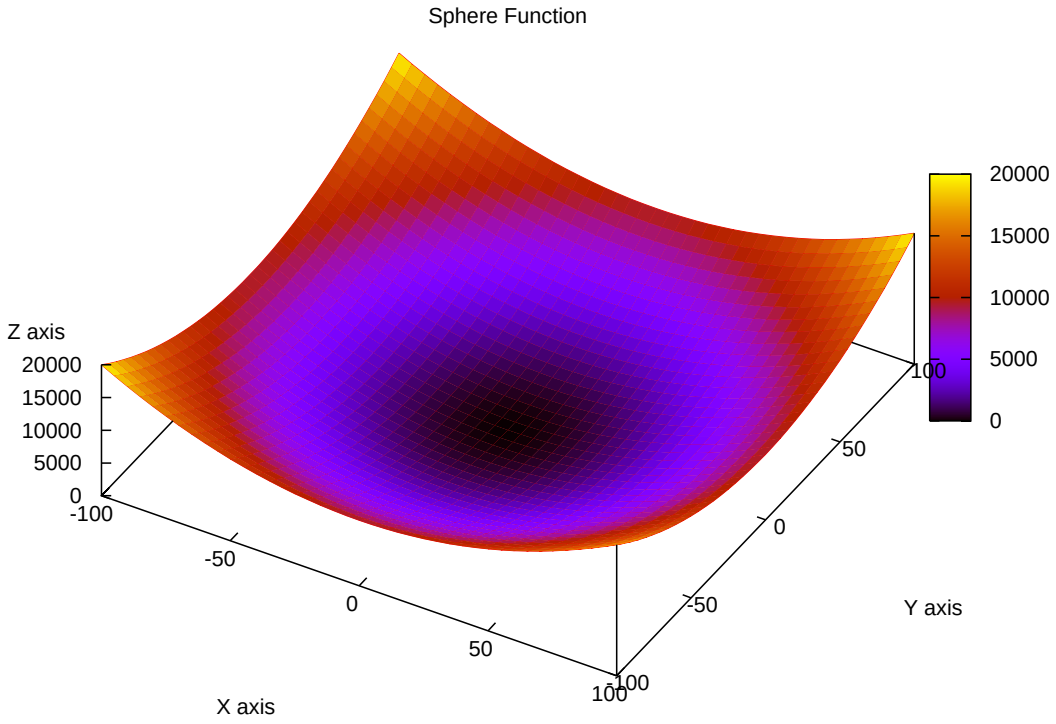

Figure 3.2: The Sphere function.

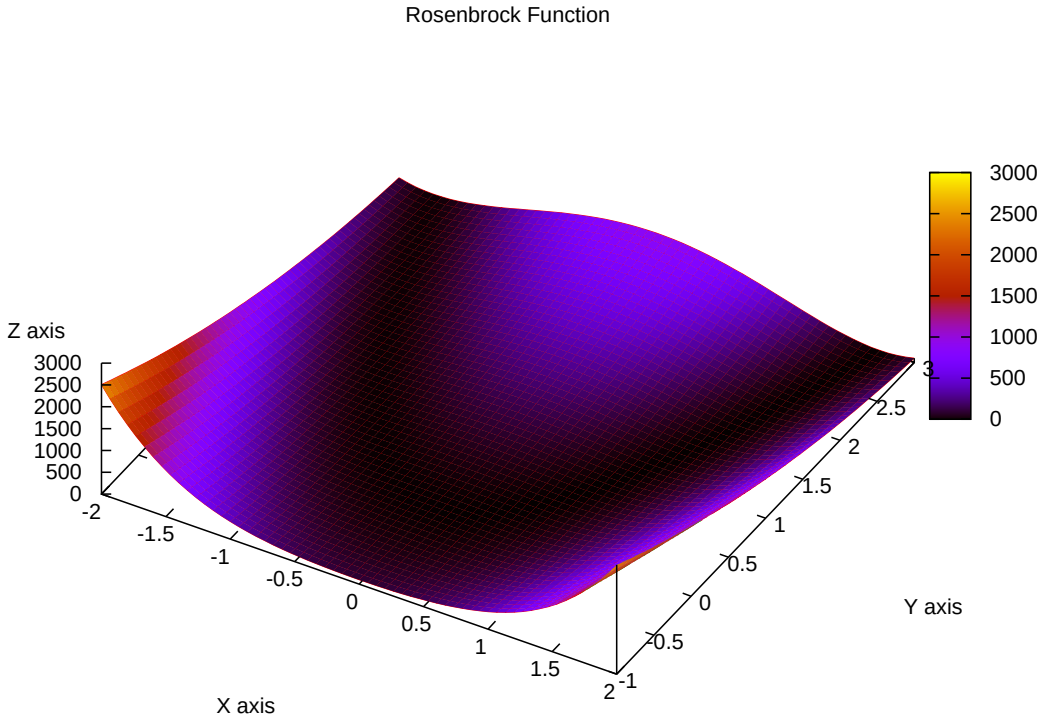

Figure 3.3: The Rosenbrock function.

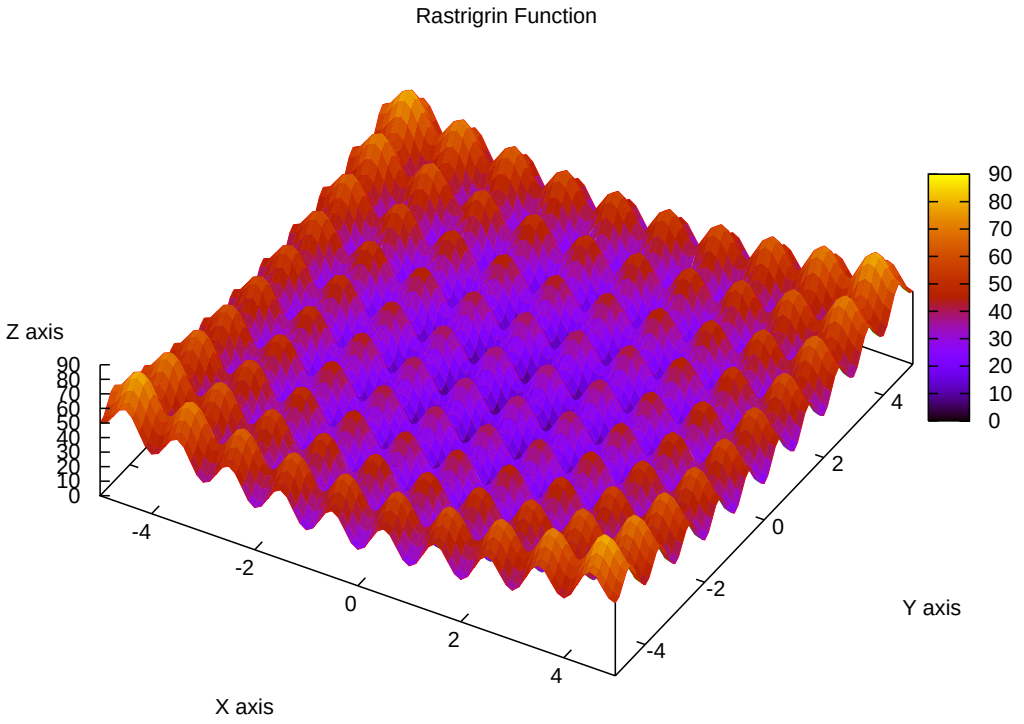

Figure 3.4: The Rastrigrin function.

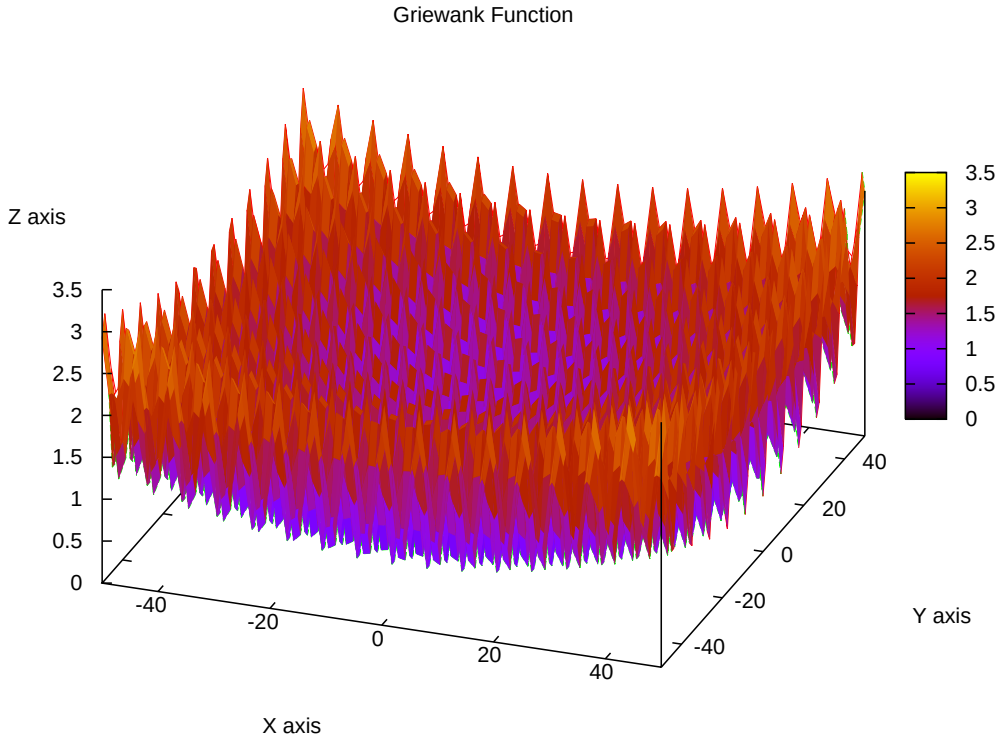

Figure 3.5: The Griewank function.

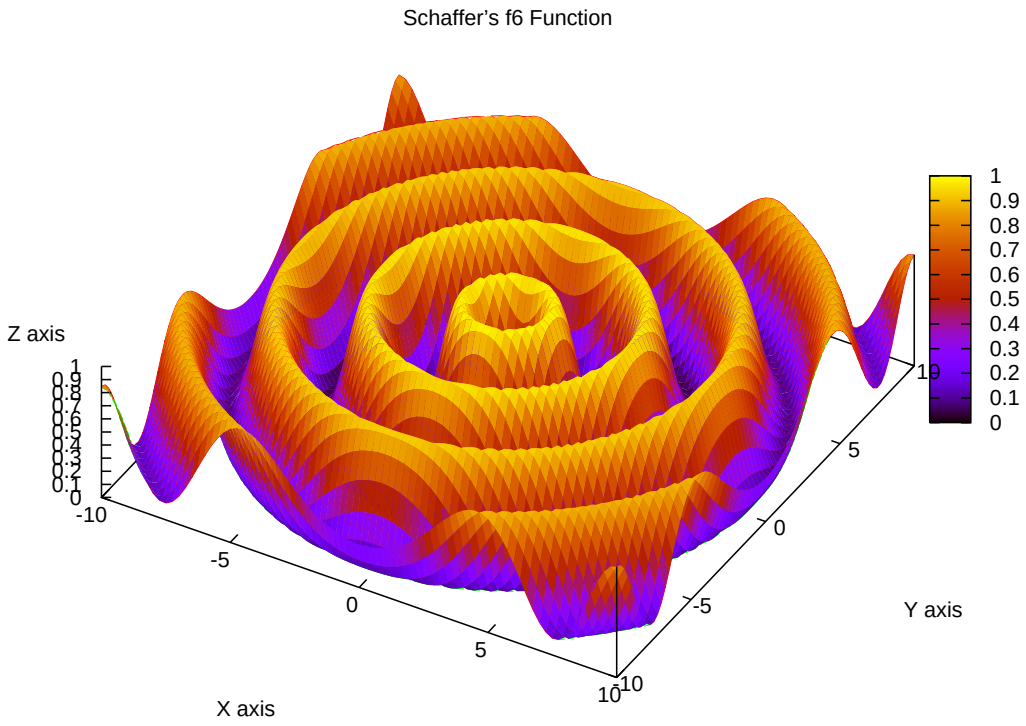

Figure 3.6: The Schaffer's f6 function.

On the unimodal Sphere function, both populations of 40 and 100 achieved the stopping criterion consistently, on all problems. However, in terms of the average number of function evaluations required to reach the stopping criterion, the population of 40 achieved the stopping criterion about twice as fast, and on all problems the population of 40 achieved the stopping criterion faster. We conclude that the population of 40 performed better on the unimodal Sphere problem.

On the unimodal Rosenbrock function, the population of 40 achieved more stopping criteria on the 10, 20, 30, and 50-dimensional problems. On the 100-dimensional problem, the stopping criterion was not ever achieved. However, the population of 40 achieved a better average minimization result. We conclude that the population of 40 performed better on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in 10-dimensions, the population of 40 achieved one more stopping criteria than the population of 100. The population of 100 achieved a better average minimization result on all problems. We conclude that population of 100 performs better than the population of 40 on the multimodal Rastrigrin function.

On the multimodal Griewank function, we find that in 10 dimensions, the same number of stopping criteria are achieved. However, in 20, 30, 50, and 100-dimensions, the population of 100 achieves more stopping criteria. Furthermore, on all problems, the population of 100 achieves a better average minimization result. We conclude that the population of 100 performs better than the population of 40 on the Griewank function.

On the 2-dimensional Schaffer's f6 function, we find that the population of 100 achieved more stopping criteria than the population of 40.

Overall the results are mixed, with the population of 100 achieving the better results on the multimodal function and the population of 40 achieving the better results on the unimodal functions.

For this thesis, we applied a population of 40, since it offers both good speed on the unimodal Sphere problem, the best results on the unimodal Rosenbrock function, and good results on the multimodal Rastrigrin, Griewank, and Schaffer's f6 functions. A population of 40 is also frequently applied in the literature [43, 4, 5].

3.4.3 Comparison of PSO Algorithms from the Literature

In this section, we take a brief look at some of the PSO algorithms that are seen in the literature. These algorithms will serve as benchmarks for comparisons in this chapter and the next.

The algorithms are as follows.

- The standard PSO (STD) (Section 2.5.2), with the standard Clerc settings and the global best neighborhood.
- The time-varying inertial weight PSO (TVW) (Section 2.5.3), with the following settings: w varies from 0.9 to 0.4, $c_1 = c_2 = 2.0$, and the global best neighborhood.
- The time-varying accelerating coefficients PSO (TVW-TVA) (Section 2.5.5), with the following settings: w varies from 0.9 to 0.4, c_1 varies from 2.5 to 0.5, c_2 varies from 0.5 to 2.5, and the global best neighborhood.
- The local best neighborhood standard PSO (LBEST) (Section 2.5.3), the standard PSO applying the LBEST neighborhood. Two neighbors plus self are taken as the neighborhood.
- The von Neumann neighborhood standard PSO (VonNeu) (Section 2.5.4), the standard PSO applying the von Neumann neighborhood. The particles are layed out on a 8 by 5 toroidal grid and 4 neighbors plus self are taken as the neighborhood.

Table 3.4 and Figure 3.8, give the simulation results. Let us compare each of the algorithms to the standard PSO algorithm to get a feel for its advantages (or disadvantages).

Comparing TVW PSO to STD PSO

On the unimodal Sphere function, both TVW PSO and STD PSO achieved the stopping criterion consistently. However in terms of the number of function evaluations required, the TVW PSO required about 30 times more function evaluations for the 10-dimensional Sphere problem, and over 2 times more function evaluations on the 100-dimensional Sphere problem. We conclude that the STD algorithm performs better than the TVW PSO algorithm on the unimodal Sphere problem.

On the unimodal Rosenbrock function, STD PSO achieved more stopping criteria on the 10, 20, 30, and 50-dimensional problems. On the 100-dimensional problem, the stopping criterion was not ever achieved. However, STD PSO achieved a better average minimization result. We conclude that the STD PSO algorithm performs better than the TVW PSO algorithm on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function in 10-dimensions, TVW PSO achieved more stopping criteria than STD PSO. On the higher dimensional problems, the stopping criterion was never achieved. However, in terms of average minimization result achieved, we find that TVW PSO consistently achieves the better

Table 3.3: Effects of varying the PSO population size. The upper number is the number of successes at reaching the stopping criterion (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	p10	p20	p40	p60	p80	p100
$f_1(x)$	10	**50** (1531) goal (—)	**50** (2442) goal (—)	**50** (4253) goal (—)	**50** (5830) goal (—)	**50** (7522) goal (—)	**50** (9076) goal (—)
	20	**50** (5436) goal (—)	**50** (5103) goal (—)	**50** (7768) goal (—)	**50** (10658) goal (—)	**50** (13285) goal (—)	**50** (16014) goal (—)
	30	42 (21154) 0.0434 (0.1996)	**50** (10015) goal (—)	**50** (12594) goal (—)	**50** (16384) goal (—)	**50** (20067) goal (—)	**50** (23558) goal (—)
	50	0 (—) 120.33 (287.38)	**50** (34679) goal (—)	**50** (29458) goal (—)	**50** (33326) goal (—)	**50** (38243) goal (—)	**50** (44930) goal (—)
	100	0 (—) 3641 (2451)	46 (189743) 1.33 (7.58)	**50** (148263) goal (—)	**50** (152737) goal (—)	**50** (147443) goal (—)	**50** (167352) goal (—)
$f_2(x)$	10	24 (122008) 1.75 (2.26)	34 (91034) 0.9066 (1.64)	**40** (142406) 0.6472 (1.46)	36 (194142) 0.9692 (1.70)	36 (215384) 0.4188 (1.35)	35 (266891) 0.7556 (1.58)
	20	3 (73780) 23.10 (37.18)	27 (190885) 1.76 (2.16)	**33** (252238) 2.46 (9.58)	17 (317428) 2.41 (9.64)	8 (266180) 3.69 (16.43)	2 (306450) 5.22 (15.97)
	30	0 (—) 316.64 (1165)	**23** (201098) 2.07 (2.47)	11 (279047) 4.52 (11.86)	8 (322905) 8.76 (17.94)	1 (316000) 12.93 (22.14)	1 (372500) 21.01 (30.64)
	50	0 (—) $>10^4$ ($>10^4$)	**15** (281373) 137.37 (893.82)	5 (302240) 28.93 (38.40)	0 (—) 44.58 (42.16)	0 (—) 54.97 (39.16)	0 (—) 49.17 (37.62)
	100	**0** (—) $>10^4$ ($>10^4$)	**0** (—) $>10^4$ ($>10^4$)	**0** (—) 162.77 (89.21)	**0** (—) 187.24 (70.83)	**0** (—) 190.04 (80.37)	**0** (—) 209.06 (95.06)
$f_3(x)$	10	0 (—) 18.80 (10.18)	0 (—) 10.15 (5.05)	**1** (16280) 5.67 (3.11)	**1** (16500) 4.56 (2.66)	**1** (23840) 4.06 (2.21)	0 (—) 3.20 (1.20)
	20	**0** (—) 72.87 (25.58)	**0** (—) 52.87 (19.30)	**0** (—) 36.57 (15.06)	**0** (—) 28.81 (10.06)	**0** (—) 24.10 (8.08)	**0** (—) 22.76 (6.07)
	30	**0** (—) 151.19 (49.90)	**0** (—) 113.33 (33.68)	**0** (—) 91.10 (27.40)	**0** (—) 67.12 (19.62)	**0** (—) 67.14 (19.41)	**0** (—) 62.22 (18.84)
	50	**0** (—) 371.03 (69.78)	**0** (—) 273.61 (62.63)	**0** (—) 226.89 (52.31)	**0** (—) 199.05 (43.50)	**0** (—) 193.08 (37.85)	**0** (—) 175.07 (34.31)
	100	**0** (—) 1055 (126.23)	**0** (—) 872.44 (122.46)	**0** (—) 771.75 (160.70)	**0** (—) 708.37 (129.40)	**0** (—) 656.99 (104.40)	**0** (—) 612.87 (123.44)
$f_4(x)$	10	**1** (2220) 0.1095 (0.0582)	0 (—) 0.0820 (0.0322)	**1** (8040) 0.0762 (0.0413)	0 (—) 0.0754 (0.0387)	**1** (48240) 0.0636 (0.0292)	**1** (32100) 0.0550 (0.0262)
	20	2 (4440) 0.2334 (0.4014)	11 (6495) 0.0399 (0.0385)	17 (9584) 0.0248 (0.0195)	15 (12852) 0.0250 (0.0228)	16 (23980) 0.0223 (0.0174)	**21** (21752) 0.0217 (0.0176)
	30	3 (19303) 1.27 (2.96)	14 (10581) 0.1154 (0.2533)	24 (14020) 0.0231 (0.0184)	26 (17979) 0.0167 (0.0121)	27 (22347) 0.0193 (0.0185)	**28** (27375) 0.0142 (0.0086)
	50	0 (—) 2.18 (2.77)	6 (42743) 0.4300 (1.09)	19 (28341) 0.0572 (0.0793)	23 (34067) 0.3064 (1.83)	25 (39629) 0.0401 (0.0542)	**30** (46023) 0.0309 (0.0549)
	100	0 (—) 34.89 (23.90)	0 (—) 1.11 (1.56)	6 (148520) 0.7767 (1.94)	10 (148350) 0.5340 (1.28)	**18** (148436) 0.5028 (1.57)	16 (155069) 0.1824 (0.2937)
$f_5(x)$	2	22 (17500) 0.0054 (0.0048)	27 (20994) 0.0045 (0.0048)	43 (14299) 0.0014 (0.0034)	45 (21420) 0.0010 (0.0029)	**49** (20762) 0.0002 (0.0014)	**49** (18416) 0.0002 (0.0014)

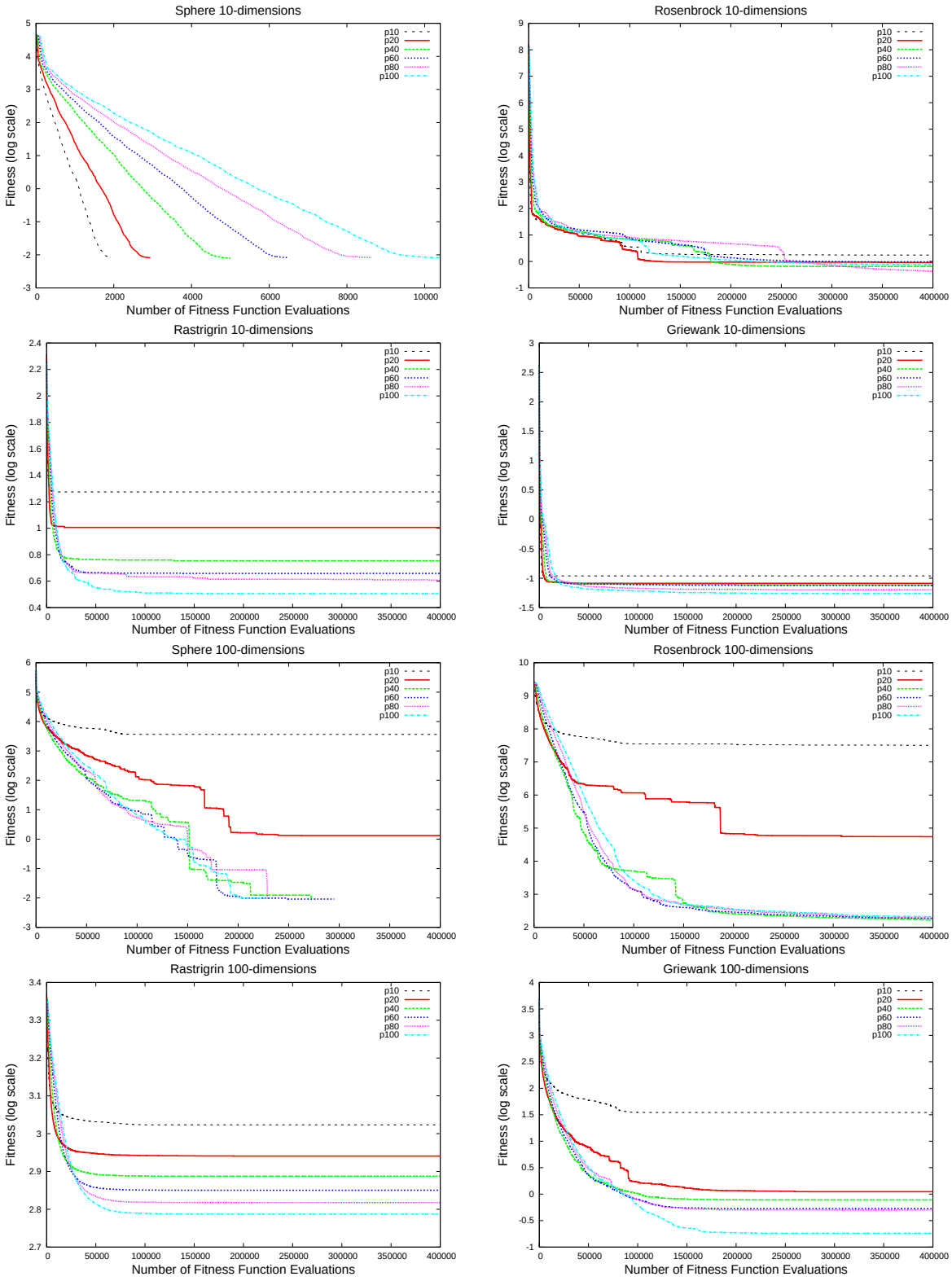

Figure 3.7: Effects of varying the PSO population size.

result. We conclude that TVW PSO performs better than the STD PSO on the multimodal Rastrigrin function.

On the multimodal Griewank function, we find that in 10 and 20-dimensions, the STD PSO algorithm achieved more stopping criteria. While in 30, 50, and 100 dimensions, TVW PSO achieves more stopping criteria. The results are mixed, and we cannot conclude that one algorithm performed better than the other.

On the 2-dimensional Schaffer's f6 function, we find that TVW achieved more stopping criteria than STD PSO.

Overall, we conclude that TVW performs better than STD PSO on the multimodal benchmark functions, and that STD PSO performs better on the unimodal benchmark functions.

Comparing TVW-TVA PSO to STD PSO

On the unimodal Sphere function, both TVW-TVA PSO and STD PSO achieved the stopping criterion consistently, on the problems of less than 100 dimensions. However in terms of the number of function evaluations required, this TVW-TVA PSO required about 14 times more function evaluations for the 10-dimensional Sphere problem, and over 3 times more function evaluations on the 50-dimensional Sphere problem. On the 100-dimensional Sphere problem, STD PSO achieved the stopping criterion consistently, whereas TVW-TVA failed. We conclude that the STD PSO algorithm performs better than the TVW-TVA PSO algorithm on the unimodal Sphere function.

On the unimodal Rosenbrock function, STD PSO achieved more stopping criteria on the 10, 20, 30, and 50-dimensional problems. On the 100-dimensional problem, the stopping criterion was not ever achieved. However, STD PSO achieved a better average minimization result. We conclude that the STD PSO algorithm performs better than the TVW-TVA PSO algorithm on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in 10-dimensions, TVW-TVA PSO achieved more stopping criteria than STD PSO. On the higher dimensional problems, the stopping criterion was never achieved. However, in terms of average minimization result achieved, we find that TVW-TVA PSO consistently achieves the better result. We conclude that TVW-TVA PSO performs better than the STD PSO on the multimodal Rastrigrin function.

On the multimodal Griewank function, we find that in 10 and 50-dimensions, the TVW-TVA PSO algorithm achieved more stopping criteria. While in 20, 50, and 100 dimensions, STD PSO achieves more stopping criteria. The results are mixed, and we cannot conclude that one algorithm performed better than the other.

On the 2-dimensional Schaffer's f6 function, we find that TVW-TVA achieved more stopping criteria than STD PSO.

Overall, we conclude that TVW-TVA performs better than STD PSO on the multimodal benchmark functions, and that STD PSO performs better on the unimodal benchmark functions.

Comparing LBEST PSO to STD PSO

On the unimodal Sphere function, both LBEST PSO and STD PSO achieved the stopping criterion consistently. Looking at the number of function evaluations required, we find that STD PSO required fewer function evaluations on the 10, 20, 30, and 50-dimensional problems. Whereas the LBEST PSO

required fewer function evaluations on the 100-dimensional problem. We conclude that the STD PSO algorithm performs slightly better than the LBEST PSO algorithm on the unimodal Rosenbrock function.

As an aside, we note that the 100-dimensional Sphere problem is interesting in that the LBEST PSO (and VonNeu PSO) required fewer function evaluations when compared to STD PSO. Since the Sphere problem is unimodal, we might expect that the instant knowledge of the best position offered by global best PSO would help speed convergence to the global minimum and would perform better than the LBEST PSO. We hypothesize that although the 100-dimensional Sphere problem is unimodal, premature convergence is occurring (in a large number of the 100 dimensions), making it difficult for progress to be made by the global best PSO. Thus LBEST PSO (and VonNeu PSO) outperforms STD PSO in the 100-dimensional Sphere problem.

On the unimodal Rosenbrock function, LBEST PSO achieved more stopping criteria on the 10-dimensional problem. STD PSO achieved more stopping criteria on the 20, 30, and 50-dimensional problems. On the 100-dimensional problem, the stopping criterion was not ever achieved. However, STD PSO achieved a better average minimization result. We conclude that the STD PSO algorithm performs better than the LBEST PSO algorithm on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in 10-dimensions, STD PSO achieved one stopping criterion and LBEST none. On the higher dimensional problems, the stopping criterion was never achieved. In terms of average minimization result achieved, we find that LBEST PSO performed better on the 10, 20, and 30-dimensional problems. On the 50 and 100-dimensional problems STD PSO performed better. We conclude that the algorithms performed equivalently on the unimodal Rastrigrin function.

On the multimodal Griewank function, LBEST PSO always achieved more stopping criteria than the STD PSO. We conclude that the LBEST PSO algorithm performs better than the STD PSO algorithm on the multimodal Griewank function.

On the 2-dimensional Schaffer's f6 function, we find that LBEST achieved more stopping criterion than STD PSO. We conclude that the LBEST PSO algorithm performs slightly better than the STD PSO algorithm on the unimodal Rosenbrock function.

Overall, the results are mixed. We cannot conclude that the LBEST PSO algorithm performed better or worse than the STD PSO algorithm.

Comparing VonNeu PSO to STD PSO

On the unimodal Sphere function, both VonNeu PSO and STD PSO achieved the stopping criterion consistently, Looking at the number of function evaluations required, We find that STD PSO required fewer function evaluations on the 10, 20, 30, and 50-dimensional problems. Whereas the VonNeu PSO required fewer function evaluations on the 100-dimensional problem. It is difficult to conclude that one algorithm performed better than the other.

On the unimodal Rosenbrock function, VonNeu PSO achieved more stopping criteria on the 10-dimensional problem. STD PSO achieved more stopping criteria on the 20, 30, and 50-dimensional problems. On the 100-dimensional problem, the stopping criterion was not ever achieved. However, VonNeu PSO achieved a better average minimization result. We conclude that the STD PSO algorithm performs slightly better than the VonNeu PSO algorithm on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in 10-dimensions, STD PSO achieved one stopping criterion and the VonNeu PSO achieved three. On the higher dimensional problems, the stopping criterion was never achieved. In terms of average minimization result achieved, we find that VonNeu PSO performed

better on all problems. We conclude that the VonNeu PSO performed better than the STD PSO on the multimodal Rastrigrin function.

On the multimodal Griewank function, VonNeu PSO always achieved more stopping criteria than the STD PSO. We conclude that the VonNeu PSO algorithm performs better than the STD PSO algorithm on the multimodal Griewank function.

On the 2-dimensional Schaffer's f6 function, we find that VonNeu PSO achieved more stopping criteria than STD PSO.

Overall, we conclude that VonNeu PSO performs better than STD PSO on the multimodal benchmark functions, and that STD PSO performs better on the unimodal benchmark functions.

3.4.4 Velocity Based Reinitialization Applied to Standard PSO

Table 3.5 and Figure 3.9 give the results comparing the standard global (STD) PSO algorithm to the VBR-PSO algorithm. First, let us compare the the STD PSO to the VBR-PSO. Second, we will look at how to determine the VBR threshold setting, α.

Comparing VBR-PSO to STD PSO

We will use $\alpha = 10^{-4}$ for this comparison.

On the unimodal Sphere function, both VBR-PSO and STD PSO achieved the stopping criteria consistently. Looking at the number of function evaluations required, we find that the results are the same on the 10, 20, 30, and 50-dimensional problems. However, STD PSO required fewer function evaluations on the 100-dimensional problem. On problems of less than 50 dimensions, no reinitializations occur and the results are identical. The stagnation threshold was never reached and VBR-PSO gracefully degenerated to the standard PSO algorithm. On the 100-dimensional problem, reinitializations occur and interfere with the optimization, suggesting that for higher dimensional problems, α should decreased. Overall, the VBR-PSO and STD PSO behaved approximately equivalently on the unimodal Sphere function.

On the unimodal Rosenbrock function, both algorithms achieved the same number of stopping criteria on problems of all dimensions. On the 50-dimensional Rosenbrock function, VBR-PSO achieved a slightly better average minimization result, suggesting that a beneficial reinitialization occurred. Still, the results are essentially identical. We conclude that the algorithms performed equivalently on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in 10-dimensions, VBR-PSO achieved more stopping criteria than STD PSO. On the higher dimensional problems, the stopping criterion was never achieved. In terms of average minimization result achieved, we find that VBR PSO performed better on all problems. We conclude that the VBR PSO performed better than the STD PSO on the multimodal Rastrigrin function.

On the multimodal Griewank function VBR-PSO always achieved more stopping criteria than the STD PSO. We conclude that the VBR-PSO algorithm performs better than the STD PSO algorithm on the multimodal Griewank function.

On the 2-dimensional Schaffer's f6 function, we find that VBR-PSO achieved more stopping criteria than STD PSO.

Overall, we observe that the VBR-PSO algorithm has performed better than the STD PSO algorithm on the multimodal benchmark functions and equivalently on the unimodal benchmark functions.

Table 3.4: A comparison of the STD, TVW, TVW-TVA, LBEST, and VonNeu PSO algorithms in terms of the number of successes at reaching the stopping criterion and the average optimization result achieved out of 50 trials. The upper number is the number of successes (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	STD	TVW	TVW-TVA	LBEST	VonNeu
$f_1(x)$	10	**50** (4253)	**50** (129928)	**50** (59706)	**50** (7734)	**50** (6149)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (7768)	**50** (180097)	**50** (76898)	**50** (16234)	**50** (12527)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (12594)	**50** (211759)	**50** (87059)	**50** (25090)	**50** (19022)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (29458)	**50** (257516)	**50** (102754)	**50** (44141)	**50** (33174)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (148263)	**50** (337687)	13 (190646)	**50** (94395)	**50** (74669)
		goal (—)	goal (—)	0.2819 (0.6015)	goal (—)	goal (—)
$f_2(x)$	10	40 (142406)	2 (198820)	23 (269431)	**44** (299738)	42 (220866)
		0.6472 (1.46)	4.00 (12.06)	0.4550 (0.9590)	0.1350 (0.5999)	0.4899 (1.29)
	20	**33** (252238)	1 (392440)	4 (339440)	5 (295032)	9 (301782)
		2.46 (9.58)	18.20 (30.25)	9.19 (14.27)	1.81 (3.01)	2.37 (3.78)
	30	**11** (279047)	1 (366920)	0 (—)	2 (243980)	1 (384320)
		4.52 (11.86)	49.73 (58.43)	32.38 (33.59)	11.63 (21.21)	8.55 (13.50)
	50	**5** (302240)	0 (—)	0 (—)	0 (—)	0 (—)
		28.93 (38.40)	112.00 (82.96)	91.00 (62.61)	48.03 (41.65)	71.95 (57.64)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		162.77 (89.21)	438.50 (166.50)	899.97 (590.11)	180.73 (72.58)	160.43 (85.78)
$f_3(x)$	10	1 (16280)	**18** (205980)	13 (119446)	0 (—)	3 (223467)
		5.67 (3.11)	0.9780 (0.9194)	1.06 (0.8513)	3.50 (1.78)	2.57 (1.59)
	20	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		36.57 (15.06)	8.34 (2.79)	10.19 (3.44)	31.78 (9.81)	23.88 (7.27)
	30	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		91.10 (27.40)	23.94 (6.44)	28.04 (6.46)	87.94 (17.00)	60.17 (17.59)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		226.89 (52.31)	68.13 (14.00)	91.22 (22.21)	243.17 (35.26)	171.21 (35.00)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		771.75 (160.70)	243.05 (34.16)	398.24 (56.10)	805.31 (98.05)	596.62 (91.29)
$f_4(x)$	10	1 (8040)	0 (—)	6 (123367)	**9** (64338)	6 (93347)
		0.0762 (0.0413)	0.0543 (0.0220)	0.0409 (0.0233)	0.0264 (0.0165)	0.0323 (0.0186)
	20	17 (9584)	9 (195756)	15 (97317)	**47** (24161)	38 (22824)
		0.0248 (0.0195)	0.0339 (0.0262)	0.0238 (0.0177)	0.0102 (0.0033)	0.0116 (0.0046)
	30	24 (14020)	26 (218557)	23 (91852)	**48** (35712)	40 (21348)
		0.0231 (0.0184)	0.0163 (0.0110)	0.0231 (0.0212)	0.0099 (0.0015)	0.0122 (0.0065)
	50	19 (28341)	32 (261756)	27 (104879)	**48** (47280)	40 (35562)
		0.0572 (0.0793)	0.0145 (0.0108)	0.0218 (0.0216)	0.0099 (0.0012)	0.0111 (0.0037)
	100	6 (148520)	33 (338880)	5 (192168)	**50** (94488)	40 (73983)
		0.7767 (1.94)	0.0154 (0.0116)	0.1538 (0.1942)	goal (—)	0.0164 (0.0230)
$f_5(x)$	2	43 (14299)	**50** (69834)	**50** (41910)	49 (31336)	**50** (18290)
		0.0014 (0.0034)	goal (—)	goal (—)	0.0002 (0.0014)	goal (—)

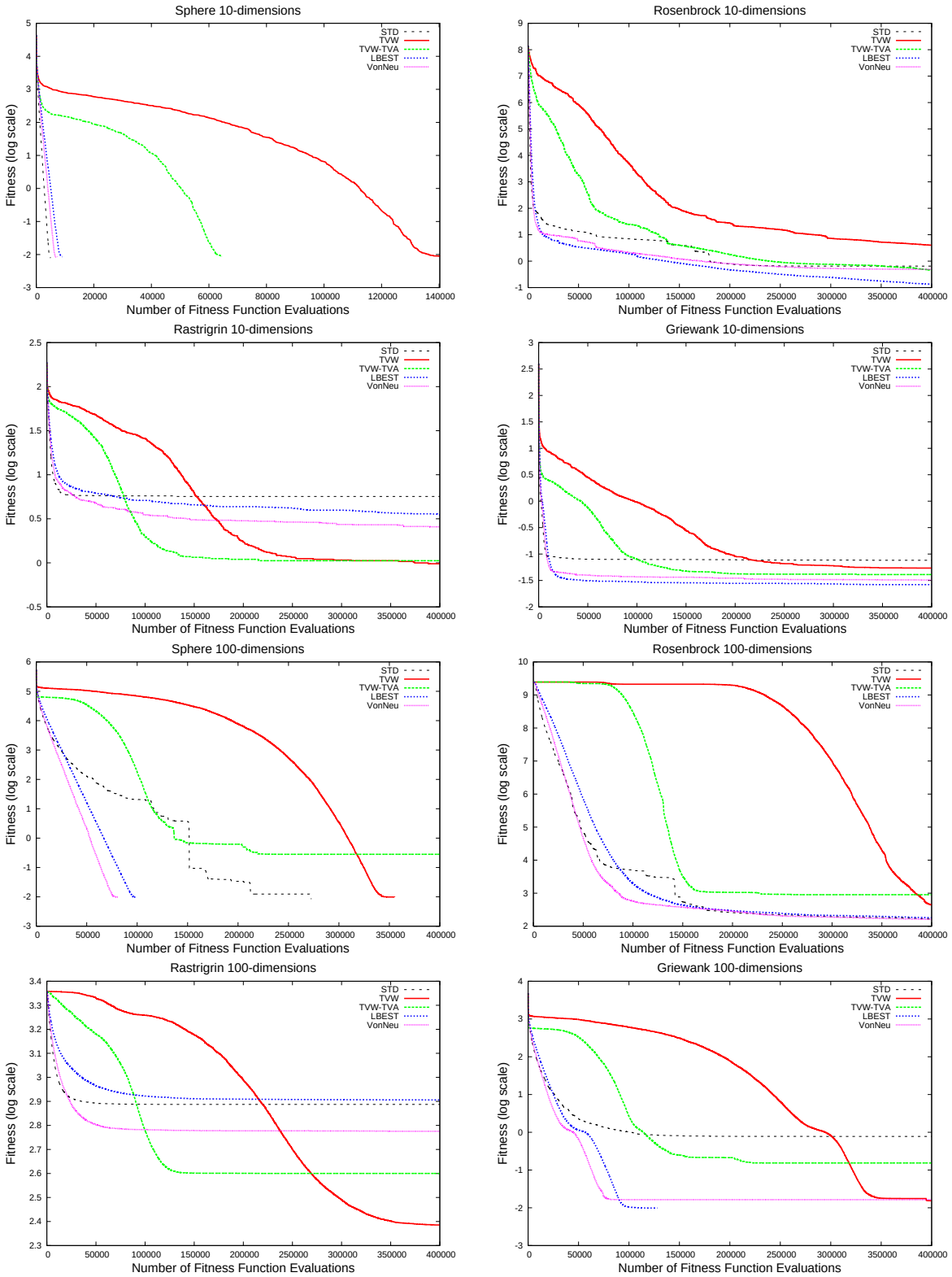

Figure 3.8: A comparison of the STD, TVW, TVW-TVA, LBEST, and VonNeu PSO algorithms.

Table 3.5: A comparison of the STD and VBR-PSO algorithms in terms of the number of successes at reaching the stopping criterion and the average optimization result achieved out of 50 trials. The upper number is the number of successes (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	STD	$\alpha = 10^{-2}$	$\alpha = 10^{-3}$	$\alpha = 10^{-4}$	$\alpha = 10^{-5}$
$f_1(x)$	10	**50** (4253)	**50** (4253)	**50** (4253)	**50** (4253)	**50** (4253)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (7768)	**50** (7768)	**50** (7768)	**50** (7768)	**50** (7768)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (12594)	**50** (12594)	**50** (12594)	**50** (12594)	**50** (12594)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (29458)	**50** (32062)	**50** (29458)	**50** (29458)	**50** (29458)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (148263)	21 (204171)	43 (153728)	**50** (152207)	**50** (150298)
		goal (—)	0.0265 (0.0345)	0.0118 (0.0082)	goal (—)	goal (—)
$f_2(x)$	10	40 (142406)	**44** (127355)	40 (142406)	40 (142406)	40 (142406)
		0.6472 (1.46)	0.3286 (1.08)	0.6472 (1.46)	0.6472 (1.46)	0.6472 (1.46)
	20	33 (252238)	24 (284492)	**36** (270829)	33 (252238)	33 (252238)
		2.46 (9.58)	1.45 (2.04)	2.28 (9.59)	2.46 (9.58)	2.46 (9.58)
	30	11 (279047)	3 (336693)	**13** (307745)	11 (279047)	11 (279047)
		4.52 (11.86)	9.22 (17.21)	6.28 (17.56)	4.52 (11.86)	4.52 (11.86)
	50	**5** (302240)	0 (—)	0 (—)	**5** (274136)	1 (301320)
		28.93 (38.40)	46.19 (27.00)	30.88 (35.83)	27.58 (31.21)	27.88 (36.02)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		162.77 (89.21)	230.54 (75.73)	167.96 (64.80)	162.77 (89.21)	162.77 (89.21)
$f_3(x)$	10	1 (16280)	**4** (247990)	**4** (204840)	2 (299040)	2 (35040)
		5.67 (3.11)	1.49 (0.6951)	1.61 (0.8163)	1.83 (0.8284)	1.79 (0.7701)
	20	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		36.57 (15.06)	15.15 (3.06)	15.36 (3.49)	16.42 (4.01)	16.31 (3.74)
	30	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		91.10 (27.40)	45.70 (5.88)	46.98 (7.46)	46.90 (7.68)	48.50 (7.99)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		226.89 (52.31)	140.79 (19.39)	153.33 (20.77)	150.43 (21.12)	156.49 (23.05)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		771.75 (160.70)	603.02 (59.89)	613.08 (70.29)	626.99 (85.96)	649.24 (90.05)
$f_4(x)$	10	1 (8040)	**8** (183165)	5 (268712)	6 (230767)	7 (128097)
		0.0762 (0.0413)	0.0221 (0.0108)	0.0226 (0.0094)	0.0222 (0.0099)	0.0230 (0.0096)
	20	17 (9584)	**50** (34346)	**50** (36107)	**50** (47549)	**50** (48226)
		0.0248 (0.0195)	goal (—)	goal (—)	goal (—)	goal (—)
	30	24 (14020)	**50** (41761)	**50** (37742)	**50** (41771)	**50** (39621)
		0.0231 (0.0184)	goal (—)	goal (—)	goal (—)	goal (—)
	50	19 (28341)	**46** (104663)	**46** (121523)	42 (95713)	45 (137500)
		0.0572 (0.0793)	0.0100 (0.0016)	0.0103 (0.0030)	0.0126 (0.0078)	0.0117 (0.0079)
	100	6 (148520)	11 (223047)	**15** (176563)	7 (268640)	9 (154138)
		0.7767 (1.94)	0.1731 (0.2539)	0.1844 (0.2680)	0.1690 (0.2380)	0.2192 (0.2693)
$f_5(x)$	2	43 (14299)	**50** (20490)	**50** (16398)	**50** (39930)	**50** (27610)
		0.0014 (0.0034)	goal (—)	goal (—)	goal (—)	goal (—)

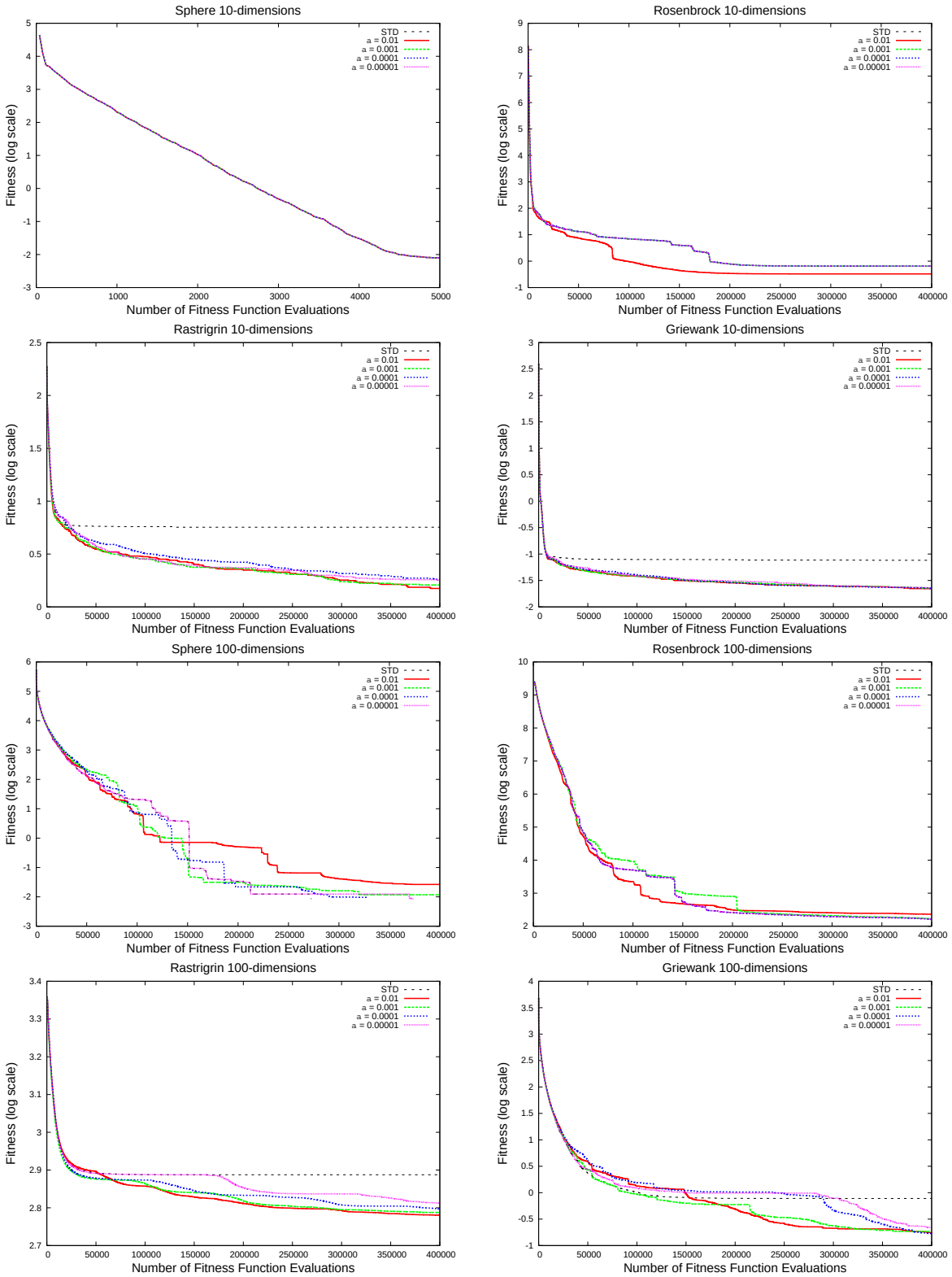

Figure 3.9: A comparison of the STD and VBR-PSO algorithms.

Determining the Appropriate Threshold, α, for VBR

Since the VBR-PSO algorithm becomes the standard PSO algorithm as α approaches 0, it is safe to choose a small value of α. Table 3.5 suggests that $\alpha = 10^{-4}$ or $\alpha = 10^{-5}$ are good settings for these benchmark problems.

We hypothesize that what is important in the VBR-PSO algorithm is that there is a "give up, record the global best, and reinitialize" point during the optimization. The VBR threshold setting, α, determines when this occurs and if it is set to reasonably conservative values, the VBR-PSO algorithm yields improved results over the standard PSO algorithm. Indeed, as $\alpha \to 0$ reinitialization becomes rare and VBR-PSO approaches PSO.

However, setting α too conservatively will not yield the benefits of VBR-PSO. We offer the following guidelines for setting α. For a particular PSO application, choose α based on how close it is desired to get to the actual local minimum of the problem. If one-half of the swarm particle's velocity vectors fit in an n-dimensional ball of radius α then VBR will consider the swarm stagnate. Keep in mind that for most real-life problems there will be many local minimums. PSO will be finding one of them, generally not the global minimum. Rather than get close to a particular local minimum, it might actually be better to reinitialize and find a better location further away from a different local minimum.

3.4.5 VBR Applied to the LBEST and the von Neumann PSOs

In the previous section, VBR was observed to improve the performance of the standard global best PSO algorithm, in particular for multimodal functions. The neighborhood methods, LBEST and von Neumann, are known to improve on the standard global best neighborhood. In this section, we apply VBR to these neighborhoods to show that VBR can also benefit them. The results are presented in Table 3.6 and Figure 3.10. A stagnation threshold, $\alpha = 10^{-4}$, was used for this comparison. The columns VBR-L and VBR-V indicate the VBR-enhanced LBEST and von Neumann PSO algorithms.

VBR enhanced LBEST PSO

Both algorithms achieve the stopping criterion consistently for the Sphere function. Furthermore, the number of function evaluations required to achieve the stopping criterion is also identical, indicating that the stagnation threshold either was not reached or neither beneficial nor harmful. We conclude that VBR-L PSO and LBEST PSO performed equivalently on the unimodal Sphere function.

On the unimodal Rosenbrock function, the VBR-L PSO achieved one more stopping criterion on the 10-dimensional problem, the stagnation threshold was reached and reinitialization was beneficial. On the higher dimensional Rosenbrock problems, the results are exactly the same, the stagnation criterion either not being reached or reinitialization being neither beneficial nor harmful. We conclude that VBR-L PSO and LBEST PSO performed equivalently on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, we find that stopping criteria was never achieved. However, in terms of average minimization result achieved, we find that the VBR-L PSO performed better on problems of all dimensions. We conclude that VBR-L PSO performed better on the multimodal Rastrigrin function.

On the Griewank function, the VBR-L PSO achieved more stopping criteria on the 10, 20, 30, and 50-dimension problems. On the 100-dimension problem both VBR-L and LBEST achieved identical results. We conclude that VBR-L PSO performed better on the multimodal Griewank function.

Table 3.6: A comparison of VBR-PSO with different neighborhoods in terms of the number of successes at reaching the stopping criterion and the average optimization result achieved out of 50 trials. The upper number is the number of successes (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	LBEST	VBR-L	VonNeu	VBR-V
$f_1(x)$	10	**50** (7734)	**50** (7734)	**50** (6149)	**50** (6149)
		goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (16234)	**50** (16234)	**50** (12527)	**50** (12527)
		goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (25090)	**50** (25090)	**50** (19022)	**50** (19022)
		goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (44141)	**50** (44141)	**50** (33174)	**50** (33174)
		goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (94395)	**50** (94395)	**50** (74669)	**50** (74669)
		goal (—)	goal (—)	goal (—)	goal (—)
$f_2(x)$	10	44 (299738)	**45** (283603)	42 (220866)	42 (220866)
		0.1350 (0.5999)	0.1787 (0.7842)	0.4899 (1.29)	0.4899 (1.29)
	20	5 (295032)	5 (295032)	**9** (301782)	**9** (301782)
		1.81 (3.01)	1.81 (3.01)	2.37 (3.78)	2.37 (3.78)
	30	**2** (243980)	**2** (243980)	1 (384320)	1 (384320)
		11.63 (21.21)	11.63 (21.21)	8.55 (13.50)	8.55 (13.50)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		48.03 (41.65)	48.03 (41.65)	71.95 (57.64)	71.95 (57.64)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		180.73 (72.58)	180.73 (72.58)	160.43 (85.78)	160.43 (85.78)
$f_3(x)$	10	0 (—)	0 (—)	3 (223467)	**6** (164567)
		3.50 (1.78)	3.24 (1.04)	2.57 (1.59)	1.49 (0.8488)
	20	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		31.78 (9.81)	24.32 (4.84)	23.88 (7.27)	15.47 (3.62)
	30	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		87.94 (17.00)	66.43 (11.57)	60.17 (17.59)	41.68 (7.69)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		243.17 (35.26)	204.54 (32.66)	171.21 (35.00)	133.23 (16.72)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		805.31 (98.05)	768.65 (93.26)	596.62 (91.29)	524.73 (60.43)
$f_4(x)$	10	9 (64338)	**41** (160759)	6 (93347)	29 (193705)
		0.0264 (0.0165)	0.0106 (0.0020)	0.0323 (0.0186)	0.0122 (0.0046)
	20	47 (24161)	**50** (27366)	38 (22824)	**50** (35251)
		0.0102 (0.0033)	goal (—)	0.0116 (0.0046)	goal (—)
	30	48 (35712)	**50** (32896)	40 (21348)	**50** (31758)
		0.0099 (0.0015)	goal (—)	0.0122 (0.0065)	goal (—)
	50	48 (47280)	**50** (48342)	40 (35562)	**50** (41796)
		0.0099 (0.0012)	goal (—)	0.0111 (0.0037)	goal (—)
	100	**50** (94488)	**50** (94488)	40 (73983)	49 (100238)
		goal (—)	goal (—)	0.0164 (0.0230)	0.0099 (0.0008)
$f_5(x)$	2	49 (31336)	**50** (20295)	**50** (18290)	**50** (15145)
		0.0002 (0.0014)	goal (—)	goal (—)	goal (—)

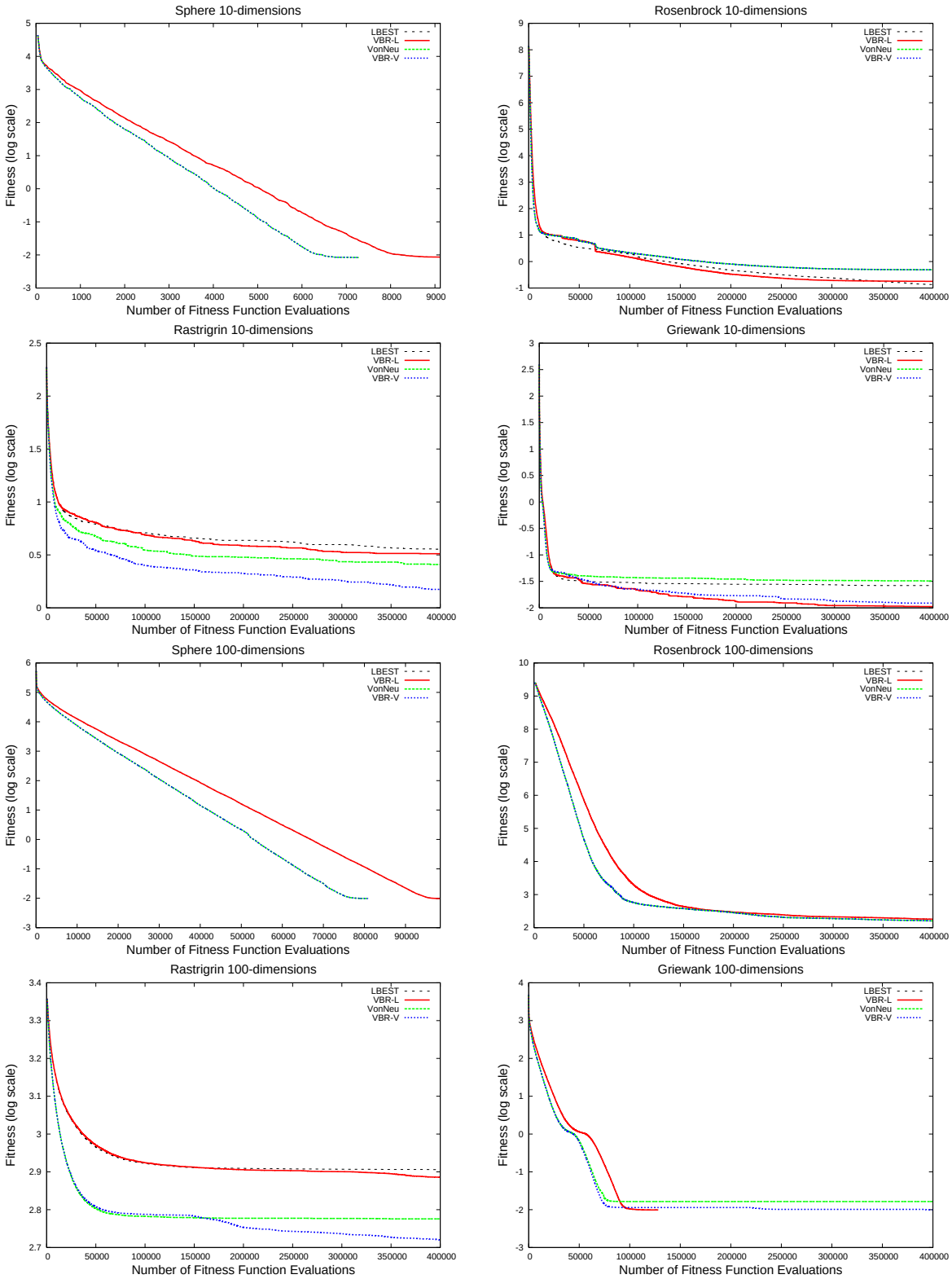

Figure 3.10: A comparison of VBR-PSO with different neighborhoods.

On the multimodal 2-dimensional Schaffer's f6 function, the results are close with VBR-L achieving one more stopping criteria. Also, in terms of the number of function evaluations required to achieve the stopping criteria, the VBR-L PSO achieved the stopping criteria faster. We conclude that VBR-L PSO performed slightly better on the multimodal Schaffer's f6 function.

Overall, we conclude that the VBR-L PSO algorithm yields improved results over the LBEST PSO algorithm for the multimodal benchmark functions and equivalent results on the unimodal benchmark functions.

VBR enhanced von Neumann PSO

Both algorithms achieve the stopping criterion consistently for the Sphere function. Furthermore, the number of function evaluations required to achieve the stopping criterion is also identical, indicating that the stagnation threshold either was not reached or neither beneficial or harmful. We conclude that VBR-V PSO and VonNeu PSO performed equivalently on the unimodal Sphere function.

On the unimodal Rosenbrock function, as in the case of the Sphere function, the results are identical. We conclude that VBR-V PSO and VonNeu PSO performed equivalently on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, we find VBR-V achieved more stopping criteria on the 10-dimensional problem. On the higher dimensional problems, the stopping criterion was never achieved. However, in terms of average minimization result achieved, we find that the VBR-V PSO performed better on problems of all dimensions. We conclude that VBR-V PSO performed better on the multimodal Rastrigrin function.

On the Griewank function, the VBR-V PSO achieved more stopping criteria on all the problems. We conclude that VBR-V PSO performed better on the multimodal Griewank function.

On the multimodal 2-dimensional Schaffer's f6 function, both algorithms achieved the stopping criterion consistently. In terms of the number of function evaluations required to achieve the stopping criteria, the VBR-V PSO achieved the stopping criteria faster. We conclude that VBR-V PSO performed slightly better on the multimodal Schaffer's f6 function.

Overall, we conclude that the VBR-V PSO algorithm yields improved results over the von Neumann PSO algorithm for the multimodal benchmark functions and equivalent results on the unimodal benchmark functions.

3.4.6 Conclusion

We proposed a new method, velocity-based reinitialization (VBR), to enhance the search capabilities of PSO algorithms. VBR is different from other reinitialization methods in that it uses a novel median-velocity-based method to determine when reinitialization should occur. VBR helps PSO algorithms achieve a good balance between exploitation (local search) and exploration (global search). With VBR the algorithm can first exploit, and then through reinitialization resume exploration.

VBR is simple to implement and has only one threshold parameter. Our experiments showed that VBR was not too sensitive to the threshold setting. The family of VBR enhanced PSO algorithms also gracefully approaches the standard PSO algorithm as the threshold setting approaches zero.

We applied VBR to enhance the global best, local best, and von Neumann neighborhood PSO algorithms. It was observed that the VBR enhanced PSO algorithms achieve improved results on the

multimodal benchmark functions, while yielding equivalent performance on the unimodal benchmark functions.

Chapter 4

Stop and Go Particle Swarm Optimization

4.1 Introduction

A key parameter to PSO optimization is the population size. In general, when computational resources are fixed, the smaller populations emphasize exploitation, and are better for unimodal functions; whereas larger populations emphasize exploration, and are needed for multimodal functions. Determining the population size is problem specific and is traditionally a matter of trial and error.

To address this problem, we propose the stop and go PSO (SG-PSO) algorithm [5], a swarm with a dynamic population size. We observe that in many practical problems there is a limit to the required accuracy of the optimization result. The SG-PSO algorithm takes advantage of this information to dynamically adjust the population size resulting in more effective use of function evaluations. More specifically, in SG-PSO, when a particle has approximately reached the required accuracy in domain space (accuracy being measured by closeness in domain space, not to the unknown optimum function value), it is stopped. The population size is effectively dynamically reduced. Stopped particles halt local search, allowing for the active particles to continue global search.

As an extension to SG-PSO, we propose the mixed SG-PSO (MSG-PSO) algorithm to facilitate the balance between global and local search. In the MSG-PSO algorithm, particles are given individual accuracy settings, resulting in a mixture of rough and precision searching particles. First, the rough searchers find promising locations in domain space and stop, then precision searchers perform further optimization.

Both SG-PSO and MSG-PSO algorithms are straightforward modifications to the standard PSO algorithm. Simulations show that SG-PSO enhances the ability of the standard PSO algorithm for the multimodal benchmark functions, while producing equivalent results on the unimodal benchmark functions. The MSG-PSO algorithm produces improvements over the SG-PSO algorithm on both multimodal and unimodal benchmark functions.

Since the neighborhood methods local best (LBEST) [28] and von Neumann (VonNeu) [34] are known to improve the standard PSO algorithm on multimodal function optimization tasks, we also applied SG to enhance these neighborhood PSO algorithms. The SG-VonNeu PSO algorithm produced improvements

over the VonNeu PSO algorithm on multimodal benchmark functions, and equivalent results on the unimodal benchmark functions. The SG-LBEST PSO algorithm produced improvements over the LBEST PSO algorithm on both unimodal and multimodal benchmark functions.

4.2 Related Research

It has been observed in the PSO literature that the appropriate population size for the particle swarm is problem dependent [9]. We conducted a study of the variation in the PSO population size in Section 3.4.2. In general, the smaller populations emphasize exploitation, offer faster convergence, and performed best on the problems of lower dimension and unimodal functions. Whereas larger populations emphasize exploration, converge slower, and performed best on problems of higher dimension and multimodal functions. This suggests the investigation of a PSO algorithm with a dynamic population size.

Although there have been many papers in the PSO literature adjusting PSO parameters such as the social and cognitive components [43] and the neighborhoods [24, 39, 28], we are aware few papers adapting the PSO population size. One such dynamically adapting swarm is called Tribes and seeks to be a no-parameter swarm algorithm [9]. In Tribes, multiple swarms exist, and there is a method for creating and destroying particles dynamically. Tribes was found to be better than the standard PSO algorithm in that is frees the user from the burden of setting parameters and that it performed "at least equivalent to the result obtained by PSO (with constriction)".

The idea of breaking up the swarm into local and global searchers was studied in the division of labor PSO [50], where the swarm particles are dynamically assigned different tasks every time step based on a response threshold probability formula. The MSG-PSO algorithm borrows the idea of giving particles individual tasks, and offers a simpler and more straightforward method to balance global and local search.

4.3 Stop and Go Particle Swarm Optimization

The SG-PSO algorithm is a simple-to-implement PSO algorithm with a dynamically resizing population. SG-PSO starts with a maximum population and stops selected particles as the optimization is performed. Stopped particles do not perform function evaluations, however, later in the optimization stopped particles may become active again. This "stop and go" design gives the SG-PSO algorithm its unique dynamic population. In addition, the stopped particles act as a type of memory for the swarm.

To determine the appropriate time to stop or start a particle, one helpful parameter is introduced: the required degree of optimization accuracy, r. In the simple case, r, can be considered a sphere in n-dimensional domain space. In the more general case, different weights can be attached to each dimension in a problem specific manner, resulting in an n-dimensional ellipsoid. Since we are dealing with benchmark functions in this study, n-dimensional Euclidean distance is used, with all dimensions being weighted equally.

At the beginning of each time step of the SG-PSO algorithm, the distance between the "solution" (approximated by the swarm's global best) and each particle's personal best is compared. If this distance is less than the required accuracy, r, the particle will neither move nor perform a function evaluation during this time step. The particle is stopped. Stopped particles are close to the global best and thus as particles stop, local search is inhibited and global search is emphasized as the remaining particles are

```
1: initialize population // set $\vec{p}_g$ to $\vec{x}_0$
2: while stopping condition not reached do
3:   for $i = 1$ to maximum population size do
4:     // skip particles within required accuracy
5:     if $\|\vec{p}_i - \vec{p}_g\| > r$ then
6:       for $d = 1$ to number of dimensions do
7:         $r_1$ = rand() // between 0 and 1
8:         $r_2$ = rand()
9:         $v_{id} = wv_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id})$
10:        $x_{id} = x_{id} + v_{id}$
11:      end for
12:      if $f(\vec{x}_i) < f(\vec{p}_i)$ then
13:        $\vec{p}_i = \vec{x}_i$
14:      end if
15:      if $f(\vec{x}_i) < f(\vec{p}_g)$ then
16:        $\vec{p}_g = \vec{x}_i$
17:      end if
18:    end if
19:  end for
20:  if all particles' $\vec{p}_i$ within distance $r$ of $\vec{p}_g$ then
21:    // swarm stopped
22:    restart all particles except the global best
23:  end if
24: end while
```

Figure 4.1: The Stop and Go PSO algorithm. r is the radius of required accuracy. The optimization result is $\vec{p}_g$.

further away from the global best. Note that since the global best can change as time progresses, stopped particles may become active again.

One more detail needs to be mentioned. There is the possibility that all the particles will converge to within the required accuracy of the current swarm's global best, and all particles will become stopped. There may be available function evaluations and the question arises as how to effectively use them. If the particles continued from their current state (for example, using the standard PSO algorithm), function evaluations would be concentrated between their personal bests and the global best, mostly within the required accuracy, r, of the global best. Although this might improve the result slightly, the optimization result is already within the required accuracy where these function evaluations would be occurring. This suggests that the function evaluations might be better applied elsewhere through the use of some type of restarting method.

There are many methods of restarting the whole swarm or part of the swarm in the literature. In [8], a "no-hope" criterion is based on the spread of particles space is used, and when appropriate the particles are restarted around the best particle. In the "self-organizing hierarchical particle swarm optimizer" (HPSO) [43], velocity components are individually reinitialized. In the Gregarious PSO (G-PSO) [42], the velocity vector is completely reinitialized when the particle reaches certain distance from the global best. In [10], after a number of time steps without improvement, the neighborhoods of the particles are randomized. In VBR (Section 3.3), the global best is stored and the whole swarm is reinitialized when the median velocity of the particles drops below a threshold.

The majority of reinitialization methods in the literature are mostly applicable to a particular PSO

variant. Some, such as the "no-hope" criterion, concentrate search around the current global best. In the SG-PSO algorithm, it is our hypothesis that the current global best is within the required accuracy of the nearest local minimum and therefore further function evaluations nearby are not likely to be very useful. In SG-PSO, a more global search oriented reinitialization method is desired. We restart all particles, except the current global best. The particles forget their current personal best, and are restarted with both new random velocities and positions.

In Section 4.5, we observe that SG-PSO yields improved results over standard PSO for multimodal functions. Intuitively, on unimodal functions, the dynamic decrease in population size offered by SG-PSO can result in quicker convergence to the minimum. However, in the standard PSO algorithm it is well-known that for multimodal functions, larger populations are needed to avoid premature convergence to a local minimum. At first glance, it seems that the decrease in population size offered by SG-PSO might be a disadvantage when facing multimodal functions. On the contrary, we hypothesize that the decrease in population size is an advantage when facing multimodal functions. The key is that with SG-PSO particles stop and reinitialization can occur. When convergence occurs to within the required accuracy setting, reinitialization is performed, allowing for the search for better local minimums. Whereas the standard PSO algorithm keeps a constant relatively large population size, investigates many possible local minimums, and then converges on one; the SG-PSO algorithm decreases the population size dynamically, converging quicker to a local minimum and then reinitialization is performed to investigate other local minimums. SG-PSO investigates local minimums in a sequential manner.

The SG-PSO algorithm is a straightforward modification to the standard PSO algorithm. A check for a stopped particle is inserted before line 4 of Figure 2.5. And a check to see if all particles have stopped and reinitialization is required is inserted before line 17 of Figure 2.5. The complete SG-PSO pseudocode is given in Figure 4.1.

4.4 Mixed SG-PSO

The mixed SG-PSO (MSG-PSO) algorithm is an extension to the SG-PSO algorithm to improve the balance between global and local search in PSO. We found that the SG-PSO algorithm's accuracy setting can be applied in a very natural and straightforward way to balance local and global search.

In the MSG-PSO algorithm, each particle is given an individual accuracy setting, r_i, where i is the particle index. Let us call the particles with high accuracy settings (smaller r_i) precision searchers, and the particles with low accuracy settings (larger r_i) rough searchers. The MSG-PSO algorithm works as follows. First, the rough searchers find promising locations in domain space and stop, then precision searchers perform further optimization.

Local search can be emphasized by increasing the number of rough searching particles relative to the number of precision searching particles. This causes the effective moving population to decrease, thereby allocating more function evaluations to local search since smaller populations converge quicker. On the other hand, global search is emphasized by increasing the number of precision searchers. The effective moving population is increased, allocating more functions to global search relative to local search.

The MSG-PSO "stop and go" population dynamics prove effective on both unimodal and multimodal problems, and are analyzed in detail in Section 4.5.4.

The required modification to the SG-PSO algorithm given in Figure 4.1 is simple, line 5 is modified as follows.

5: **if** $\|\vec{x}_i - \vec{p}_g\| > r_i$ **then**

In this study, the r_i, required accuracy settings are constants, fixed at the beginning of the MSG-PSO algorithm.

4.5 Experiments

In this section, we apply SG to enhance the global best, local best, and von Neumann neighborhood PSO algorithms. We observe that both the SG enhanced standard global best PSO algorithm and the SG enhanced von Neumann neighborhood PSO algorithms achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal benchmark functions. Furthermore, we observed that the SG enhanced standard local best PSO algorithm achieves improved results on both unimodal and multimodal benchmark functions.

We study the MSG-PSO and compare it to the SG-PSO algorithm and the STD PSO algorithm. Due to its inherent ability to balance global and local search, the MSG-PSO algorithm produces improvements over the SG-PSO algorithm on both multimodal and unimodal benchmark functions. In comparison to the STD PSO algorithm, the MSG-PSO is found to achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal benchmark functions.

Next, the unique population dynamics of the SG-PSO algorithms are studied. And finally, a comparison of the SG-PSO, VBR-PSO, and differential evolution is provided.

All studies are conducted with the benchmark functions and PSO parameter settings as discussed in Section 3.4.1.

4.5.1 Stop and Go Applied to the Standard PSO Algorithm

In Table 4.1 and Figure 4.2, we present a comparison of the results for standard PSO algorithm to the SG-PSO algorithm with r varying from 10^{-3} to 10^{-9}. First, we note that the accuracy setting of $r = 10^{-9}$ performs about the same as the standard PSO algorithm. This is to be expected, since as $r \to 0$ there will be very few stopped particles. The single exception being the one current global best particle which is always stopped. In particular, note that accuracy setting $r = 10^{-9}$, performed roughly equivalent the standard PSO algorithm on the Rastrigrin function (both achieving no stopping criteria). This is in sharp contrast to the other $r \in [10^{-3}, 10^{-5}]$ accuracy settings, where perfect scores of 50 stopping criteria were achieved for the 10 and 20 dimensional problems.

Let us choose the accuracy setting of $r = 10^{-5}$ for a detailed comparison of SG-PSO to STD PSO.

On the unimodal Sphere function, both achieved the stopping criterion consistently, on all problems. In terms of the average number of function evaluations required, STD PSO achieved the stopping criterion faster on problems of 10, 20, and 30-dimensions. And SG-PSO achieved the stopping criterion faster on problems of 50 and 100-dimensions. We conclude that the algorithms achieved equivalent results on the unimodal Sphere function.

On the unimodal Rosenbrock function, the STD PSO achieved more stopping criteria on the 10, 20-dimensional problems. On the 30 and 50-dimensional problems, SG-PSO achieved more stopping criteria. On the 100-dimensional problem, the stopping criterion was never achieved. However, SG-PSO achieved a better average minimization result. We conclude that the algorithms achieved equivalent results on the unimodal Rosenbrock function.

Table 4.1: A comparison of the STD and SG-PSO algorithms in terms of the number of successes at reaching the stopping criterion and the average optimization result achieved out of 50 trials. The upper number is the number of successes (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	STD	$r = 10^{-3}$	$r = 10^{-4}$	$r = 10^{-5}$	$r = 10^{-9}$
$f_1(x)$	10	**50** (4253)	**50** (4282)	**50** (4282)	**50** (4282)	**50** (4282)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (7768)	**50** (8104)	**50** (8104)	**50** (8104)	**50** (8104)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (12594)	**50** (12754)	**50** (12754)	**50** (12754)	**50** (12754)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (29458)	**50** (28667)	**50** (28667)	**50** (28667)	**50** (28667)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (148263)	**50** (127372)	**50** (126565)	**50** (126802)	**50** (127870)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
$f_2(x)$	10	**40** (142406)	5 (179142)	37 (125160)	36 (128286)	**40** (148904)
		0.6472 (1.46)	0.6898 (1.45)	0.8095 (1.59)	1.12 (1.79)	0.8053 (1.59)
	20	**33** (252238)	4 (140502)	30 (276479)	32 (261125)	26 (239312)
		2.46 (9.58)	3.28 (10.17)	2.37 (10.15)	0.7913 (1.57)	1.25 (1.83)
	30	11 (279047)	0 (—)	9 (257091)	**14** (267583)	**14** (272274)
		4.52 (11.86)	9.32 (17.75)	2.94 (3.52)	6.52 (17.92)	5.16 (11.75)
	50	5 (302240)	1 (156378)	3 (275678)	**8** (252024)	**8** (252024)
		28.93 (38.40)	22.16 (29.50)	48.11 (48.68)	18.17 (25.30)	18.17 (25.30)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		162.77 (89.21)	146.19 (59.63)	153.01 (68.20)	153.01 (68.20)	153.01 (68.20)
$f_3(x)$	10	1 (16280)	**50** (46864)	**50** (50215)	**50** (60698)	0 (—)
		5.67 (3.11)	goal (—)	goal (—)	goal (—)	6.11 (2.80)
	20	0 (—)	**50** (161999)	**50** (190333)	**50** (217527)	0 (—)
		36.57 (15.06)	goal (—)	goal (—)	goal (—)	36.20 (10.52)
	30	0 (—)	**43** (296603)	21 (352006)	8 (368222)	0 (—)
		91.10 (27.40)	0.1469 (0.3422)	0.7015 (0.6917)	1.55 (1.24)	82.16 (27.60)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		226.89 (52.31)	9.42 (4.40)	29.49 (11.69)	50.89 (15.41)	200.84 (51.05)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		771.75 (160.70)	378.08 (74.18)	516.32 (116.15)	562.73 (125.95)	661.86 (124.26)
$f_4(x)$	10	1 (8040)	**28** (241400)	21 (272716)	25 (199891)	1 (6397)
		0.0762 (0.0413)	0.0141 (0.0101)	0.0149 (0.0083)	0.0148 (0.0102)	0.0755 (0.0478)
	20	17 (9584)	23 (69635)	**24** (72983)	23 (93344)	21 (9788)
		0.0248 (0.0195)	0.0172 (0.0147)	0.0166 (0.0111)	0.0148 (0.0090)	0.0267 (0.0231)
	30	24 (14020)	30 (53097)	22 (18944)	**32** (38122)	23 (14136)
		0.0231 (0.0184)	0.0154 (0.0106)	0.0186 (0.0137)	0.0149 (0.0110)	0.0191 (0.0153)
	50	19 (28341)	**24** (54645)	23 (83507)	**24** (67930)	12 (30515)
		0.0572 (0.0793)	0.0247 (0.0260)	0.0233 (0.0210)	0.0328 (0.0342)	0.1743 (0.4016)
	100	6 (148520)	19 (143303)	14 (145665)	**20** (207393)	8 (117050)
		0.7767 (1.94)	0.0586 (0.1010)	0.0776 (0.0789)	0.0828 (0.1009)	0.5292 (0.7876)
$f_5(x)$	2	43 (14299)	40 (11130)	40 (15107)	40 (25973)	**45** (12640)
		0.0014 (0.0034)	0.0019 (0.0039)	0.0019 (0.0039)	0.0019 (0.0039)	0.0010 (0.0029)

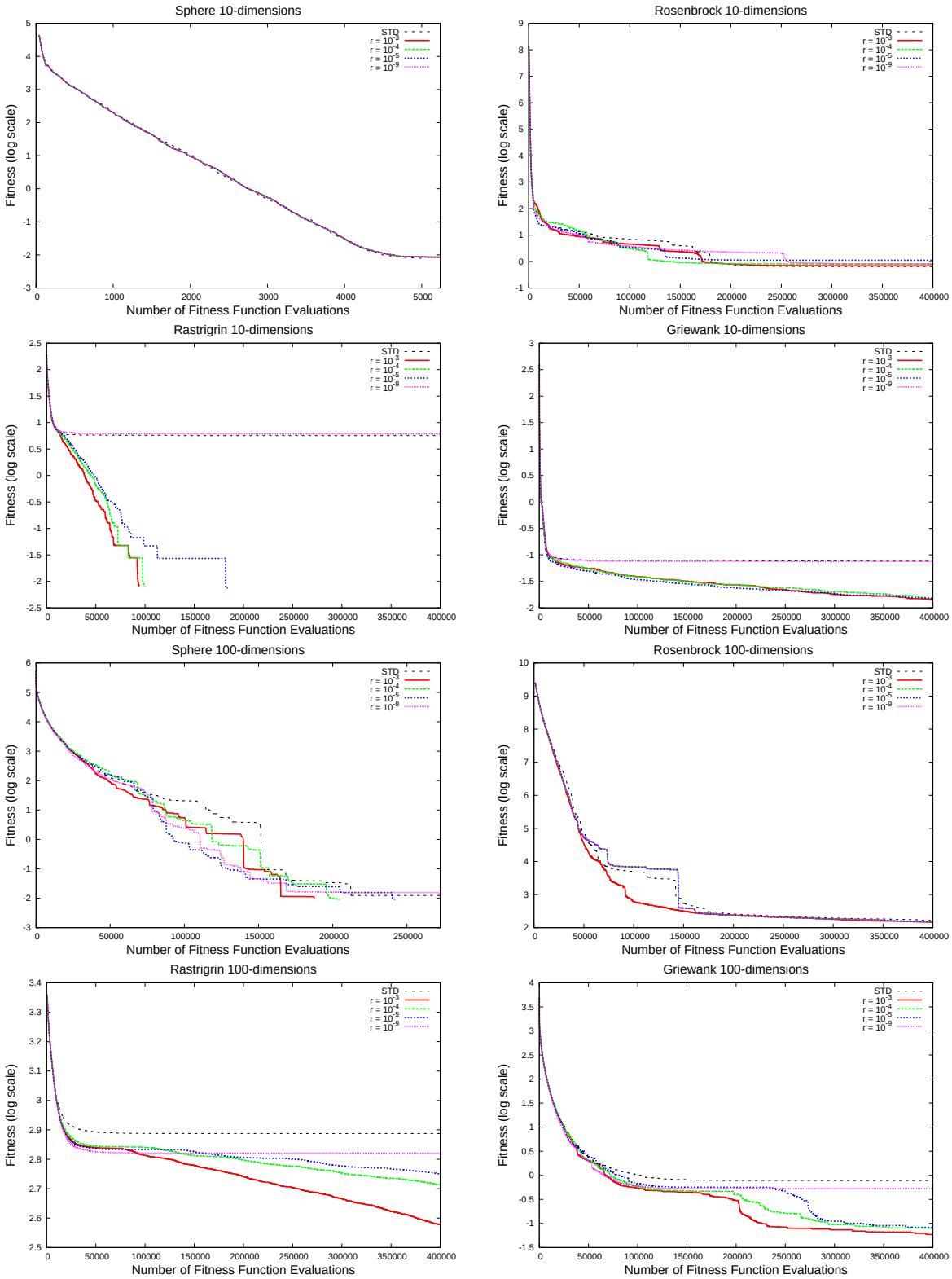

Figure 4.2: A comparison of the STD and SG-PSO algorithms

On the multimodal Rastrigrin function, in 10, 20, and 30-dimensions, SG-PSO achieved more stopping criteria than STD PSO. On the 50 and 100-dimensional problems, the stopping criterion was never achieved. However, SG-PSO achieved a better average minimization result. We conclude that SG-PSO performs better than STD PSO on the multimodal Rastrigrin function.

On the multimodal Griewank function, SG-PSO achieved more stopping criteria on problems of all dimensions. We conclude that SG-PSO performs better than STD PSO on the Griewank function.

On the 2-dimensional Schaffer's f6 function, we find that STD PSO achieved more stopping criteria than SG-PSO. We conclude that STD PSO performs better than SG-PSO on the 2-dimensional Schaffer's f6 function.

Overall, we conclude that SG-PSO has effectively enhanced the ability of the standard PSO algorithm for the multimodal benchmark functions, while producing equivalent results on the unimodal benchmark functions.

4.5.2 Stop and Go Applied to the LBEST and the von Neumann PSOs

In the previous section, SG was observed to improve the performance of the standard global best PSO algorithm. The neighborhood methods, LBEST and von Neumann (VonNeu), are known to improve on the standard global best neighborhood. In this section, we create the SG enhanced the LBEST (SG-LBEST) and the SG enhanced von Neumann (SG-VonNeu) neighborhood PSO algorithms. The results are presented in Table 4.2 and Figure 4.3. The accuracy setting $r = 10^{-4}$, was chosen for this comparison. The columns SG-L and SG-V give the results for the SG-LBEST and SG-VonNeu algorithms.

Comparing the SG-LBEST PSO to the LBEST PSO

On the unimodal Sphere function, both algorithms achieved the stopping criterion consistently. In terms of the number of function evaluations required to achieve the stopping criterion, we observe that SG-LBEST PSO required fewer function evaluations for all problems. We conclude that SG-LBEST PSO performed slightly better than LBEST PSO on the unimodal Sphere function.

On the unimodal Rosenbrock function, the SG-LBEST PSO achieved more stopping criteria on the 10 and 20-dimensional problems. On the 30, 50, and 100-dimensional problems the results were equivalent. In terms of average minimization result achieved, we find that the SG-LBEST PSO achieved a better minimization result on the 30 and 50-dimensional problems. LBEST PSO achieved the better result on the 100-dimensional problem. We conclude that SG-LBEST PSO performed slightly better than LBEST PSO on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, the SG-LBEST PSO achieved more stopping criteria on the 10-dimensional problem. Stopping criteria were not achieved on the other problems. In terms of average minimization result achieved, on the 20, 30, 50, and 100-dimensional problems, we find that SG-LBEST always achieved the better result. We conclude that SG-LBEST PSO performed better on the multimodal Rastrigrin function.

On the multimodal Griewank function, the SG-LBEST PSO achieved more stopping criteria on the 10, 20, 30, and 50-dimension problems. On the 100-dimension problem both SG-LBEST PSO and LBEST PSO achieved identical results. However, in terms of the average number of function evaluations required to reach the stopping criterion, the SG-LBEST PSO required fewer function evaluations. We conclude that SG-LBEST PSO performed better on the multimodal Griewank function.

On the multimodal 2-dimensional Schaffer's f6 function, the results are close with the SG-LBEST PSO achieving one more stopping criteria. We conclude that SG-LBEST PSO performed slightly better on the multimodal Schaffer's f6 function.

Overall, we conclude that the SG-LBEST PSO algorithm yields improved results over the LBEST PSO algorithm for both unimodal and multimodal benchmark functions.

Comparing the SG-VonNeu PSO to the VonNeu PSO

On the unimodal Sphere function, both algorithms achieved the stopping criterion consistently. In terms of the number of function evaluations required to achieve the stopping criterion, we observe that VonNeu PSO required fewer function evaluations for all problems. We conclude that VonNeu PSO performed slightly better than SG-VonNeu PSO on the unimodal Sphere function.

On the unimodal Rosenbrock function, the SG-VonNeu PSO achieved more stopping criteria on the 10 and 30-dimensional problems. On the 20-dimensional problem, VonNeu PSO achieved more stopping criteria. On the 50 and 100-dimensional problems the stopping criterion was never achieved. In terms of average minimization result achieved, the SG-VonNeu PSO achieved better results on the 10, 20, 30, and 50-dimensional problems. The VonNeu PSO achieved a better result on the 100-dimensional problem. We conclude that the two PSO algorithms performed equivalently on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, the SG-VonNeu PSO achieved more stopping criteria on the 10 and 20-dimensional problems. Stopping criteria were not achieved on the other problems. In terms of average minimization result achieved, SG-VonNeu achieved a better result on the 50 and 100-dimensional problems. We conclude that SG-VonNeu PSO performed better on the multimodal Rastrigrin function.

On the multimodal Griewank function, the SG-VonNeu PSO achieved more stopping criteria on all problems. We conclude that SG-VonNeu PSO performed better on the multimodal Griewank function.

On the multimodal 2-dimensional Schaffer's f6 function, both algorithms consistently achieved stopping criterion. However, we observe that SG-VonNeu PSO required fewer function evaluations. We conclude that SG-VonNeu PSO performed slightly better than the VonNeu PSO on the multimodal Schaffer's f6 function.

Overall, we conclude that the SG-VonNeu PSO algorithm yields improved results over the VonNeu PSO algorithm on the multimodal benchmark functions and roughly equivalent results on the unimodal benchmark functions.

4.5.3 Mixed SG-PSO

Table 4.3 and Figure 4.4 give detailed results for the Mixed SG-PSO simulations. The results for MSG-PSO are shown for 50-50 mixtures of particles using the following labels. MIXa is a mixture of accuracy settings of $r = 10^{-4}$ and $r = 1$. MIXb is a mixture of accuracy settings of $r = 10^{-4}$ and $r = 10$. MIXc is a mixture of accuracy settings of $r = 10^{-4}$ and $r = 100$. The SG-PSO results for accuracy setting $r = 10^{-4}$ are given for comparison.

These accuracy settings where chosen based on the results from Section **??**. An accuracy setting of $r = 10^{-4}$, produced good results for the SG-PSO algorithm. Accordingly this setting was chosen for the precision searchers. To make the rough searchers stop more frequently than the precision searchers, there accuracy setting was varied at levels greater than 10^{-4}.

As a guideline for choosing the settings, we recommend basing the precision searchers' accuracy setting

Table 4.2: Applying SG-PSO to the LBEST and von Neumann neighborhoods. SG-L shows the SG enhanced LBEST results. SG-V shows the SG enhanced von Neumann results. The upper number is the number of successes (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	STD	LBEST	SG-L	VonNeu	SG-V
$f_1(x)$	10	**50** (4253)	**50** (7734)	**50** (7034)	**50** (6149)	**50** (6170)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (7768)	**50** (16234)	**50** (14806)	**50** (12527)	**50** (12562)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (12594)	**50** (25090)	**50** (22769)	**50** (19022)	**50** (19381)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (29458)	**50** (44141)	**50** (40270)	**50** (33174)	**50** (34809)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (148263)	**50** (94395)	**50** (89316)	**50** (74669)	**50** (81425)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
$f_2(x)$	10	40 (142406)	44 (299738)	**48** (204923)	42 (220866)	43 (177540)
		0.6472 (1.46)	0.1350 (0.5999)	0.0922 (0.5568)	0.4899 (1.29)	0.4876 (1.29)
	20	**33** (252238)	5 (295032)	17 (294272)	9 (301782)	4 (285094)
		2.46 (9.58)	1.81 (3.01)	1.80 (3.93)	2.37 (3.78)	2.23 (3.80)
	30	**11** (279047)	2 (243980)	2 (180996)	1 (384320)	6 (315014)
		4.52 (11.86)	11.63 (21.21)	5.60 (7.36)	8.55 (13.50)	8.47 (12.96)
	50	**5** (302240)	0 (—)	0 (—)	0 (—)	0 (—)
		28.93 (38.40)	48.03 (41.65)	30.25 (31.90)	71.95 (57.64)	42.80 (40.78)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		162.77 (89.21)	180.73 (72.58)	187.52 (74.93)	160.43 (85.78)	198.48 (75.51)
$f_3(x)$	10	1 (16280)	0 (—)	24 (213043)	3 (223467)	**27** (134584)
		5.67 (3.11)	3.50 (1.78)	1.24 (1.66)	2.57 (1.59)	1.70 (2.40)
	20	0 (—)	0 (—)	0 (—)	0 (—)	**2** (271800)
		36.57 (15.06)	31.78 (9.81)	28.16 (7.95)	23.88 (7.27)	21.23 (8.79)
	30	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		91.10 (27.40)	87.94 (17.00)	74.47 (12.13)	60.17 (17.59)	45.21 (17.50)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		226.89 (52.31)	243.17 (35.26)	201.61 (33.28)	171.21 (35.00)	130.76 (35.37)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		771.75 (160.70)	805.31 (98.05)	687.82 (93.44)	596.62 (91.29)	508.51 (78.97)
$f_4(x)$	10	1 (8040)	9 (64338)	**17** (79783)	6 (93347)	**17** (130395)
		0.0762 (0.0413)	0.0264 (0.0165)	0.0221 (0.0141)	0.0323 (0.0186)	0.0235 (0.0142)
	20	17 (9584)	47 (24161)	**50** (32128)	38 (22824)	43 (39583)
		0.0248 (0.0195)	0.0102 (0.0033)	goal (—)	0.0116 (0.0046)	0.0093 (0.0024)
	30	24 (14020)	48 (35712)	**50** (40132)	40 (21348)	42 (29860)
		0.0231 (0.0184)	0.0099 (0.0015)	goal (—)	0.0122 (0.0065)	0.0107 (0.0040)
	50	19 (28341)	48 (47280)	**50** (43913)	40 (35562)	43 (37023)
		0.0572 (0.0793)	0.0099 (0.0012)	goal (—)	0.0111 (0.0037)	0.0111 (0.0048)
	100	6 (148520)	**50** (94488)	**50** (90987)	40 (73983)	42 (80588)
		0.7767 (1.94)	goal (—)	goal (—)	0.0164 (0.0230)	0.0127 (0.0100)
$f_5(x)$	2	43 (14299)	49 (31336)	**50** (16144)	**50** (18290)	**50** (13178)
		0.0014 (0.0034)	0.0002 (0.0014)	goal (—)	goal (—)	goal (—)

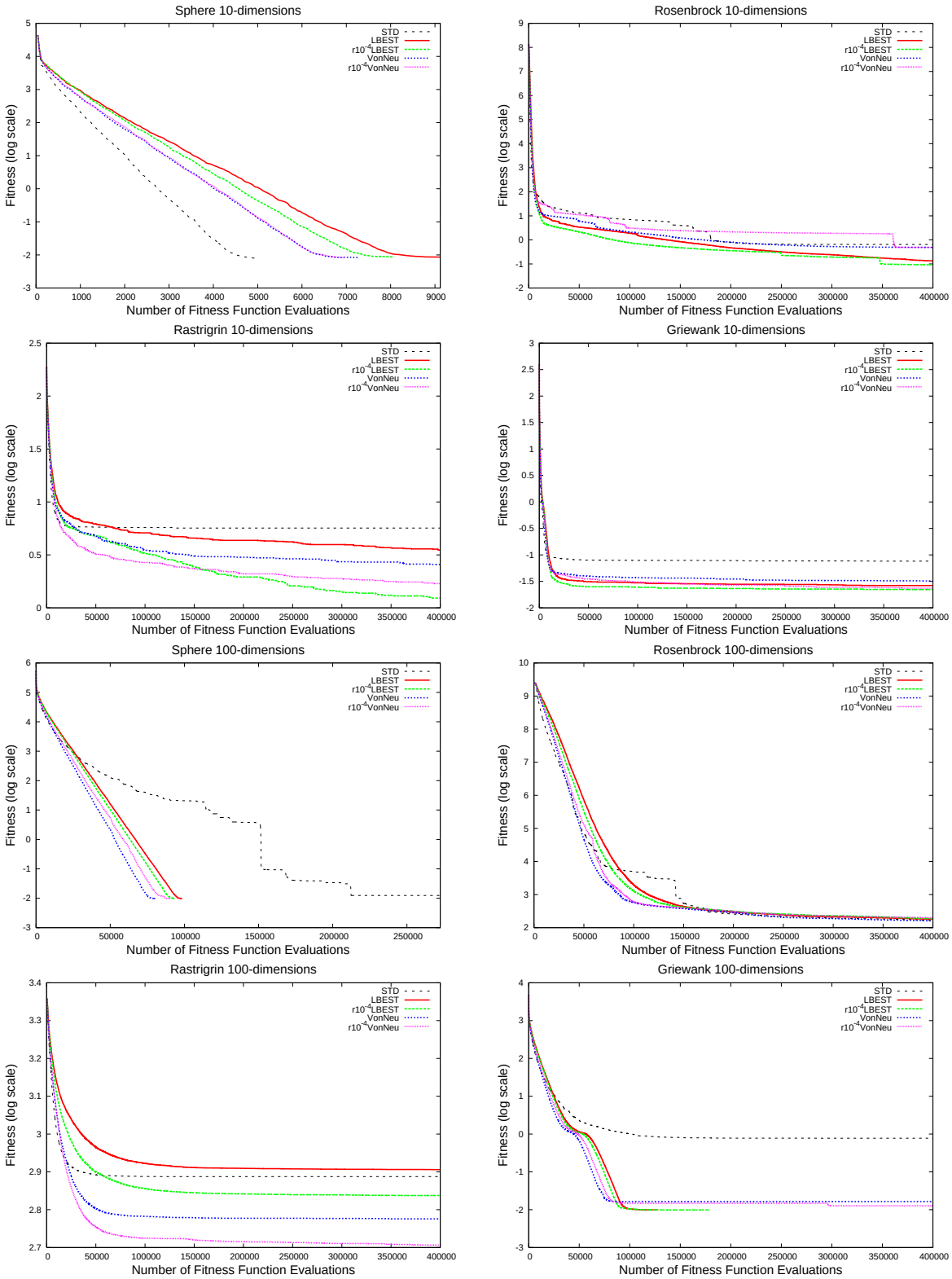

Figure 4.3: Applying SG-PSO to the LBEST and von Neumann neighborhoods

on a good SG-PSO setting, and then adjusting the rough searchers' accuracy setting such that it is greater than the precision searchers' accuracy setting. We note that as the rough searchers' accuracy setting approaches positive infinity, more particles are stopped and the effective PSO population decreases. Also, as the precision searchers' accuracy setting approaches zero, the precision searchers stop less, behaving more and more like the standard PSO.

Comparing MSG-PSO and SG-PSO

Let us make two important observations concerning the MSG-PSO algorithms.

First, on the lower dimensional Sphere problems, the MIXa, MIXb, and MIXc PSO algorithms all achieve the results in slightly fewer function evaluations than the SG-PSO. We can explain this by observing that more particles stop due to half of the particles having higher required accuracies. The MSG-PSO algorithms have an effective smaller population than the SG-PSO algorithm. Recall that smaller populations are observed to be better for the sphere problem of the lower dimensions (see Section 3.4.2). The smaller population results in slightly more function evaluations being applied for MIXb and MIXc on the 100-dimensional Sphere problem. MIXa has the smallest accuracy settings, thus a larger effective population, and requires the fewest function evaluations on average to achieve stopping criterion as expected. Future research might consider basing the accuracy setting on the number of dimensions of the problem domain space.

Second, considering the 2-dimensional Schaffer's f6 problem, we point out that in only two dimensions, the very granular accuracy settings of MIXb and MIXc show a degradation of performance on the 2-dimensional Schaffer's f6 function. We explain this through observing that in two dimensions, an accuracy setting of 100 in the case of MIXc is about 25 % of the search domain 2-D volume. In 30 dimensions, an accuracy setting of 100 is a very small fraction of the search domain 30-D volume, about $100/10^{30}$. The accuracy setting of 100 is too granular for 2-D Schaffer's f6 function, leading to the degradation in performance.

Let us perform a detail comparison to the SG-PSO algorithm using MIXa, referring to Table 4.3 and Figure 4.4.

On the unimodal Sphere function, both algorithms achieve the stopping criterion consistently. In terms of the number of function evaluations required to achieve the stopping criterion, we observe that MSG-PSO required fewer function evaluations for all problems. We conclude that MSG-PSO performed better than SG-PSO on the unimodal Sphere function.

On the unimodal Rosenbrock function, MSG-PSO achieved more stopping criteria on the 10, 20, 30 and 50-dimensional problems. On the 100-dimensional problem the stopping criterion was never achieved. In terms of average minimization result achieved, SG-PSO achieved better results on the 100-dimensional problem. We conclude that the MSG-PSO algorithm performed better on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, the MSG-PSO achieved more stopping criteria on the 30-dimensional problem. On the 10 and 20-dimensional problems, both MSG-PSO and SG-PSO achieved the stopping criterion consistently. On the 50 and 100-dimensional problems, the stopping criterion was not achieved. In terms of average minimization result achieved, MSG-PSO achieved a better result on the 50 and 100-dimensional problems. We conclude that MSG-PSO performed better on the multimodal Rastrigrin function.

On the multimodal Griewank function, the MSG-PSO achieved more stopping criteria on the 10,

20, 30, and 100-dimensional problems. On the 50-dimensional problem, both algorithms achieved the same number of stopping criterion. However, on the 50-dimensional problem, MSG-PSO achieving the stopping criterion with fewer function evaluations on average. We conclude that MSG-PSO performed better on the multimodal Griewank function.

On the multimodal 2-dimensional Schaffer's f6 function, MSG-PSO achieved more stopping criteria. We conclude that MSG-PSO performed better on the multimodal Schaffer's f6 function.

Overall, we conclude that MSG-PSO can be applied to increase the effectiveness of the SG-PSO algorithm. In particular, we suggest that the degree of global and local search can be adjusted by configuring the mix of high and low accuracy settings. Future research might consider how to adapt the individual accuracy settings, dynamically.

Comparing MSG-PSO and STD PSO

Let us compare the results of MSG-PSO (using MIXa) and the STD PSO, referring to Table 4.3 and Figure 4.4.

Both algorithms achieve the stopping criterion consistently for the Sphere function. In terms of the number of function evaluations required to achieve the stopping criterion, we observe that MSG-PSO required fewer function evaluations for all problems. We conclude that MSG-PSO performed better than STD PSO on the unimodal Sphere function.

On the unimodal Rosenbrock function, MSG-PSO achieved more stopping criteria on the 10, and 50-dimensional problems. Both MSG-PSO and STD PSO achieved the same number of stopping criteria on the 20-dimensional problem, with STD PSO achieving the better average minimization result. On the 30-dimensional problem, STD PSO achieved more stopping criteria. On the 100-dimensional problem, the stopping criterion was not achieved. In terms of average minimization result, STD PSO achieved the better result. We conclude that MSG-PSO and STD PSO performed equivalently on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, the MSG-PSO achieved more stopping criteria on the 10, 20, and 30-dimensional problems. On the 50 and 100-dimensional problems, the stopping criterion was not achieved. However, in terms of average minimization result achieved, MSG-PSO achieved a better result. We conclude that MSG-PSO performed better on the multimodal Rastrigrin function.

On the multimodal Griewank function, the MSG-PSO achieved more stopping criteria on all problems. We conclude that MSG-PSO performed better on the multimodal Griewank function.

On the multimodal 2-dimensional Schaffer's f6 function, STD PSO achieved one more stopping criterion. We conclude that STD-PSO performed slightly better on the multimodal Schaffer's f6 function.

Overall, we conclude that MSG-PSO performs better than the STD PSO on the multimodal functions and equivalent or better on the unimodal benchmark functions.

4.5.4 Population Dynamics

The population dynamics reveal hints about the inner workings of the SG-PSO algorithms. Most notably, the SG-PSO algorithms produce distinct population dynamics on unimodal and multimodal benchmark functions. While reinitializations occur on multimodal functions, they do not occur on unimodal functions (with reasonably small accuracy settings).

Table 4.3: Comparison of the STD, SG-PSO, and MSG-PSO algorithms in terms of the number of successes at reaching the stopping criterion and the average optimization result achieved out of 50 trials. The upper number is the number of successes (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	STD	$r = 10^{-4}$	MIXa	MIXb	MIXc
$f_1(x)$	10	**50** (4253)	**50** (4282)	**50** (3945)	**50** (3356)	**50** (2691)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (7768)	**50** (8104)	**50** (7335)	**50** (6523)	**50** (5428)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (12594)	**50** (12754)	**50** (11724)	**50** (10817)	**50** (9851)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (29458)	**50** (28667)	**50** (26603)	**50** (27340)	**50** (27782)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (148263)	**50** (126565)	**50** (113504)	**50** (132842)	**50** (135474)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
$f_2(x)$	10	40 (142406)	37 (125160)	**41** (129385)	39 (110730)	37 (93232)
		0.6472 (1.46)	0.8095 (1.59)	0.4880 (1.29)	0.6720 (1.46)	0.8870 (1.65)
	20	**33** (252238)	30 (276479)	**33** (237257)	25 (165890)	30 (173013)
		2.46 (9.58)	2.37 (10.15)	2.82 (13.32)	2.06 (2.81)	0.8429 (1.59)
	30	11 (279047)	9 (257091)	15 (245726)	19 (208132)	**22** (260319)
		4.52 (11.86)	2.94 (3.52)	6.77 (17.80)	5.35 (17.05)	1.80 (2.58)
	50	5 (302240)	3 (275678)	4 (334443)	**8** (281692)	**8** (309432)
		28.93 (38.40)	48.11 (48.68)	31.66 (37.95)	11.66 (21.40)	15.37 (26.62)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		162.77 (89.21)	153.01 (68.20)	164.59 (68.26)	173.62 (74.69)	154.58 (102.93)
$f_3(x)$	10	1 (16280)	**50** (50215)	**50** (53614)	**50** (40268)	**50** (40229)
		5.67 (3.11)	goal (—)	goal (—)	goal (—)	goal (—)
	20	0 (—)	**50** (190333)	**50** (183660)	**50** (154486)	**50** (171000)
		36.57 (15.06)	goal (—)	goal (—)	goal (—)	goal (—)
	30	0 (—)	21 (352006)	25 (313729)	34 (316332)	**36** (316737)
		91.10 (27.40)	0.7015 (0.6917)	0.6838 (0.8253)	0.3848 (0.6528)	0.3064 (0.5333)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		226.89 (52.31)	29.49 (11.69)	16.64 (6.16)	20.33 (7.76)	21.34 (9.34)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		771.75 (160.70)	516.32 (116.15)	432.78 (113.38)	478.86 (113.10)	551.00 (122.92)
$f_4(x)$	10	1 (8040)	21 (272716)	31 (229385)	21 (210832)	**41** (183953)
		0.0762 (0.0413)	0.0149 (0.0083)	0.0122 (0.0070)	0.0143 (0.0083)	0.0105 (0.0054)
	20	17 (9584)	24 (72983)	**31** (78251)	30 (60372)	26 (57964)
		0.0248 (0.0195)	0.0166 (0.0111)	0.0137 (0.0104)	0.0140 (0.0081)	0.0177 (0.0143)
	30	24 (14020)	22 (18944)	30 (30426)	**35** (21337)	25 (47385)
		0.0231 (0.0184)	0.0186 (0.0137)	0.0138 (0.0068)	0.0131 (0.0071)	0.0188 (0.0138)
	50	19 (28341)	**23** (83507)	**23** (51310)	20 (49508)	18 (64803)
		0.0572 (0.0793)	0.0233 (0.0210)	0.0210 (0.0158)	0.0285 (0.0298)	0.0348 (0.0352)
	100	6 (148520)	14 (145665)	15 (165840)	**18** (196682)	10 (208829)
		0.7767 (1.94)	0.0776 (0.0789)	0.0550 (0.0526)	0.0621 (0.0649)	0.1100 (0.1079)
$f_5(x)$	2	**43** (14299)	40 (15107)	42 (37402)	35 (9354)	34 (19341)
		0.0014 (0.0034)	0.0019 (0.0039)	0.0016 (0.0036)	0.0029 (0.0045)	0.0031 (0.0045)

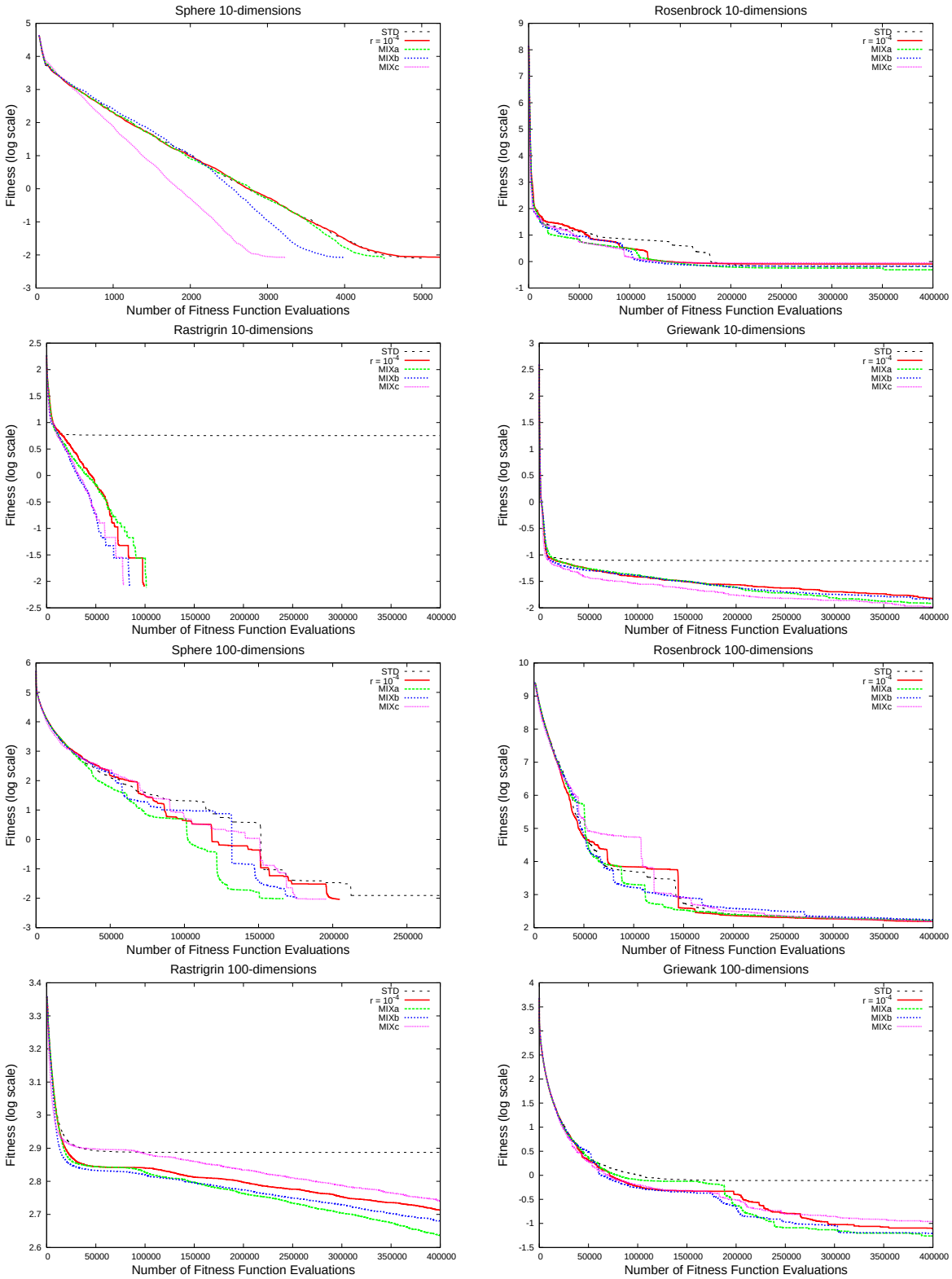

Figure 4.4: Comparison of the STD, SG-PSO, and MSG-PSO algorithms

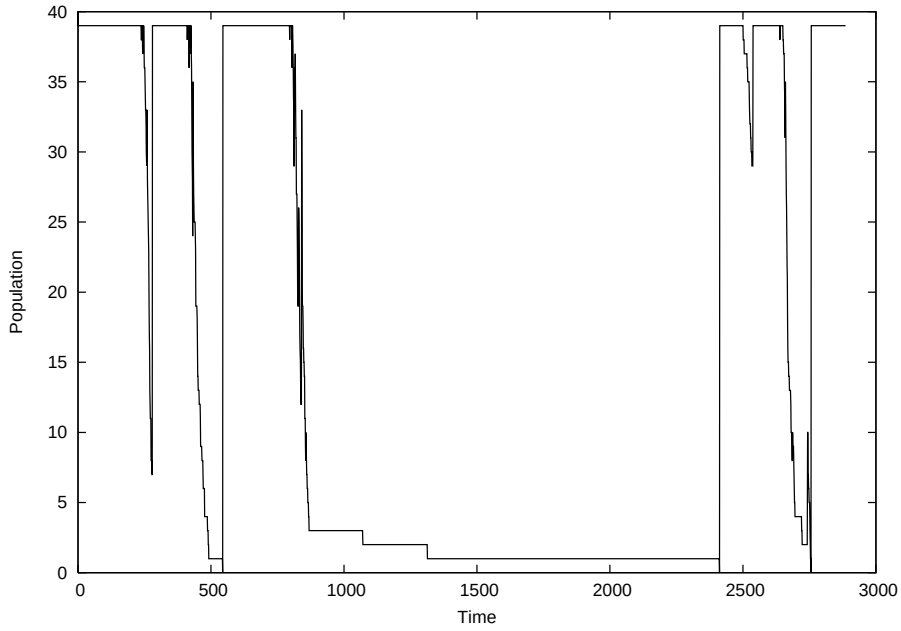

Figure 4.5: Variation in population size vs. time for the multimodal Rastrigrin 10-D benchmark function, for an SG-PSO simulation with an accuracy setting of $r = 10^{-4}$. The population remains steady optimizing like standard PSO until the 300^{th} time step at which point most particles stop, the population size dropping below 10. Soon, a better global best is found by the these few particles and all particles continue optimization from their previous locations. At about time step 500, the active population has dropped again and all particles stop, all particles except the global best are reinitialized. Reinitializations occur again at approximately time steps 2500 and 2800. The optimization is finished when the stopping criterion is reached at about time step 2900.

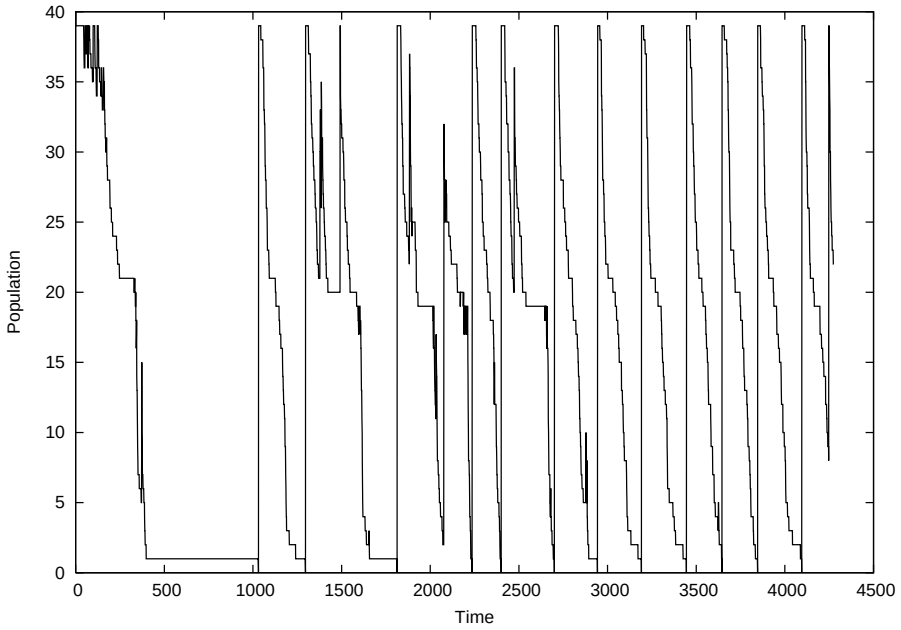

Figure 4.6: Variation in population size vs. time for the multimodal Rastrigrin 10-D benchmark function, for a MSG-PSO simulation with a 50-50 mixture of accuracy settings of $r = 10^{-4}$ (precision searchers) and $r = 1$ (rough searchers). Initially, the rough searchers stop and go as the global best is consistently improved. At about time step 250, significant improvements in the global best are not found and the rough searchers begin to stop and not reactivate. There is a short plateau until about time step 300 where mostly precision searchers are concentrating on local improvement. As this becomes futile, the precision searchers begin to stop and the active population drops briefly to 5. The global best is then improved significantly and the population jumps to 15. However, further significant improvement is not found and all particles reach their required accuracy and stop at about time period 1000. At this point all the particles except the global best are reinitialized. Optimization continues in a similar manner, after 12 more reinitializations the stopping criterion is reached at about time step 4300.

An example of the multimodal SG-PSO population dynamics is shown in Figure 4.5. [1] Initially, SG-PSO behaves much like standard PSO, with all particles except the global best particle moving. Then, as the particles get near a local minimum, many particles get within the required accuracy of the current global best. These particles stop while the moving particles continue to search. If the global best is improved significantly, the stopped particles activate again. Eventually, when all particles are within the required accuracy of the global best, they stop and all particles except the global best are reinitialized.

We hypothesize that it is the stop, wait, and go dynamics that allow for the more effective use of function evaluations in the SG-PSO algorithms, and thus the improvements in optimization results over the standard PSO algorithm. This is particularly clear in MSG-PSO where there are two particle accuracy settings: rough searchers (low accuracy setting) and precision searchers (high accuracy setting). The rough searchers are observed to stop first, and wait for local search to progress before continuing movement.

An example of the multimodal MSG-PSO population dynamics is shown in Figure 4.6. Throughout the run, there are plateaus observed at a population of about 20, where the rough searchers have mostly stopped, but the precision searchers are continuing to improve the local minimum.

[1] The graphs are from [5], constructed from the average of 200 runs of 200,000 function evaluations each, where a time step is 40 function evaluations.

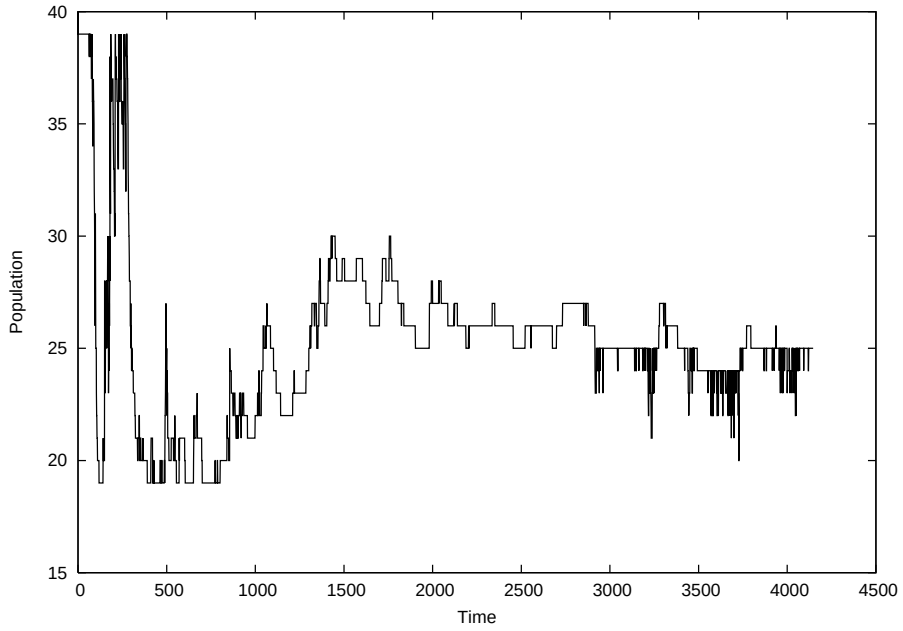

Figure 4.7: Variation in population size vs. time for the unimodal Rosenbrock 10-D benchmark function, for a MSG-PSO simulation with a 50-50 mixture of accuracy settings of $r = 10^{-4}$ (precision searchers) and $r = 1$ (rough searchers). After the first 300 time steps, most of the rough searchers have stopped, being close enough to the global best. The mostly precision searchers continue to improve the minimum, until approximately time step 800. From this point on, consistent improvements in the global best are found and some rough searchers activate again. The stop and go behavior continues until the end of the run, with the population (mostly precision searchers) fluctuating around 25. Unlike multimodal function optimization, reinitialization is rarely observed.

Although the population dynamics of the multimodal benchmark functions of 10 or more dimensions were all similar to the Rastrigrin function studied in Figure 4.5 and Figure 4.6, the population dynamics of the unimodal benchmark functions are different. For the SG-PSO, the graphs of the unimodal Rosenbrock (and Sphere) population dynamics were mostly straight lines, with only the current global best particle ever stopped. The MSG-PSO population dynamics on the unimodal Rosenbrock function are more interesting. In Figure 4.7, we observe stop and go action throughout the run, and hypothesize that it is the mixture of precision and rough searchers the produces these population dynamics. In both SG-PSO and MSG-PSO, reinitializations are not observed on unimodal functions.

4.6 Comparison of PSO Algorithms

We now present one last comparison, including the standard PSO algorithm, VBR-PSO, SG-PSO, and the differential evolution (DE) algorithm (discussed in Section 2.4). DE was chosen since it is becoming popular in the current literature, and was one of the algorithms used for comparison to PSO in [37]. Results for SG and VBR are given with threshold setting of $\alpha = 10^{-4}$ and with an accuracy setting of $r = 10^{-5}$, respectively. We also combined the SG and VBR methods to produce the SG-VBR PSO algorithm, where both SG and VBR are performed, a threshold setting of $\alpha = 10^{-4}$ and an accuracy setting of $r = 10^{-5}$ were applied. The results are presented in Table 4.4 and Figure 4.8.

4.6.1 Comparing VBR-PSO to SG-PSO

First, let us look at the SG and VBR methods. On the unimodal Sphere function, both SG-PSO an VBR-PSO consistently achieve the stopping criterion. In terms of the average number of function evaluations required to achieve the stopping criterion, VBR-PSO performed better on the problems with fewer than 50 dimensions, and SG-PSO performed better on the 50 and 100-dimension problems. We conclude that VBR-PSO and SG-PSO performed equivalently on the unimodal Sphere problem.

On the unimodal Rosenbrock function, the results are also roughly equivalent, with VBR-PSO performing better in terms of the number of stopping criteria achieved on the 10 and 20-dimensional problems and SG-PSO performing better on the 30 and 50-dimensional problems. In terms of average minimization result achieved, VBR-PSO performed better on the 10 and 30-dimensional problems, and SG-PSO perform best on the 20, 50, and 100-dimensional problems. Overall, we conclude VBR-PSO and SG-PSO performed equivalently on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in terms of the number of stopping criteria achieved, SG-PSO outperformed VBR-PSO on the 10, 20, and 30-dimensional problems, with neither algorithm achieving the criterion on the higher dimensional problems. In terms of the average minimization result, SG-PSO outperformed VBR-PSO on all problems. We conclude that SG-PSO clearly outperformed VBR-PSO on the multimodal Rastrigrin function.

On the multimodal Griewank function, in terms of the number of stopping criteria achieved, we find that VBR-PSO outperformed SG-PSO on the 20, 30, and 50-dimensional problems. And, SG-PSO outperformed VBR-PSO on the 10 and 100-dimensional problems. We conclude that VBR-PSO performed slightly better on the multimodal Griewank function.

On the 2-dimensional Schaffer's f6 function, we find that VBR-PSO clearly performed better SG-PSO.

Overall, both VBR-PSO and SG-PSO perform roughly equivalently. Each outperforming the other

on different problems. This leads to the idea of combining the SG and VBR methods.

4.6.2 Combining VBR and SG

In Section 4.6.1, we observed that VBR-PSO and SG-PSO perform better on slightly different problems. Here we combine the two algorithms to create the VBR-SG PSO algorithm. A VBR threshold setting off $\alpha = 10^{-3}$ and a SG accuracy setting of $r = 10^{-5}$ were applied.

On the unimodal Sphere function, the stopping criteria were consistently achieved. In terms of the average number of function evaluations required to achieve the stopping criterion, the results have the characteristics of the SG-PSO algorithm. Before the VBR threshold is reached, the particles reach the required accuracy and are stopped in place.

On the unimodal Rosenbrock function, the results are identical to the results of SG-PSO algorithm, in terms of the number of stopping criteria achieved, the average number of function evaluations required, and the average minimization result achieved. The VBR threshold is not reached and the SG-PSO algorithm takes precedence.

On the multimodal Rastrigrin function, the results are different. Here the VBR threshold takes precedence over the SG required accuracy. Similar to VBR-PSO, the VBR-SG PSO algorithm achieves no stopping criteria, where VBR-PSO achieved only two stopping criteria and SG-PSO achieved over 100 stopping criteria. The VBR-SG PSO achieved a better average minimization result for problems of all sizes, in comparison to VBR-PSO. Overall on the Rastrigrin function, SG-PSO performs the best, followed by VBR-SG PSO, and then VBR-PSO. The VBR-SG PSO algorithm offers an intermediate result, closer to VBR-PSO than SG-PSO.

On the multimodal Griewank function, SG-PSO achieved the most stopping criteria on the 10 and 100-dimensional problems. For the 20 and 30-dimensional problems, both VBR-PSO and VBR-SG PSO achieve perfect scores bettering SG-PSO. On the 50-dimensional problem, VBR-SG PSO performs the best followed by VBR-PSO. Like the Rastrigrin function, we see again that VBR-SG PSO has produced a result intermediate of VBR-PSO and SG-PSO.

On the 2-dimensional Schaffer's f6 function in terms of the number of stopping criteria achieved, we see that the VBR-PSO result is best followed by VBR-SG PSO, and the SG-PSO. Like the Rastrigrin and Griewank functions, the VBR-SG PSO result is intermediate of VBR-PSO and SG-PSO.

We observe that the combined VBR-SG PSO algorithm shares some of the characteristics of both the VBR and the SG algorithms. On the unimodal benchmark functions, the results were close or identical to the SG-PSO algorithm: the particles reach the SG required accuracy before the VBR threshold is achieved. On the multimodal functions, the results closely resembled the VBR-PSO results: VBR threshold is mostly achieved before the SG required accuracy is reached. Further research is needed to investigate other methods of combining the SG and VBR algorithms.

4.6.3 Comparing PSO and DE

For comparison to DE, we applied the $DE/rand/1/bin$ algorithm and settings as described in (Section 2.4). We will use the SG-PSO algorithm for the DE to PSO comparison.

On the unimodal Sphere function, both DE and SG-PSO reached the stopping criteria consistently. Looking at the average number of function evaluations required to reach the stopping criterion, we find that DE required many more function evaluations. DE requires about 3.8 times as many function

evaluations as SG-PSO on average to reach the stopping criterion on the 10-dimensional Sphere problem. And about 1.2 times as many function evaluations on average to reach the stopping criterion on the 100-dimensional Sphere problem. We conclude that SG-PSO performed better than DE on the unimodal Sphere function.

On the Rosenbrock function, in terms of the number of stopping criteria achieved, DE is clearly the best performer on the 10, 20, 30-dimensional problems. On the 50-dimensional problem, SG-PSO outperformed DE. On the 100-dimensional problem, the stopping criterion was not achieved, and SG-PSO achieved the better average minimization results. We conclude that DE performed better than SG-PSO on the unimodal Rosenbrock function.

On the multimodal Rastrigrin function, in terms of the number of stopping criteria achieved, we find that SG-PSO outperforms DE on the 10, 20, and 30-dimensional problems. On the 50 and 100-dimensional problems, the stopping criterion was not achieved. In terms of average minimization result achieved, the SG-PSO outperforms DE on the 50 and 100-dimensional problems. We conclude that SG-PSO performed better than DE on the multimodal Rastrigrin function.

On the multimodal Griewank function, in terms of the number of stopping criteria achieved, DE outperforms SG-PSO on all problems, achieving the stopping criteria consistently on the 50 and 100-dimensional problems. We conclude that DE performed better than SG-PSO on the multimodal Griewank function.

On the multimodal 2-dimensional Schaffer's f6 function, DE outperforms SG in terms of the number of stopping criteria achieved.

The graphs in Figure 4.8, illustrate an advantage of the PSO algorithms over DE. It clear the DE is a bit of a slow starter relative to PSO. On the 100-dimensional problems, for the first 100,000 function evaluations, DE is always outperformed by PSO. The same is true for the 10-dimensional problems, with the exception of the Rosenbrock function.

Overall, we conclude that both DE and PSO have their strong points, with neither outperforming the other on all benchmark functions. DE performs quite well when enough function evaluations are permitted, on the other hand, with limited function evaluations, PSO outperforms DE.

4.7 Conclusion

In this section, we studied the stop and go method, a new method to dynamically adapt the PSO population size. SG was applied to three different PSO algorithms: the standard global best, the local best neighborhood, and the von Neumann neighborhood. We observed that both the SG enhanced standard global best PSO algorithm and the SG enhanced von Neumann neighborhood PSO algorithms achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal benchmark functions. Furthermore, we observed that the SG enhanced standard local best PSO algorithm achieves improved results on both unimodal and multimodal benchmark functions.

We hypothesize that as particles approach the swarm's global best and stop, local search is postponed, allowing for the remaining particles to effectively apply function evaluations to global search. SG-PSO is a method of dynamically balancing local and global search.

As an extension to the SG-PSO algorithm, we developed the mixed SG-PSO (MSG-PSO) algorithm to further facilitate the balance between global and local search. In the MSG-PSO algorithm, particles are given individual accuracy settings, resulting in a mixture of precision and rough searching particles.

Table 4.4: Comparison of PSO algorithms introduced in this thesis. The upper number is the number of successes at reaching the stopping criterion (bold if the best result for the benchmark function) and the number in parenthesis is the average number of function evaluations required for a success. The lower number is the average optimization result achieved and the number in parenthesis is the standard deviation of the results.

Func.	Dim	STD	DE	$\alpha = 10^{-4}$	$r = 10^{-5}$	VBR-SG
$f_1(x)$	10	**50** (4253)	**50** (16194)	**50** (4253)	**50** (4282)	**50** (4282)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	20	**50** (7768)	**50** (37224)	**50** (7768)	**50** (8104)	**50** (8104)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	30	**50** (12594)	**50** (57534)	**50** (12594)	**50** (12754)	**50** (12754)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	50	**50** (29458)	**50** (91762)	**50** (29458)	**50** (28667)	**50** (28667)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
	100	**50** (148263)	**50** (172688)	**50** (152207)	**50** (126802)	**50** (125168)
		goal (—)	goal (—)	goal (—)	goal (—)	goal (—)
$f_2(x)$	10	40 (142406)	**50** (63828)	40 (142406)	36 (128286)	36 (128286)
		0.6472 (1.46)	goal (—)	0.6472 (1.46)	1.12 (1.79)	1.12 (1.79)
	20	33 (252238)	**50** (177186)	33 (252238)	32 (261125)	32 (261125)
		2.46 (9.58)	goal (—)	2.46 (9.58)	0.7913 (1.57)	0.7913 (1.57)
	30	11 (279047)	**49** (341684)	11 (279047)	14 (267583)	14 (267583)
		4.52 (11.86)	0.0171 (0.0550)	4.52 (11.86)	6.52 (17.92)	6.52 (17.92)
	50	5 (302240)	0 (—)	5 (274136)	**8** (252024)	**8** (252024)
		28.93 (38.40)	37.82 (22.10)	27.58 (31.21)	18.17 (25.30)	18.17 (25.30)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		162.77 (89.21)	226.56 (85.98)	162.77 (89.21)	153.01 (68.20)	153.01 (68.20)
$f_3(x)$	10	1 (16280)	41 (279585)	2 (299040)	**50** (60698)	0 (—)
		5.67 (3.11)	0.6382 (2.13)	1.83 (0.8284)	goal (—)	1.67 (0.5781)
	20	0 (—)	0 (—)	0 (—)	**50** (217527)	0 (—)
		36.57 (15.06)	37.37 (19.02)	16.42 (4.01)	goal (—)	15.50 (2.84)
	30	0 (—)	0 (—)	0 (—)	**8** (368222)	0 (—)
		91.10 (27.40)	98.24 (29.87)	46.90 (7.68)	1.55 (1.24)	41.85 (5.81)
	50	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		226.89 (52.31)	254.57 (38.22)	150.43 (21.12)	50.89 (15.41)	138.56 (22.30)
	100	**0** (—)	**0** (—)	**0** (—)	**0** (—)	**0** (—)
		771.75 (160.70)	683.55 (43.27)	626.99 (85.96)	562.73 (125.95)	575.38 (67.07)
$f_4(x)$	10	1 (8040)	**38** (170800)	6 (230767)	25 (199891)	7 (184431)
		0.0762 (0.0413)	0.0108 (0.0055)	0.0222 (0.0099)	0.0148 (0.0102)	0.0209 (0.0089)
	20	17 (9584)	48 (55390)	**50** (47549)	23 (93344)	**50** (61319)
		0.0248 (0.0195)	0.0092 (0.0035)	goal (—)	0.0148 (0.0090)	goal (—)
	30	24 (14020)	**50** (61524)	**50** (41771)	32 (38122)	**50** (34507)
		0.0231 (0.0184)	goal (—)	goal (—)	0.0149 (0.0110)	goal (—)
	50	19 (28341)	**50** (93614)	42 (95713)	24 (67930)	49 (94520)
		0.0572 (0.0793)	goal (—)	0.0126 (0.0078)	0.0328 (0.0342)	0.0102 (0.0031)
	100	6 (148520)	**50** (168886)	7 (268640)	20 (207393)	18 (227465)
		0.7767 (1.94)	goal (—)	0.1690 (0.2380)	0.0828 (0.1009)	0.1290 (0.2067)
$f_5(x)$	2	43 (14299)	**50** (14092)	**50** (39930)	40 (25973)	49 (34537)
		0.0014 (0.0034)	goal (—)	goal (—)	0.0019 (0.0039)	0.0002 (0.0014)

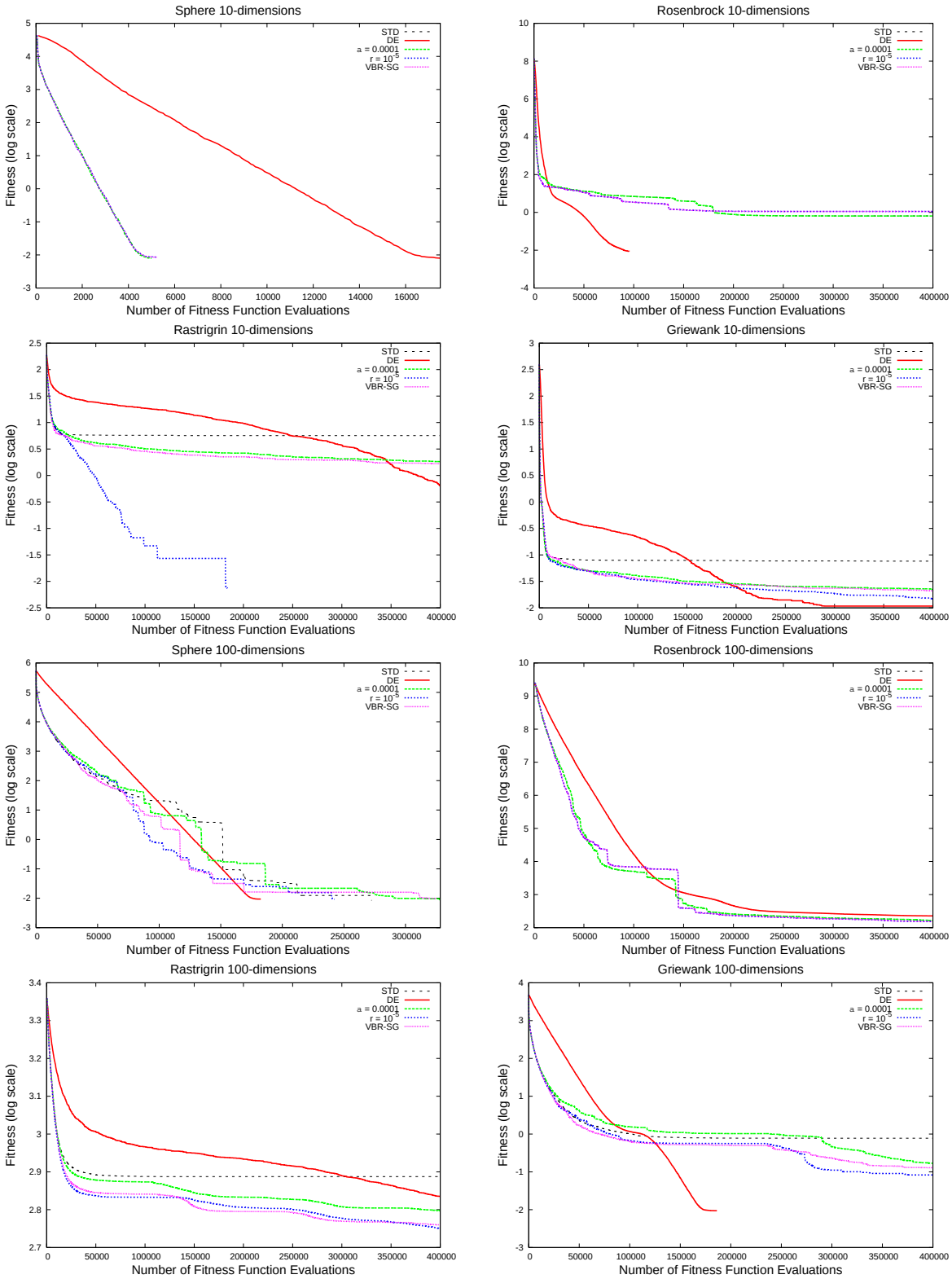

Figure 4.8: Comparison of PSO to DE

First, the rough searchers find promising locations in domain space and stop, then precision searchers perform further optimization. The MSG-PSO algorithm was observed to show improvements over the SG-PSO algorithm.

We analyzed the population dynamics of the SG-PSO and MSG-PSO algorithms. As the particles required accuracies were achieved, the particles were observed to stop and wait for the search to progress before continuing. We hypothesize that the "stop and go" behavior of the particles in the SG-PSO and MSG-PSO algorithms allows for more effective application of function evaluations, leading to gains in performance over the standard PSO algorithm. We noted that while reinitializations occur on multimodal functions, they rarely occur on unimodal functions.

We compared the VBR-PSO, SG-PSO, and differential evolution (DE) algorithms. We observed that the SG-PSO and VBR-PSO algorithms produced roughly equivalent results, each outperforming the other on different benchmark functions. When given enough function evaluations, it was found that the DE algorithm outperformed SG-PSO. However, DE was a slow starter and was outperformed by SG-PSO when the number of function evaluations was limited.

Chapter 5

Conclusion

In this thesis, two new methods for increasing the effectiveness of PSO were proposed: velocity-based reinitialization (VBR) and stop and go (SG).

5.1 Velocity-based Reinitialization PSO

The velocity-based reinitialization method is different from other reinitialization methods in that it uses a novel median-based method to determine when reinitialization should occur. VBR helps PSO algorithms achieve a good balance between exploitation (local search) and exploration (global search). With VBR the algorithm can first exploit, and then through reinitialization resume exploration.

VBR is simple to implement and has only one threshold parameter. Our experiments showed that VBR was not too sensitive to the threshold setting. The family of VBR enhanced PSO algorithms also gracefully approaches the standard PSO algorithm as the threshold setting approaches zero.

VBR was applied to enhance three different PSO algorithms: the global best, local best, and von Neumann neighborhood PSO algorithms. It was observed that the VBR enhanced PSO algorithms achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal functions.

We suggest the following possible avenues of research concerning VBR. First, since VBR applies complete swarm reinitialization, saving the current global best and restarting, VBR may be a fruitful method to apply to dynamic problems. With VBR, changes in the environment can be detected naturally after reinitialization. Second, investigations into basing the VBR threshold parameter on the number of dimensions of the search domain space or dynamically adapting the threshold parameter appear promising.

5.2 Stop and Go PSO

The stop and go method is a new method to dynamically adapt the PSO population size. We observed that in many practical problems there is a limit to the required accuracy of the optimization result. The SG-PSO algorithm takes advantage of this information to dynamically adjust the population size resulting in more effective use of function evaluations. More specifically, in SG-PSO, when a particle has approximately reached the required accuracy in domain space (accuracy being measured by closeness in domain space, not to the unknown optimum function value), it is stopped. The population size is

effectively dynamically reduced. Stopped particles halt local search, allowing for the active particles to continue global search.

SG was applied to three different PSO algorithms: the standard global best, the local best neighborhood, and the von Neumann neighborhood. We observed that both the SG enhanced standard global best PSO algorithm and the SG enhanced von Neumann neighborhood PSO algorithms achieve improved results on the multimodal benchmark functions, while yielding equivalent performance on the unimodal benchmark functions. Furthermore, we observed that the SG enhanced standard local best PSO algorithm achieves improved results on both unimodal and multimodal benchmark functions.

We hypothesize that as particles approach the swarm's global best and stop, local search is postponed, allowing for the remaining particles to effectively apply function evaluations to global search. SG-PSO is a method of dynamically balancing local and global search.

As an extension to the SG-PSO algorithm, we developed the mixed SG-PSO (MSG-PSO) algorithm to further facilitate the balance between global and local search. In the MSG-PSO algorithm, particles are given individual accuracy settings, resulting in a mixture of rough and precision searching particles. First, the rough searchers find promising locations in domain space and stop, then precision searchers perform further optimization. The MSG-PSO algorithm was observed to show improvements over the SG-PSO algorithm.

We suggest the following possible avenues of research concerning SG. First, investigations into basing the SG required accuracy parameter on the number of dimensions of the search domain space or dynamically adapting the required accuracy parameter appear promising. Second, since the SG and VBR, performed better on different types functions, it might be possible to combine the SG and VBR methods. A very basic investigation of this idea was conducted in Section 4.6.2.

5.3 Future Directions of PSO Research

We have introduced the VBR and SG methods for PSO in this thesis. We will use this final section to give our opinions about some of the exciting future directions in particle swarm research. One of the most interesting directions of current PSO research is into discovering the essential elements of the swarm. This research could lead to simplifications of the PSO algorithm.

An example is the fully informed particle swarm (FIPS) [39]. In FIPS, the neighborhood best of the PSO velocity equation is eliminated. It is found that the PSO velocity equation can be simplified to one term where the particles head to an average of the positions of the particles in the neighborhood.

Also proceeding along the lines of simplification is the Bare Bones Swarm (BBS). In [29], Kennedy studies the distribution of points samples by PSO about the mean of the personal best and global best points. A plot of this distribution reveals that the points samples are almost (but not quite) Gaussian. The leads to the BBS, where the whole velocity update equation is eliminated and points about the median of the personal best and global best positions are simply sampled from a Gaussian distribution. Unfortunately, the BBS (and its variants) did not work as well as the standard PSO. Still, we believe that this is an important direction of research. Further research is continuing in this direction [30, 31] and may discover a simpler method of sampling points that might allow the velocity update equation to be eliminated or improved upon.

A unique characteristic possessed by PSO, but not other common evolutionary algorithms, is its memory. This suggests that PSO might excel over other optimization methods where a memory is

beneficial. On such application is multiobjective optimization (MO), where one is after the many Pareto optimal solutions. PSO has been applied to MO in [12]. It is likely that other applications where PSO's memory is beneficial will be discovered through further research.

Appendix A

List of Publications

Peer-Reviewed Journals

[1] K.J. Binkley and M. Hagiwara. Applying self-adaptive evolutionary algorithms to two-dimensional packing problems using a four corners' heuristic. *European Journal of Operational Research*, 183(3):1230–1248, 2007.

[2] K.J. Binkley and M. Hagiwara. Balancing exploitation and exploration in particle swarm optimization: Velocity-based reinitialization. *Transactions of the Japanese Society for Artificial Intelligence*, 23(1):27–35, 2008.

[3] K.J. Binkley and M. Hagiwara. The stop and go particle swarm: A swarm with a dynamically adapting population size. *Transactions of the Japanese Society for Artificial Intelligence*, 23(3):234–244, 2008.

[4] K.J. Binkley, K. Seehart, and M. Hagiwara. A study of artificial neural network architectures for othello evaluation functions. *Transactions of the Japanese Society for Artificial Intelligence*, 22(5):461–471, 2007.

International Conferences

[1] K.J. Binkley and M. Hagiwara. Particle swarm optimization with area of influence: increasing the effectiveness of the swarm. *Proceedings 2005 IEEE Swarm Intelligence Symposium*, pages 45–52, 2005.

Bibliography

[1] Isi web of knowledge. Website, 2008. `http://apps.isiknowledge.com`.

[2] P.J. Angeline. Using selection to improve particle swarm optimization. *Evolutionary Computation Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 84–89, 1998.

[3] H.G. Beyer and H.P. Schwefel. Evolution strategies–A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[4] K.J. Binkley and M. Hagiwara. Balancing exploitation and exploration in particle swarm optimization: Velocity-based reinitialization. *Transactions of the Japanese Society for Artificial Intelligence*, 23(1):27–35, 2008.

[5] K.J. Binkley and M. Hagiwara. The stop and go particle swarm: A swarm with a dynamically adapting population size. *Transactions of the Japanese Society for Artificial Intelligence*, 23(3):234–244, 2008.

[6] T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.

[7] M. Braik, A. Sheta, and Ayesh A. Particle swarm optimisation enhancement approach for improving image quality. *International Journal of Innovative Computing and Applications*, 1(2):138–145, 2008.

[8] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *Proceedings of the 1999 Congress on Evolutionary Computation*, 3, 1999.

[9] M. Clerc. TRIBES-A parameter free particle swarm optimizer, 2003.

[10] M. Clerc. Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Technical Report CSM-460, Department of Computer Science, University of Essex, 2006.

[11] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[12] C.A.C. Coello, G.T. Pulido, and M.S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, 2004.

[13] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[14] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the 2000 Congress on Evolutionary Computation*, 1, 2000.

[15] R.C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. *Proceedings of the 2001 Congress on Evolutionary Computation*, 1:81–86, 2001.

[16] R.C. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. *Proceedings of the 2001 Congress on Evolutionary Computation*, 1, 2001.

[17] D.B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.

[18] N. Franken and AP Engelbrecht. Comparing PSO structures to learn the game of checkers from zero knowledge. *The 2003 Congress on Evolutionary Computation*, 1, 2003.

[19] N. Franken and AP Engelbrecht. PSO approaches to coevolve IPD strategies. *The 2004 Congress on Evolutionary Computation*, 1, 2004.

[20] D.E. Goldberg et al. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co Reading, Mass, 1989.

[21] X. Hu and R.C. Eberhart. Adaptive particle swarm optimization: detection and response to dynamic systems. *Proceedings of the 2002 Congress on Evolutionary Computation*, 2, 2002.

[22] X. Hu, R.C. Eberhart, and Y. Shi. Particle swarm with extended memory for multiobjective optimization. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 193–197, 2003.

[23] X. Hu, R.C. Eberhart, and Y. Shi. Swarm intelligence for permutation optimization: a case study of n-queens problem. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 243–246, 2003.

[24] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer. *Proceedings of the 2003 Congress on Evolutionary Computation*, 2, 2003.

[25] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer for dynamic optimization problems. *1st European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments*, 2004.

[26] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(6):1272–1282, 2005.

[27] J. Kennedy. The particle swarm: social adaptation of knowledge. *IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[28] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. *Proceedings of the 1999 Congress on Evolutionary Computation*, 3, 1999.

[29] J. Kennedy. Bare bones particle swarms. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 80–87, 2003.

[30] J. Kennedy. Dynamic-probabilistic particle swarms. *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 201–207, 2005.

[31] J. Kennedy. In Search of the Essential Particle Swarm. *IEEE Congress on Evolutionary Computation*, pages 1694–1701, 2006.

[32] J. Kennedy and R.C. Eberhart. Particle swarm optimization. *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 4, 1995.

[33] J. Kennedy and R.C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 5:4104–4108, 1997.

[34] J. Kennedy and R. Mendes. Population structure and particle swarm performance. *Proceedings of the 2002 Congress on Evolutionary Computation*, 2, 2002.

[35] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[36] J.R. Koza. *Genetic programming*. MIT Press Cambridge, Mass, 1992.

[37] W.B. Langdon and R. Poli. Evolving Problems to Learn About Particle Swarm Optimizers and Other Search Algorithms. *IEEE Transactions on Evolutionary Computation*, 11(5):561–578, 2007.

[38] R. Mendes. *Population Topologies and Their In fluence in Particle Swarm Performance*. PhD thesis, Universidade do Minho, 2004.

[39] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

[40] L. Messerschmidt and AP Engelbrecht. Learning to play games using a PSO-based competitive learning approach. *IEEE Transactions on Evolutionary Computation*, 8(3):280–288, 2004.

[41] K.E. Parsopoulos and M.N. Vrahatis. Particle swarm optimization method in multiobjective problems. *Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607, 2002.

[42] S. Pasupuleti and R. Battiti. The gregarious particle swarm optimizer (G-PSO). *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 67–74, 2006.

[43] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, 2004.

[44] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. *Evolutionary Computation Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 69–73, 1998.

[45] Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. *Proceedings of the 1999 Congress on Evolutionary Computation*, 3, 1999.

[46] R. Storn and K. Price. Differential Evolution–A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[47] T.J. Su, H.C. Wang, and J.W. Liu. Particle Swarm Optimization for Image Noise Cancellation. *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 1, 2007.

[48] P.N. Suganthan. Particle swarm optimizer with neighborhood operator. *Proceedings of the 1999 Congress on Evolutionary Computation*, 3:1958–1962, 1999.

[49] F. van den Bergh and AP Engelbrecht. A Cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

[50] J.S. Vesterstrom, J. Riget, and T. Krink. Division of labor in particle swarm optimisation. *Proceedings of the 2002 Congress on Evolutionary Computation*, 2, 2002.

[51] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[52] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.