Noname manuscript No.
(will be inserted by the editor)

A Novel Abstraction for Swarm Intelligence: Particle *Field* Optimization

Nathan Bell and B. John Oommen

Abstract Particle Swarm Optimization (PSO) is a popular meta-heuristic for black-box optimization. In essence, within this paradigm, the system is fully defined by a swarm of "particles" each characterized by a set of features such as its position, velocity and acceleration. The consequent optimized global best solution is obtained by comparing the personal best solutions of the entire swarm. Many variations and extensions of PSO have been developed since its creation in 1995, and the algorithm remains a popular topic of research. In this work we submit a new, abstracted perspective of the PSO system, where we attempt to move away from the swarm of individual particles, but rather characterize each particle by a field or distribution. The strategy that updates the various fields is akin to Thompson's sampling. By invoking such an abstraction, we present the novel Particle *Field* Optimization (PFO) algorithm which harnesses this new perspective to achieve a model and behavior which is completely distinct from the family of traditional PSO systems.

Key Words: *Particle Swarm Optimization, Meta-heuristic Optimization, Swarm Intelligence.*

1 Introduction

The efficient use of resources is a fundamental problem which permeates every aspect of our lives. Nearly every decision we make can be interpreted, in some way, to involve the concept of minimizing the "cost", or maximizing the "returns" of our actions. In fact, this concept can be considered to be a core challenge of life of any kind, as resources and energy are inherently limited, and must be used efficiently in order to ensure long-term survival and prosperity. Any form of "intelligence", then, *must* have the ability to deal with problems of this type.

A preliminary and very brief version of this paper was presented at *AAMAS'15, the 2015 International Conference on Autonomous Agents and Multiagent Systems*, in Istanbul, Turkey, in May 2015. The paper was nominated to be the *Best Student Paper* of the Conference. *Both the authors are extremely grateful to the anonymous Referees of the initial version of this paper, and in particular, to the anonymous "Referee 3", for for their valuable comments. Their comments significantly improved the quality of this paper.*

This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada: K1S 5B6. E-mail address: nathanbell.42@gmail.com.

Chancellor's Professor ; *Fellow: IEEE* and *Fellow : IAPR*. This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada: K1S 5B6. E-mail address: oommen@scs.carleton.ca. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway. This author is grateful for the partial support provided by NSERC, the Natural Sciences and Engineering Research Council of Canada.

Address(es) of author(s) should be given

As humans, we possess a strong natural ability to work with these problems. However, as we develop and progress, we are faced with increasingly complex problems which go far beyond our limitations. As a result, finding new tools and methods which can be applied to these problems is important for our continued advancement.

The field of function optimization presents a formalized framework for modelling and solving certain problems of this type. Given an "objective" function, which takes a number of parameters as its input, the goal is to find the combination of parameter values which returns the "best" value. This framework is abstract enough that a wide variety of different problems can be interpreted as being "function optimization" problems. In this work, for the sake of clarity, we will refer to "optimization" as the problem of *minimizing* a function.

The objective function of an optimization problem can be any function which maps any number of parameters to a single final resultant value. Because of this rather non-specific definition, we see that there are an extremely wide variety of distinct properties that these functions can have. This variety of properties presents a significant challenge when approaching function optimization. Many properties, such as discontinuities, multimodal complexity, and elaborate constraints on the input parameter values, render "standard" optimization methods impossible or infeasible, and so currently, new methods must be created and tailored to deal with functions with specific properties.

"Black-box" optimization is a category of optimization problems dealing with functions about which no information is available. The fact that these functions are totally *unknown* presents a unique challenge for optimization. Without any knowledge of the function itself, algebraic manipulations (such as computing the derivatives) are impossible. As a result of this, traditional analytical methods of discovering the optimum no longer apply. In fact, it is unknown whether or not a single global optimum exists at all, and it is impossible to verify whether any solution found is globally optimal. The task of optimization, then, must be approached from a different perspective. Finding the optimal value becomes a search process within an unknown, and often infinite, solution space.

There exist a wide variety of strategies for approaching this search problem, each making use of its own distinct principles and perspectives. One such strategy is Particle Swarm Optimization (PSO). Created in 1995 as the result of experiments modelling a "collision-less" bird-flocking swarm behavior, the PSO algorithm has remained a popular subject of research and development. The PSO algorithm, briefly surveyed in Section 2.2, consists of a population of extremely basic individuals or particles which "fly" through the solution space. Through communication, these basic individuals are able to find "good" solutions to the black-box optimization problem.

Many variations of the basic PSO algorithm have been proposed, seeking to improve the algorithm and introduce, in it, new behavioral aspects. Among these variants is the Bare Bones Particle Swarm (BBPS), which applies an abstraction to the behavior of the particles within the swarm. In this paper, we show that inherent in the BBPS algorithm, there exists a previously-unexplored opportunity for an additional level of abstraction. Indeed, we show that each particle can be abstracted by a field or distribution, and where the form/parameters of the distribution are updated by using a scheme akin to Thompson's sampling, leading to a completely new and unique perspective on particle swarm systems. Taking advantage of this new perspective, we propose a novel search algorithm known as Particle Field Optimization (PFO), which is rather fascinating.

The rest of the paper is organized as follows. First of all, in Section 2, we present a relatively brief overview of the field including that of black-box optimization and meta-heuristics in Section 2.1 and the PSO in Section 2.2. The paper then continues to the submission of the new PFO paradigm in Section 3. Section 4 details the experimental results obtained by testing our scheme and comparing it with a benchmark algorithm on a *set* of internationally-recognized benchmark functions. Section 5 concludes the paper and presents possible avenues for future work.

2 Survey of the Field

2.1 Black-box Optimization and Meta-Heuristics

When dealing with black-box optimization, the only way of gaining information about the objective function is to generate and evaluate potential solutions. By observing the results of these function evaluations, some limited information can be deduced about the function, which can then be used to guide the search process. The most straight-forward approach to this would be a basic, deterministic hill-climbing algorithm. A hill climbing algorithm simply generates a number of "adjacent" solutions to the current best-found solution, and updates that best-found solution if one of the adjacent solutions is better. The process then loops until no better adjacent solution is discovered. This approach will quickly guarantee convergence to some *locally* optimal solution, but is not well suited to the problem of black-box optimization, in general. The majority of real-life applications involve functions with very complex, multi-modal solution spaces, and since nothing is known about the function being optimized, it must be assumed that these complexities are present. On functions such as these, a hill-climbing algorithm will perform poorly compared to other methods, exploiting the local information but becoming stuck in local optima.

In order to perform well on unknown functions, it is essential for the scheme to avoid stagnating or converging to a suboptimal point. If we consider the hill-climbing algorithm, the opposite extreme would be an algorithm which simply generates, and evaluates, totally random solutions within the solution space. Clearly, this strategy would not become stuck in local optima as the hill-climbing strategy would. However, simply generating random solutions is no better than one which sequentially generates and tests every possible solution, and so this strategy is not feasible either, due to the existence of a potentially infinite number of points in the solution space.

We thus argue that neither the extreme *exploitation* phase of a hill-climber, nor the extreme *exploration* phase of generating totally random solutions, are suitable for the problem of black-box optimization. To effectively and efficiently search the unknown space of a black-box function, an algorithm must strike a happy medium between these two extremes. *Exploitation* is necessary to find the locally optimal solution in some general area, and *exploration* is necessary to escape local optima and locate more areas for potential exploitation. This strategy of balancing exploitation with random exploration describes a category of algorithms known as "Meta-Heuristics".

The term "meta-heuristic" is a combination of the modifier "meta" ("beyond"), and the word "heuristic" ("proceeding to a solution by trial-and-error or by rules that are only loosely defined"). In general, a meta-heuristic algorithm is one which combines randomization with a higher-level guiding process to perform a search of some kind. This "higher guiding process" is what differentiates a meta-heuristic method from a

simple heuristic method. The heuristic process within a meta-heuristic is modified by this guiding process, thus elevating the behavior to be *beyond* a simple heuristic.

When investigating for new strategies for solving problems, nature provides a constant source of inspiration. These can range from algorithms modelling the process of evolution, to algorithms modelling river formation, and further to those that model the behavior of swarms of creatures. Because of this, there exist many meta-heuristic search and optimization algorithms which take inspiration from, or attempt to model, these natural phenomena. In fact, nearly all nature-inspired algorithms can be classified as meta-heuristics, due to the inherent randomness and necessity for adaptive behavior.

Of particular interest to us in this work, are those algorithms that model so-called "swarming" behavior. In nature, there exist many examples of social creatures that cooperatively act together and display a behavior which, as a whole, is beyond the individual capabilities of the creatures themselves. This phenomenon is known as "Swarm Intelligence" (SI), and provides a very interesting perspective for problem solving. In an SI system, a population of rudimentary or primitive "individuals" work and interact with one another in order to solve a higher-level problem beyond the capabilities of any single individual.

2.2 Particle Swarm Optimization

The core PSO algorithm consists of a population of particles, which act according to individual behavior and influence each other via communication. A PSO particle i consists of a position $\mathbf{X_i}$ within the solution space, a velocity $\mathbf{V_i}$, and a personal best-found point $\mathbf{P_i}$. The algorithm begins with an initialization stage, which is followed by an iterative simulation stage.

During initialization, the population of particles is created and their positions and velocities, are assigned random values. Following this, the simulation stage begins, and it runs indefinitely unless a termination criterion is met or the algorithm is manually stopped. At each step of the simulation, particles update their velocities, move accordingly, and evaluate their new location. Their corresponding personal best solution is also updated, if necessary. A particle i updates its velocity $\mathbf{V_i}$, and position $\mathbf{X_i}$, according to the equation:

$$\mathbf{V_i} = \omega\mathbf{V_i} + \mathbf{U}[0, \phi_p] \otimes (\mathbf{P_i} - \mathbf{X_i}) + \mathbf{U}[0, \phi_g] \otimes (\mathbf{P_g} - \mathbf{X_i})$$
$$\mathbf{X_i} = \mathbf{X_i} + \mathbf{V_i},$$

where ω is an inertial weighting parameter, ϕ_p and ϕ_g are acceleration weighting parameters, $\mathbf{U[a, b]}$ represents a uniform random distribution between values a and b, $\otimes$ is the element-wise multiplication of vectors, and $\mathbf{P_g}$ is the global best particle.

Once a termination criterion is met, or the simulation is manually stopped, the position of the global best particle $\mathbf{P_g}$, which represents the best solution found by the algorithm, is returned as the output.

2.3 Variations and Extensions of the PSO Algorithm

Due to the simple and flexible nature of the PSO algorithm, countless variations and extensions have been proposed since it's initial creation, and it remains an active topic of research today. The research and development of the PSO algorithm is also quite varied, with many distinct strategies and approaches being explored.

Some of these strategies seek to improve existing components of the PSO algorithm [4], [8] [11], or to introduce adaptive behaviors of various levels of sophistication over the core algorithm to improve useability [15], [17], [22]. Other strategies include hybridizing the PSO algorithm with other population-based ideas such as genetic algorithms [1], [5], [10] [16].

In addition to the further development of the PSO algorithm, there have been many attempts made at analyzing the behavior of the core PSO algorithm. The high level behavior of the PSO algorithm is very complex and difficult to analyze, and so many analyses consider simplified versions of the algorithm, with the motivation of eventually building back towards the full PSO [2], [8], [13], [14]. In addition to this, analysis has shown that the PSO algorithm displays a number of biases, including a bias towards the center of the initialization area [12], and a bias favoring movement parallel to the axes[1] [19].

2.4 Bare Bones Particle Swarm

The Bare Bones Particle Swarm (BBPS) algorithm [6] is a PSO variant which seeks to emulate the high-level behavior of the basic PSO algorithm, while simultaneously using a much simpler particle update strategy. Observations of the basic PSO system suggested that very similar behavioral patterns could be achieved without the complicated velocity and acceleration components. Further, they analyzed observations of a *single* particle using the standard velocity strategy, when the system attained a state of swarm stagnation. This scenario occurs when the *Personal* Best Point (BP) and Global BP remain constant. These observations produced a histogram of the points evaluated characterized by a distinct bell-curved shape [6]. This phenomenon led the authors of [6] to the conjecture that rather than dealing with the particle velocities and accelerations, each particle could be updated by merely sampling a random distribution constructed to approximate this histogram.

The BBPS algorithm entirely eliminates the velocity component of the particle and merely determines the particle's next position by sampling a Gaussian random distribution. This Gaussian distribution is constructed and sampled individually for each dimension. The mean of this distribution is set to be the midpoint between the global best-found point and the particle's own best-found point, and the standard deviation is set as the absolute distance between the global and personal BPs for each dimension. Given a particle i with position $\mathbf{X_i}$, whose personal BP is $\mathbf{P_i}$ and for which the global BP is $\mathbf{P_g}$, the particle's next position is determined as:

$$\mathbf{P_m} = \frac{\mathbf{P_i} + \mathbf{P_g}}{2},$$
$$\mathbf{X_i} = \boldsymbol{\mathcal{N}}(\mathbf{P_m}, \boldsymbol{\sigma^2}),$$

where $\boldsymbol{\mathcal{N}}$ represents a function which creates a Gaussian random vector dimension-by-dimension, centred on $\mathbf{P_m}$. $\boldsymbol{\sigma}$ is a diagonal matrix of the standard deviations (and the squared quantity represents the corresponding variances) for each dimension, set to be the absolute values of the difference between the vectors $\mathbf{P_i}$ and $\mathbf{P_g}$ for each dimension.

This update strategy is marginally more abstract than the basic PSO velocity strategy, since the particles no longer "fly" through the space, but the general behavior of the swarm remains the same. At a higher level,

[1] The issue of the biases present in PSO algorithms was further explored in the Masters thesis of the first author. After a more detailed investigation, we discovered that this is not as pertinent in the PFO as in the PSO.

the algorithm still involves a population of particles which move through the solution space, and which are influenced by the memories of their BPs and the communication of the global BPs.

3 The Particle Field Optimization (PFO) Algorithm

3.1 The Need for Further Abstraction

In this section we argue the need for a further level of abstraction, and how the PSO paradigm can be extended to yield the PFO.

Gleaning insight into the BBPS model: The BBPS model discards the velocity component of the basic PSO model and introduces a more abstract method for updating the positions of the particles. However, if we take a closer look at the consequences of this change, it is possible to take the abstraction to a level that is even higher, whence one discovers a completely new perspective and new avenues for development.

To motivate this abstraction, we first consider the issue of what constitutes a single particle in these two models. In the basic PSO model, an individual particle consists of: A current position, a current velocity, and a personal best-found position. Each of these components is required in order to determine the subsequent position of the particle for the next iteration. The particle's position in the next iteration depends upon its current position and the particle's velocity. The updated velocity, in turn, depends on the particle's current velocity, the particle's personal best-found position and the population's global best-found position. Each component of the particle contributes to the update function and must be maintained for use in the next iteration.

In the BBPS model, the velocity term is removed and an individual particle consists of only a current position and a personal best-found position. However, unlike the basic PSO model, updating the particle does *not* require both of these components. Specifically, the position of the particle in the next iteration is independent of the position of the particle in the current iteration. When updating a particle, the next position is generated by *sampling* a Gaussian random distribution, which, in turn, is constructed using the particle's personal best, and the communicated global best points. After creation, the position is used for only *one* purpose, i.e., the updating of the particle's personal best-found point. The global best point is, thereafter, updated using the population's personal best-found points, and so does not rely *directly* on the particle's current position either.

Eliminating the particle's position component entirely: As a result, it is possible to remove the particle's position component from the model entirely. Rather than storing and maintaining the particle's current position, we can modify the update operations to work directly with the particle's personal best point while maintaining equivalent behavior. A new point is generated by sampling the constructed random distribution. If the new point has a better value than the particle's personal best-found point, the best-found point is set equal to the new point. The only difference is that this newly generated point is temporary, rather than being maintained for the next iteration.

Modeling the "position" by a random *distribution*: After removing the particle's current position component, an individual particle now consists of only a personal best-found position. The particle no longer exists as an explicit point in space. Conceptually, the particle's personal BP is a memory that the particle maintains of it's own best evaluated point, and the global BP can be thought of as the collective memory of the population. These two "memories" define the random distribution used to update the particle. Though the particle

no longer exists as an explicit point in space, we can, instead, think of the particle's "position" in that space as being defined by the random *distribution* itself. This is because this distribution represents the probability field of possible positions for the particle. From this perspective, the particle exists as a random *field* defined by its own memory and the collective memory of the population.

With the new concept of what the population individuals are, the high-level perspectives of the algorithm change drastically. The metaphor of a swarm of particles "flying" through space no longer aptly describes the high-level concept of the algorithm. Rather, the algorithm now consists of a population of "particle fields" which move throughout the space in a different way. Because the "positions" of these "particle fields" are defined as random distributions, "evaluating" a "particle field's" current "position" is non-deterministic, and so these "particle fields" do not necessarily have to "move" to explore new points. These "particle fields" remain "stationary" in the space until either the individual's personal best point changes, or the population's global best point changes. This population of "particle fields" can, itself, be perceived as a random field of particles defined as a mixture distribution made up of each individual distribution. This population-level distribution can be thought of as an abstract representation of a particle swarm, representing a probability distribution of all possible particle locations for the next iteration. This perspective of the high-level concept is significantly different than the BBPS algorithm, although their behaviors are analogous. However, with this new perspective, it is possible to explore new directions in improving or changing the behavior of the algorithm.

A consequence of the above assertions is the following observation: The behavior of the algorithm based on this abstracted model is unchanged. However, interestingly, the metaphors and concepts of the original PSO can be re-formulated to create the PFO, as we explain below.

3.2 How the PFO Abstraction is Affected

Modeling using *Mixture* distributions: With this new perspective of the BBPS algorithm, we can begin taking steps toward a new algorithm using this abstracted philosophy. In this abstracted BBPS model, each individual in the population would generate and evaluate a single new point per iteration so as to maintain the concept of an individual representing a single particle. However, it is not necessary to maintain this metaphor in our new model, and so we can approach the generation and evaluation of candidate solutions from a distinct perspective.

Rather than have each individual directly representing a candidate solution in and of itself as in traditional PSO algorithms, we use the population in a more indirect way to explore the solution space. Collectively, the population represents a complex, finite mixture distribution made up of an underlying set of simple multivariate Gaussian distributions, which are defined by the particle field individuals. Traditionally, each individual would generate, and evaluate a single new solution. However, this population distribution can be used to guide the search in a different way.

Generating candidate solutions: Candidate solutions are instead generated by sampling this population-level distribution. Although the population-level distribution is complex, the sampling process is a simple matter of sampling one of the underlying component distributions, selected at random. In the context of the population, this means randomly selecting a particle field individual and then sampling the multivariate Gaussian distribution defining that particle field's "position".

Once the candidate solution points have been generated and evaluated, each particle field individual in the population is updated. Each individual is updated using the candidate solutions generated from that individual's own distribution. If a candidate solution is better than the individual's own personal best point, the personal best point is set equal to that solution. In this way, the population defines a random distribution which guides the search and generation of candidate solutions, which are then used to update the population and redefine the search area for the next iteration. This process of updating the particle fields is demonstrated on both the individual and population-levels in Figures 1 and 2 respectively.

This concept of the population guiding the search leads us to the next step toward a new, distinct algorithm. Because the complex distribution created by the population is a finite mixture distribution, it is a simple matter to apply a weighting scheme to the distribution. By weighting the contribution of each particle field to the population distribution, it is possible to incorporate additional information into the search process, independent of the underlying PSO processes.

Using different relative sizes: Finally, since the population of the particle field individuals no longer directly represents candidate solutions, it is no longer necessary that the number of candidate solutions generated and evaluated at each iteration equals the size of the population. Because of this, it is possible to further modify the behavior by using different relative population and candidate solution point pool sizes. Due to the nature of the population distribution, changing the number of particle field individuals in the population effects the "resolution" of this distribution. Consequently, larger populations will result in a more complex population distribution, while a smaller population will result in a less complex distribution. Additionally, since particle field individuals are only updated if they are used to generate candidate solutions, individuals in a smaller population will be more likely to be updated. In short, a smaller population will lead to faster convergence, but a larger population may better explore the solution space.

3.3 Implementing the PFO Abstraction

With these changes taken into account, we have now moved away from the traditional PSO paradigm and arrived at a new, distinct algorithm, which will be hereafter referred to as Particle *Field* Optimization (PFO). This algorithm consists of a population of "particle field" individuals and a "point pool" of candidate solution points. The population of particle field individuals uses PSO principles to guide the search of the solution space, which is carried out by generating and evaluating, the pool of candidate solution points. Analogous to traditional PSO algorithms, the PFO algorithm consists of an initialization phase and a simulation phase which loops until some termination criterion is met, at which point the best solution found by the algorithm is returned as output. As its parameters, the algorithm takes an initialization range, a population size and a pool size. The population size parameter specifies the number of particle field individuals which form the population used to guide the search. The pool size parameter specifies the number of candidate solutions to be generated and evaluated, at each step of the simulation. A weighting function may be specified, which weights the contribution of each individual to the population-wide distribution.

The initialization phase initializes the population of particle field individuals. A particle field individual stores only a personal best-found position, and so the initialization of these individuals is simple. Each individual

Fig. 1: The update process of an individual particle field in a single dimension.

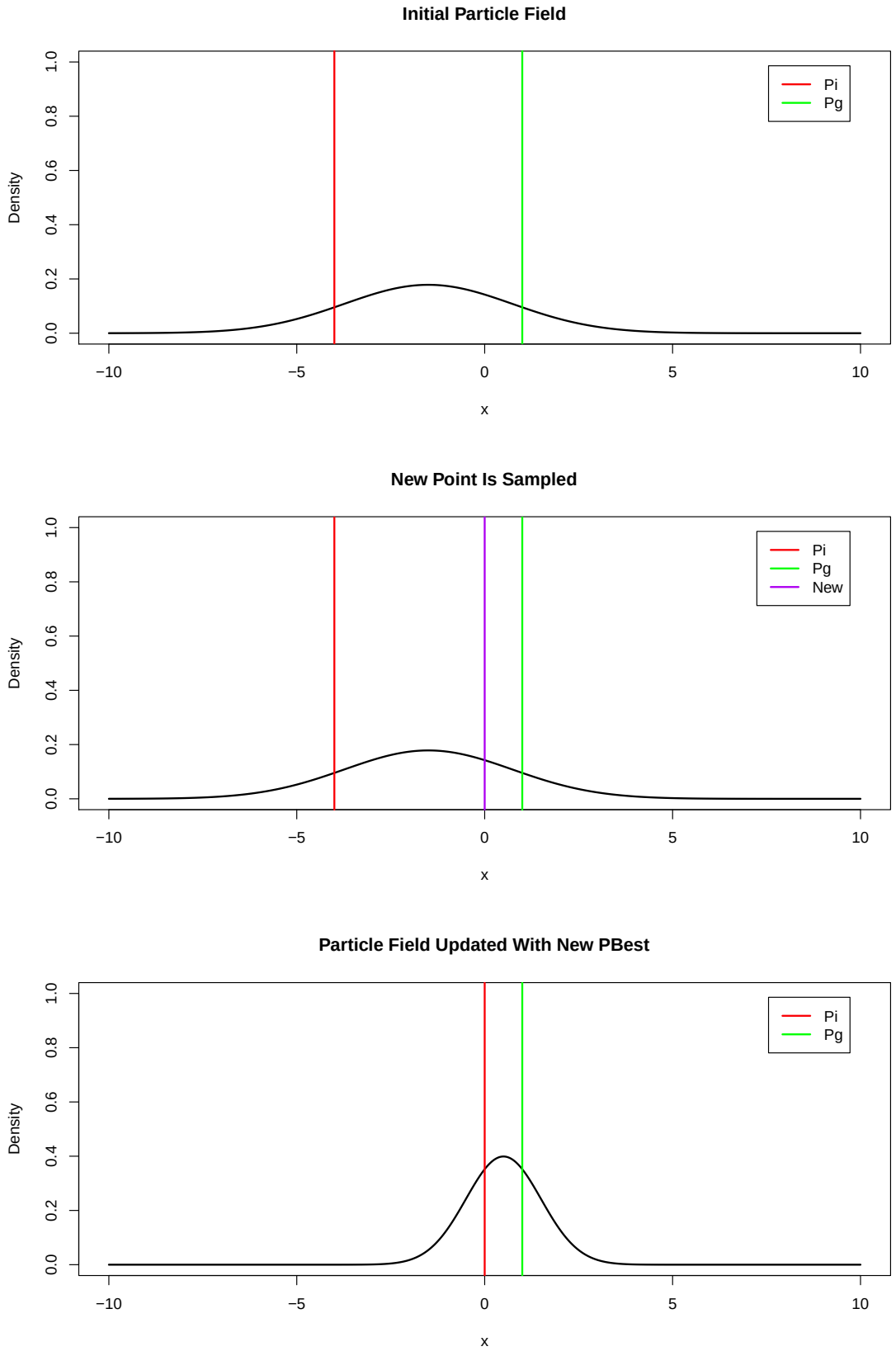
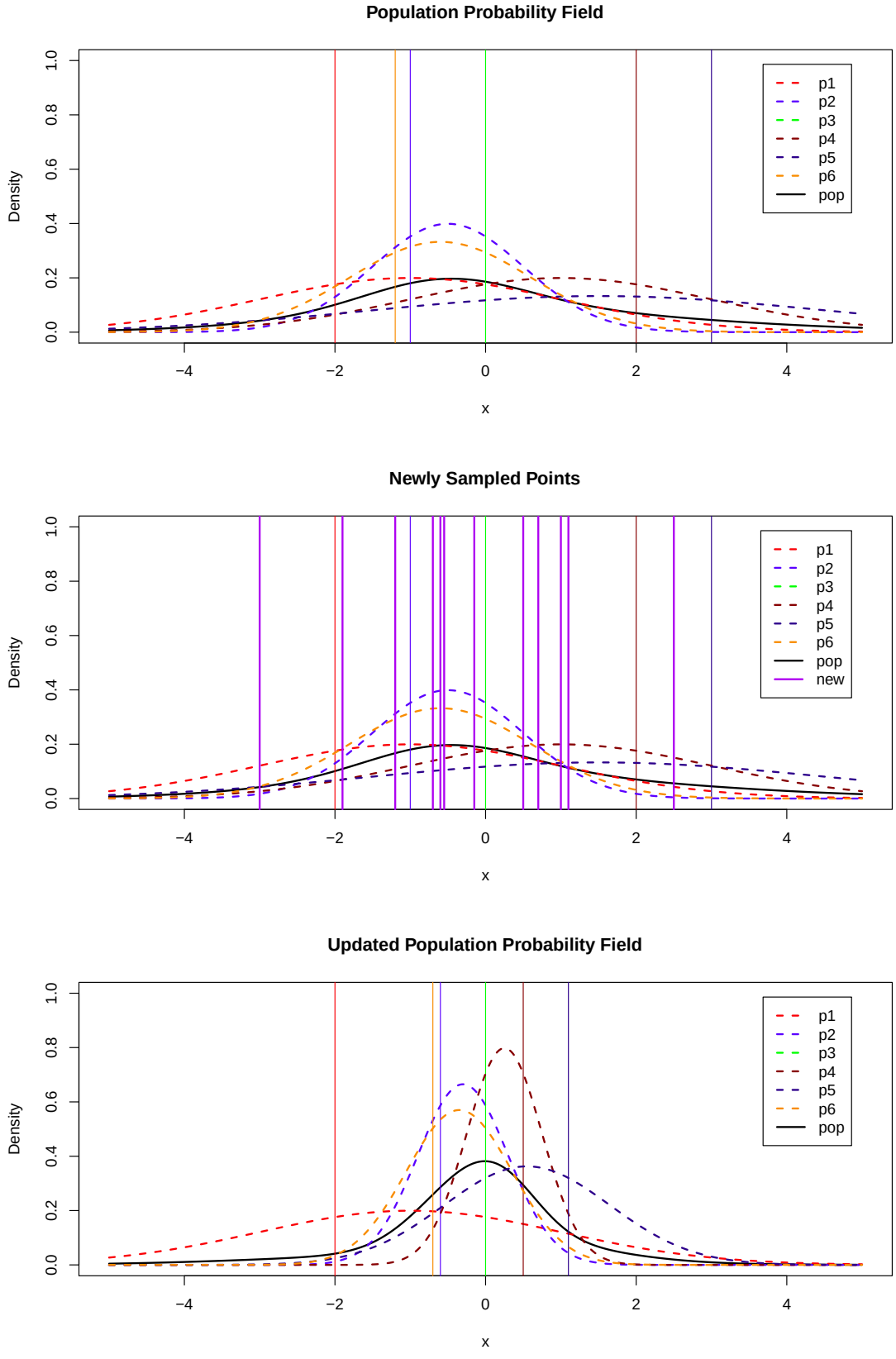


Fig. 2: The update process of the particle field population in a single dimension. Each of the distributions p_1 through p_6 represents a particle field individual, and the distribution "pop" represents the resulting population distribution.



is assigned an initial personal best point by sampling a uniform random distribution defined by the initialization range.

The simulation phase loops until a termination criterion is met, typically the maximum number of iterations. Each iteration consists of two phases. In the first phase, candidate solutions are generated, and for each point in the point pool we do the following: A particle field individual is randomly selected from the population and then the point is generated by sampling the distribution defined by that individual. Given a particle field with personal best point $\mathbf{P}_i$ and for which the global best (or neighbourhood best), point is $\mathbf{P}_g$, the position of the candidate solution point, $\mathbf{c}$, is determined according to:

$$\begin{aligned} \mathbf{P_m} &= \frac{\mathbf{P_i} + \mathbf{P_g}}{2}, \\ \mathbf{c} &= \boldsymbol{\mathcal{N}}(\mathbf{P_m}, \boldsymbol{\sigma^2}). \end{aligned}$$

In the above, $\boldsymbol{\sigma^2}$ is chosen to be a diagonal matrix such that its diagonal elements are the d^{th}-components of the corresponding vectors, i.e., ${\sigma^2}_{dd} = \mathbf{P_{id}} - \mathbf{P_{gd}}$ for each dimension d. Once the candidate solution has been generated, the objective function is evaluated, using this generated point as its input, in order to assign it a value. After each candidate solution in the point pool has been generated, the second phase begins. In the second phase, the population of particle field individuals is updated. Each individual updates its own best-found point using the set of candidate solutions generated from its own distribution. Each individual selects the best point from the set of associated candidate solutions. If the best associated candidate solution is better than the individual's personal best-found point, the individual sets its personal best-found point to be equal to that candidate solution point. The pool of candidate solutions is then "emptied", and the simulation continues to the next iteration.

Once the termination criterion has been met, the global best-found point is returned as output of the algorithm. The formal algorithm is presented in Algorithm 1.

Remarks: From an algorithmic perspective, one primary enhancement in the BBPS to create the PFO is to change the representation of "each particle into a field" using which we can "select each particle (field) with a given probability" (Line 28, Algorithm 1). This change allows us to accentuate the difference between the population size and the pool size. In other words, in the simplest sense, the PFO algorithm can be considered be similar to the BBPS that uses the uniform weighting scheme and where the pool size is the same as the population size. Expressed differently, the PFO can be seen to be a generalization of the BBPS with the additional setting parameters, i.e., the weighting scheme and the pool size. The flexible pool size that ranges from unity to "infinity", as explored in some existing algorithms, provides the algorithm the property to be more flexible, and to be solvable in a parallelized manner. That being said, we would not, strictly speaking, merely consider the PFO to be a *generalization* of BBPS. Rather, we suggest that it is an *abstraction* that goes beyond the BBPSs behaviour. To be more specific, in the case that the weighting scheme is uniform, and the particle field population size is equal to the pool size, the PFO will still behave somewhat differently, since it is possible for each particle field individual to evaluate multiple, or no points.

Algorithm 1 Particle Field Optimization Algorithm

Input:
Function $f()$ to be minimized
Initialization range $lbound$, $ubound$
Particle field population size n_{pop}
Candidate solution point pool size n_{pool}
Weighting function $w()$
Output:
Point $\mathbf{P_g}$ representing best-found solution
Method:

```
create particle field population P with size n_pop
create candidate solution point pool C with size n_pool
for each particle field i ∈ P do
   P_i ⇐ U[lbound, ubound]
   if f(P_i) < f(P_g) then
      P_g ⇐ P_i
   end if
end for

while termination criterion not met do
   for each particle field i ∈ P do
      S_i ⇐ ∅
   end for
   for each candidate solution point c ∈ C do
      select particle field i ∈ P with probability w(i) / Σ_{p∈P} w(p)
      c ⇐ N( (P_i + P_g)/2, |P_i − P_g| )
      S_i ⇐ S_i ∪ c
   end for
   for each particle field i ∈ P do
      choose point c_min from S_i which minimizes f(c_min)
      if f(c_min) < f(P_i) then
         P_i ⇐ c_min
         if f(P_i) < f(P_g) then
            P_g ⇐ P_i
         end if
      end if
   end for
end while
return P_g
```

3.4 Analogous Paradigms

Before we embark on a rigorous experimental verification of the PFO paradigm, we believe that it is expedient to mention some unexpected similarities between the newly-abstracted phenomenon and two existing techniques, namely, Bayesian Optimization and Thompson's Sampling.

3.4.1 Bayesian Optimization

Bayesian Optimization is a strategy for black box optimization which, briefly stated, models the black box function as a random distribution [18]. In a general sense, the algorithm begins with a prior distribution over the function to be optimized. This prior distribution is then used to determine an acquisition function, which, in turn, is sampled to choose the next point that has to be explored. The latter point is then evaluated, and is used to form the *a posteriori* distribution to be used in the next iteration. Theoretically, Bayesian Optimization represents an extremely powerful scheme for gleaning the most possible information from the evaluated solutions. However, in practice, the computations required to integrate and update these distributions

is prohibitively expensive. Consequently, Bayesian Optimization is better suited to situations where the costs associated with evaluating a solution are extremely high.

From a comparative perspective, if we more thoroughly investigate the general procedure of the PFO algorithm, we are able to discover some similarities to the Bayesian Optimization philosophy. Quite briefly, in both algorithms, the solution space is explored by sampling some prior function, and these sampled points are subsequently utilized to convert the *a priori* distribution into the corresponding *a posteriori* distribution, used in the next iteration. The primary difference between these two algorithms is the manner by which these distributions are created.

3.4.2 Thompson's Sampling

Thompson's sampling is a heuristic approach to the "multi-armed bandit" problem [20]. The multi-armed bandit presents a set of actions with unknown rewards, and requires that these actions be chosen to optimize the cumulative reward.

The unknown rewards of each action can be modeled as random distributions with unknown means. The unknown mean of a random distribution can, itself, be modeled using a random distribution. Thompson's sampling uses the rewards returned by choosing the actions, so as to model the distribution of the means of the rewards for each action. Each "mean distribution" is then sampled, and the action corresponding to the highest sampled mean reward is chosen.

Although there is a connection between the two, the comparative aspects between Thompson's Sampling and the PFO are not directly obvious. Indeed, the particle field individuals guide the search by defining a distribution to be sampled. This distribution represents the particle field's "belief" of where the next point should be evaluated. When generating a new point, an individual is chosen and a point is generated by sampling that individual's distribution. The *action* of choosing a particular particle field returns a *reward* in the form of a newly-explored solution point. The weighting schemes that are applied to this process guide the choice of the particle field individual. One can thus see a similarity between the two paradigms. In fact, we believe that it would be possible to apply a Thompson's sampling process instead of the PFO's weighting schemes, which would provide a rich avenue for future work.

We conclude this section by stating that these techniques are quite distinct from the PSO from which the PFO originates. It is thus interesting to observe that a relatively simple abstraction can build connections between such disparate approaches. This further demonstrates the potential of the PFO abstraction to provide new avenues of development, as we can now consider methods to apply knowledge from these previously unrelated schemes into the PSO/PFO.

4 Experimental Verification of the PFO

4.1 The Experimental Test Bed

In order to accurately measure the performance of a black-box optimization algorithm, empirical testing must be carried out on a variety of test functions. The test suite selected to evaluate the PFO algorithm consists of a variety of different test functions commonly used for evaluating PSO related algorithms. This suite includes

Table 1: Test Function Suite

Function	Dim	Formula
Rosenbrock	2	$(1-x_1)^2+100(x_2-x_1^2)^2$
Schaffer F6	2	$0.5+\frac{sin^2(\sqrt{x_1^2+x_2^2})-0.5}{[1+0.001*(x_1^2+x_2^2)]^2}$
Sphere	10	$\sum_{i=1}^{d} x_i^2$
Ackley	10	$20+e-20e^{-\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}}-e^{-\frac{1}{d}\sum_{i=1}^{d}cos\left(2\pi x_i^2\right)}$
Rastrigin	2	$10d+\sum_{i=1}^{d}\left[x_i^2-10cos(2\pi x_i)\right]$
	10	
	20	
Griewank	2	$1+\frac{1}{4000}\sum_{i=1}^{d}x_i^2-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)$
	10	
	20	

the Rosenbrock, Shaffer F6, Sphere, Ackley, Rastrigin and Griewank functions. The full suite, including the choice of the dimensionality for each function, is presented in Table 1.

4.2 Overview of Testing Strategy

To demonstrate the power of the PFO paradigm, we conducted our testing with the goal of discovering the range of possible behavioral patterns, and the effects of the parameters on these patterns. Of course, it was infeasible to achieve the testing against *all the known* meta-heuristics and using *all* the available, recorded test functions. This decision was made in light of the "No Free Lunch Theorem (NFL) for Optimization" which suggests that the theoretical performance of all meta-heuristic optimization methods are equal when averaged over *all* possible functions [21]. In this context, it should be pointed out that the NFL thoerem, in and of itself, is possibly too generic or theoretical to apply so strictly to reality, and so, most problems encountered in the real world share many properties (such as continuity or symmetry), which can circumvent the conclusions of the NFL theorem. Consequently, we believe there is more to be gained by exploring the range of behaviors of a new optimization algorithm, than by attempting to find the best performing results. In reality, understanding how the behavior of the algorithm varies with parameter values is valuable when considering the practical applications of the algorithm in any specific application domain. Our goal with the testing phase of the PFO was to explore the possible ranges of the algorithm's behavioral patterns, and the effects of the parameters on these patterns. Our goal was not to determine the set of parameters that worked best on any specific data set.

The PFO algorithm can be configured using three primary parameters which modify the algorithm's behavior. These primary parameters are the "pool size" parameter, the "population size" parameter and the weighting scheme used by the algorithm. It is important to recall that the number of function evaluations performed by

the PFO algorithm is dictated by the pool size parameter, rather than the population size parameter. For this reason, a single value was chosen for the pool size parameter for each function in order to ensure a consistent number of function evaluations. Testing was carried out on the PFO algorithm in order to investigate the range of behaviors possessed by the algorithm, and the effects of each parameter on that behavior.

Weighting schemes were chosen to explore different options for prioritizing the choice of particle field individual based on different metrics and are described here briefly. The "None" weighting simply weighted each individual equally, representing a case where an equal weighting was applied across the population. The "PBest" scheme weights each individual proportional to their current personal best found point. The "Average" scheme weights each individual proportional to the average of all points generated in the previous iteration. The "Relative Average" scheme weights each individual proportional to the average of all points generated in the previous iteration *relative* to the individual's personal best found point. The "Percent" scheme weights each individual proportional to the percentage of points generated in the previous iteration better than the individual's personal best found point. A detailed justification of the parameter values and weighting schemes chosen for testing can be found in [3], and is omitted here in the interest of brevity.

The PFO algorithm was created by introducing a number of changes to the BBPS algorithm. As a result, the BBPS algorithm provides the best baseline point of comparison for assessing the effects of these changes. In addition to this, the BBPS algorithm takes only one parameter, which is the population size parameter. For each test, this parameter was set equal to the corresponding PFO pool size in order to ensure that the number of function evaluations remained consistent between the algorithms. Because of this, the BBPS algorithm required no parameter tuning itself, and so provides a very straightforward and dependable baseline of comparison.

4.3 Experimental Results

We now present the results of the testing carried out on the PFO algorithm. Due to the large number of tests required to adequately investigate the PFO algorithms, we summarize each set of tests rather briefly and include plotted results for only a portion of the tests. Admittedly, due to space limitations, the results presented here are not comprehensive – a more detailed exegesis of the results of an exhaustive set of tests and parameter settings can be found in [3]. A brief overview of the best and worst results on each test function is presented in Table 2, and depicted graphically in the respective graphs subsequently.

4.3.1 Sphere Function

PFO configurations with a particle field population size smaller than the pool size performed best on the Sphere function, outperforming the baseline BBPS considerably. When the population size was equal to the pool size, performance was similar to the BBPS, and when the population size was larger than the pool size, the performance was considerably degraded. In all cases, there was a relatively small amount of variance in performance between the tested weighting schemes.

4.3.2 Rosenbrock Function

For the Rosenbrock function, PFO configurations using a population size of half the pool size performed poorly when compared to the baseline BBPS. Despite the Rosenbrock function being unimodal, these configurations

Table 2: A brief overview of the results for each test function.

Function	Dim	BBPS	Best PFO Configuration	Worst PFO Configuration
Sphere	10	$5.528287E^{-66}$	$1.329902E^{-89}$	$4.218981E^{-40}$
Rosenbrock	2	0.0170631	0.000081423	0.4980816
Schaffer F6	2	0.010073551	0.007617807	0.032898468
Ackley	10	0.0089554	$3.841728E^{-15}$	6.45359
Rastrigin	2	0.35890299	0.06705732	1.2443847
	10	4.466349	2.760911	6.579886
	20	17.55725	11.04656	21.33815
Griewank	2	0.01291898	0.005651862	0.259916617
	10	0.07132231	0.06830714	0.07989598
	20	0.02426064	0.02069583	0.02826242

quickly became stagnant and got stuck in a sub-optimal area within the first 50 iterations. Variations in the performance among the weighting schemes with this population size was also considerable, but with no particular outliers. Configurations with a population size equal to the pool size performed, on average, considerably better than the baseline BBPS. The variation in performance among the weighting schemes with this population size was significant, with the Average weighting scheme performing dramatically worse than the others. All the PFO configurations with a population size of twice the pool size performed dramatically better than the baseline BBPS.

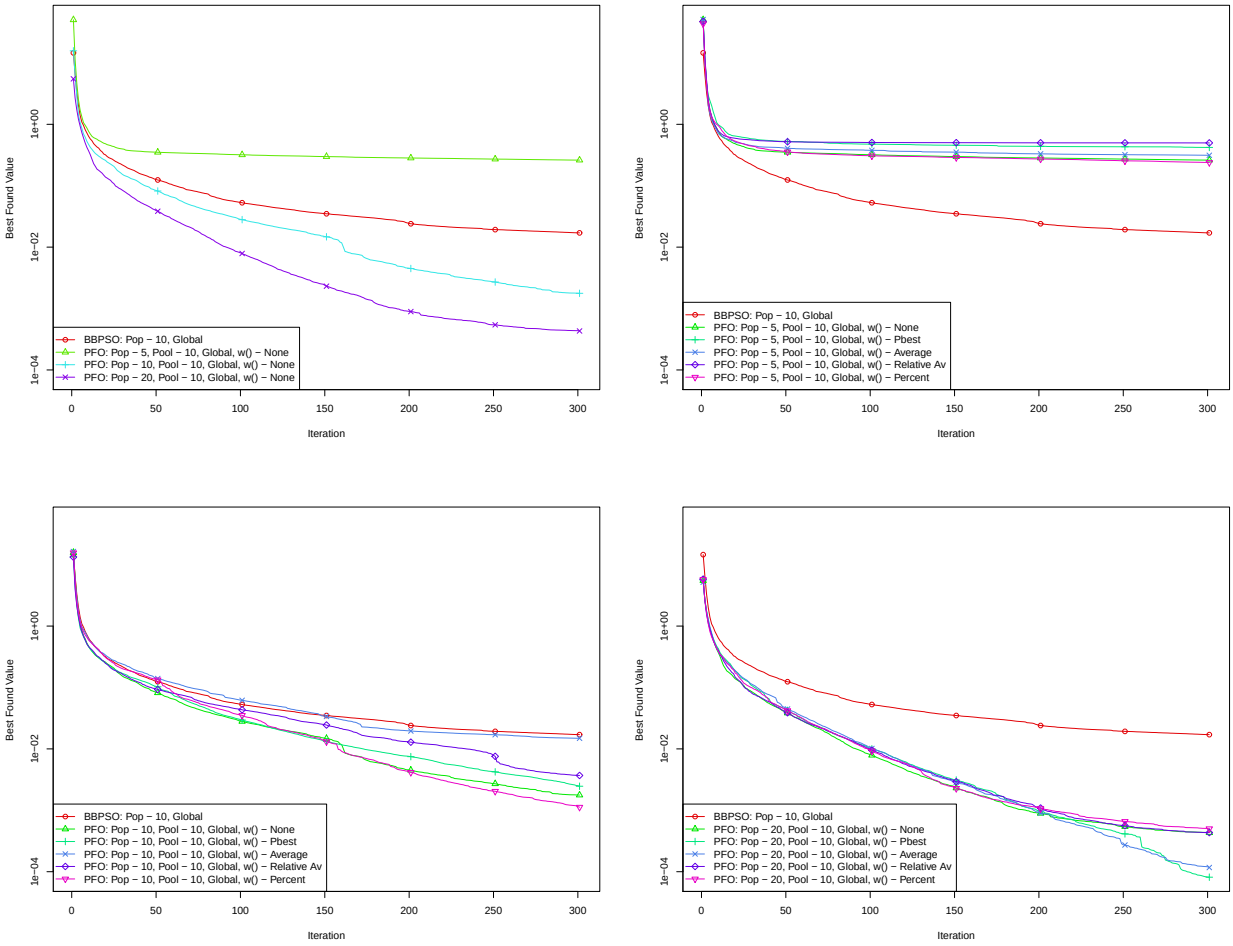
4.3.3 Schaffer F6 Function

In the case of the Schaffer F6 function, PFO configurations using a population size smaller than the pool size performed poorly compared to the baseline BBPS. The variation in performance among weighting schemes in this group was fairly significant, with two performing much worse than the others. These were the Personal Best and Relative Average weighting schemes. With a population size equal to the pool size, the PFO showed, on average, a small improvement over the baseline BBPS. With a population size of twice the pool size there was evidently no significant improvement over the baseline BBPS. The performance of these configurations was very similar to those which used a population size equal to the pool size which was a result unique to the Schaffer F6 function.

4.3.4 Ackley Function

The PFO configurations for the Ackley function using a population size of 25, performed the worst by a large margin. With a population size of 50, the PFO algorithm performed slightly better than the baseline BBPS

Fig. 3: The performance results of the various algorithms for the Rosenbrock function when the number of dimensions was 2.

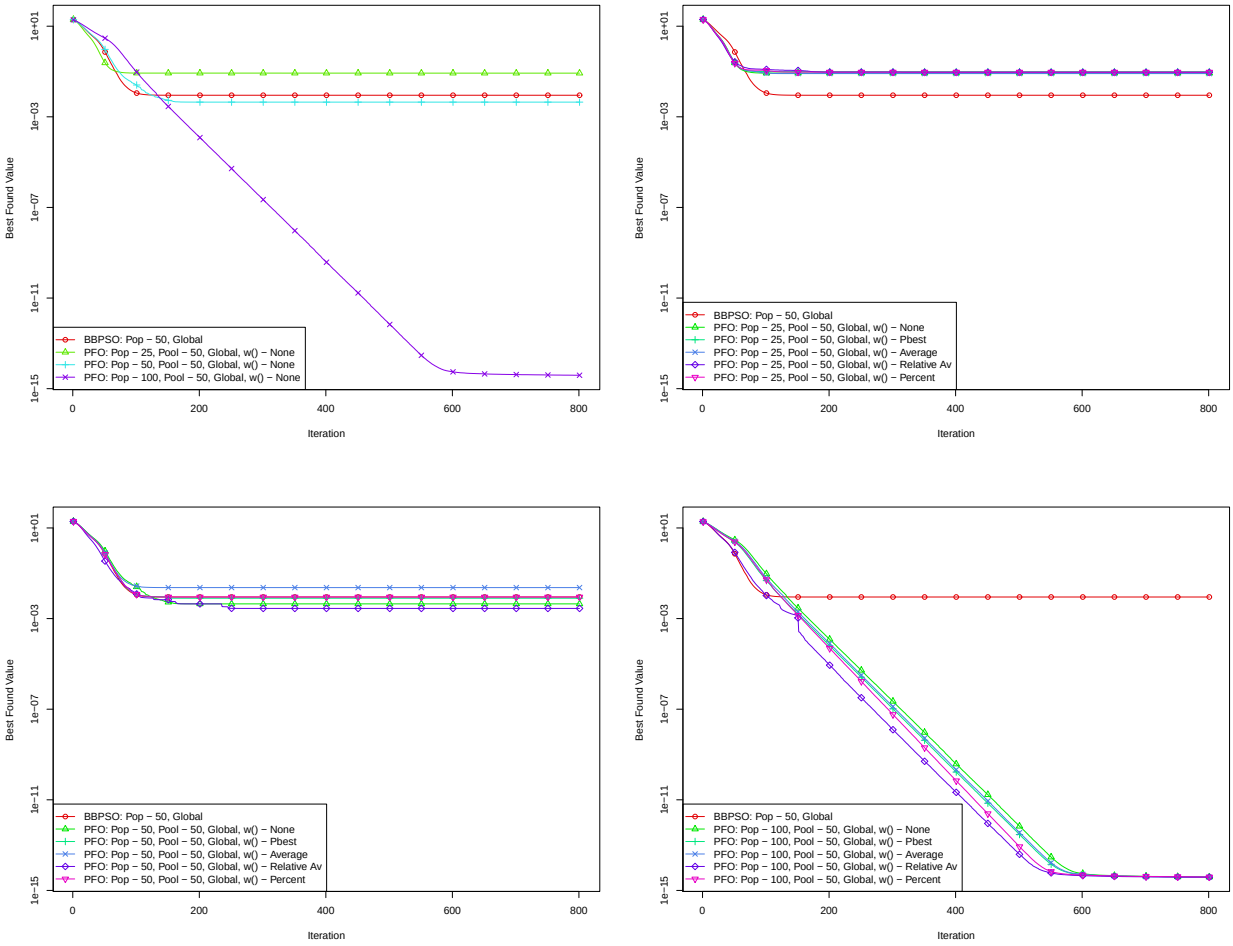


on the average, and configurations with a population size of 100 significantly out-performed all others. This dramatic result confirms the effect that a "bad initialization" can have on the algorithm's performance on the Ackley function. With a large population size, the chance for the entire population to be initialized in a bad area is dramatically reduced, and this, in turn reduces the rate of occurrence of extremely poor final values. A benefit of the PFO algorithm over other population-based methods, is that this population size can be increased without affecting the number of function evaluations per iteration.

4.3.5 Rastrigin Function

Between the 2, 10 and 20 dimensional versions of the Rastrigin function, the impact of the relative values of the particle field population size and pool size parameters were consistent. When using a population size less than the pool size, performance was poor compared to the BBPS. With a population size equal to the pool size performance was, on average, similar to the baseline BBPS and with a population size greater than the pool

Fig. 4: The performance results of the various algorithms for the Ackley function when the number of dimensions was 10.



size performance was significantly improved over the baseline BBPS. In all cases, the impact of the weighting scheme on performance was relatively insignificant.

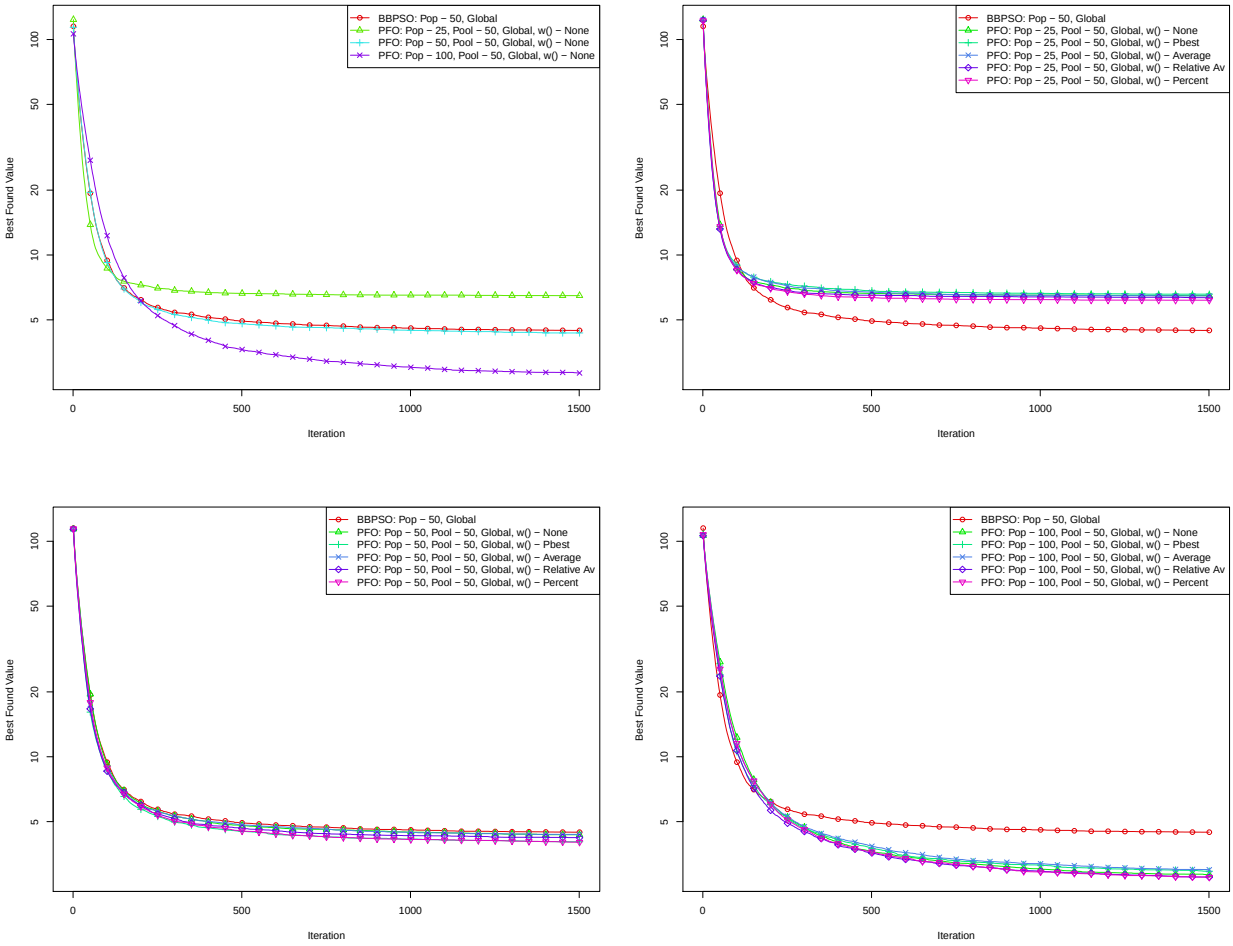
4.3.6 Griewank Function

Results on the Griewank function varied between the 2, 10 and 20 dimension versions of the function. On the 2-dimensional Griewank function, PFO configurations using population sizes smaller than the pool size performed much worse than both the baseline BBPS and PFO configurations with larger population sizes, which performed much better than the BBPS.

In terms of the final overall results, the different relative population sizes performed approximately the same as one another on the 10-dimensional Griewank function. This result is unique to the Griewank function, as results on all other test functions showed that the population size did have a dramatic impact.

Similar to the Griewank 10 function, on the 20-dimensional Griewank function, the configurations with different relative population sizes performed about the same as one another. However, in this case, the smaller

Fig. 5: The performance results of the various algorithms for the Rastrigin function when the number of dimensions was 10.

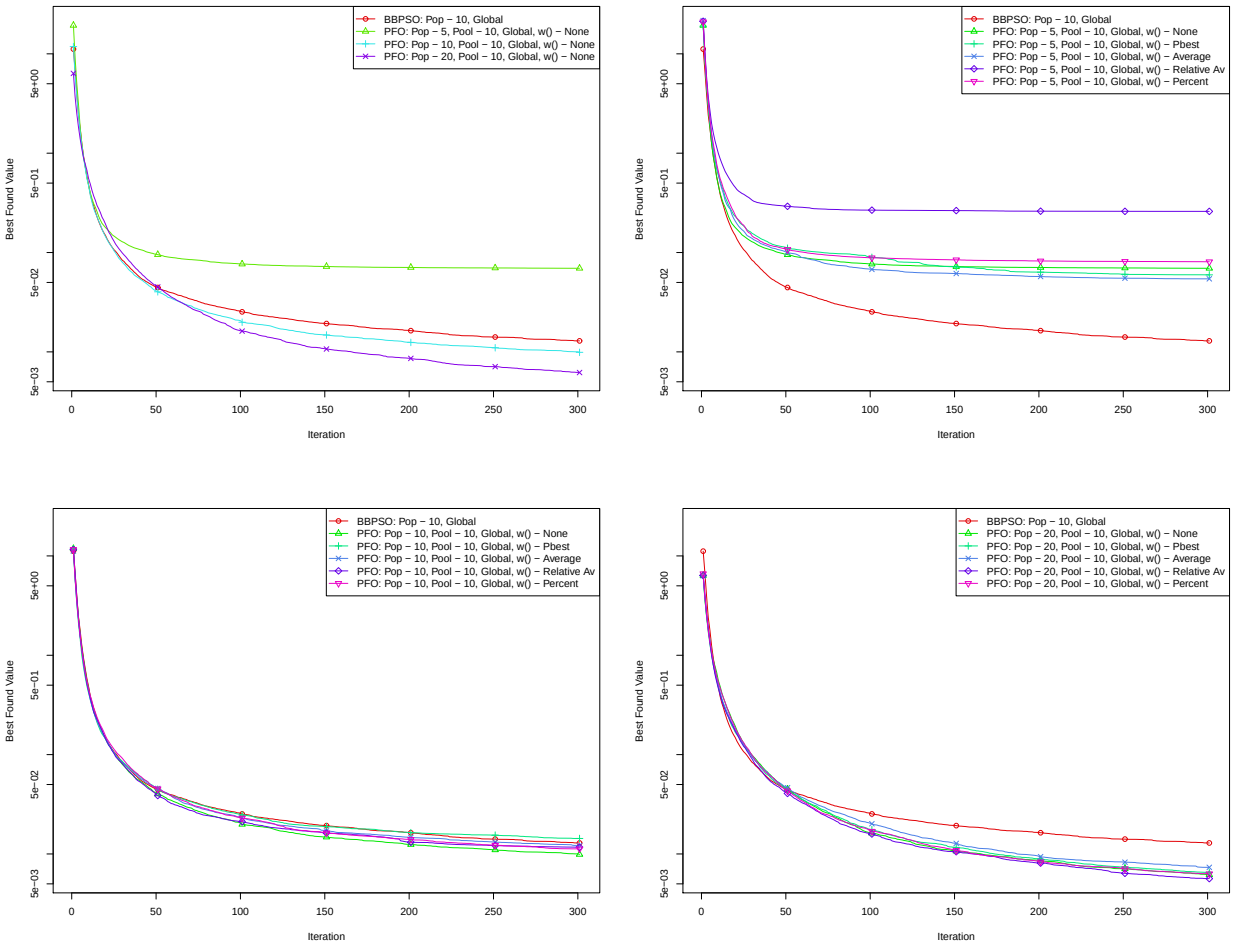


relative population size performed slightly *better* than the population size equal to the pool size, which, in turn, performed slightly better than the larger relative population size. This would suggest that as the dimensionality of the Griewank function increases, the larger population size becomes less beneficial, and more detrimental. Considering this along with the results of the Sphere function, this observation is in line with the idea that the Griewank function becomes "easier" as the number of dimensions is increased.

4.4 Discussions

From analyzing the results presented, it is possible to identify some patterns and gain insight into the behavior of the PFO algorithm, as well as the effects of the algorithm's parameters. From a cursory glance, the test results show a great amount of variation among the different configurations of the PFO algorithm on each test function, with many configurations performing significantly better than the baseline BBPS, and some performing significantly worse. This high variance among results shows that the different parameter values chosen had a significant impact on the PFO's behavior and performance.

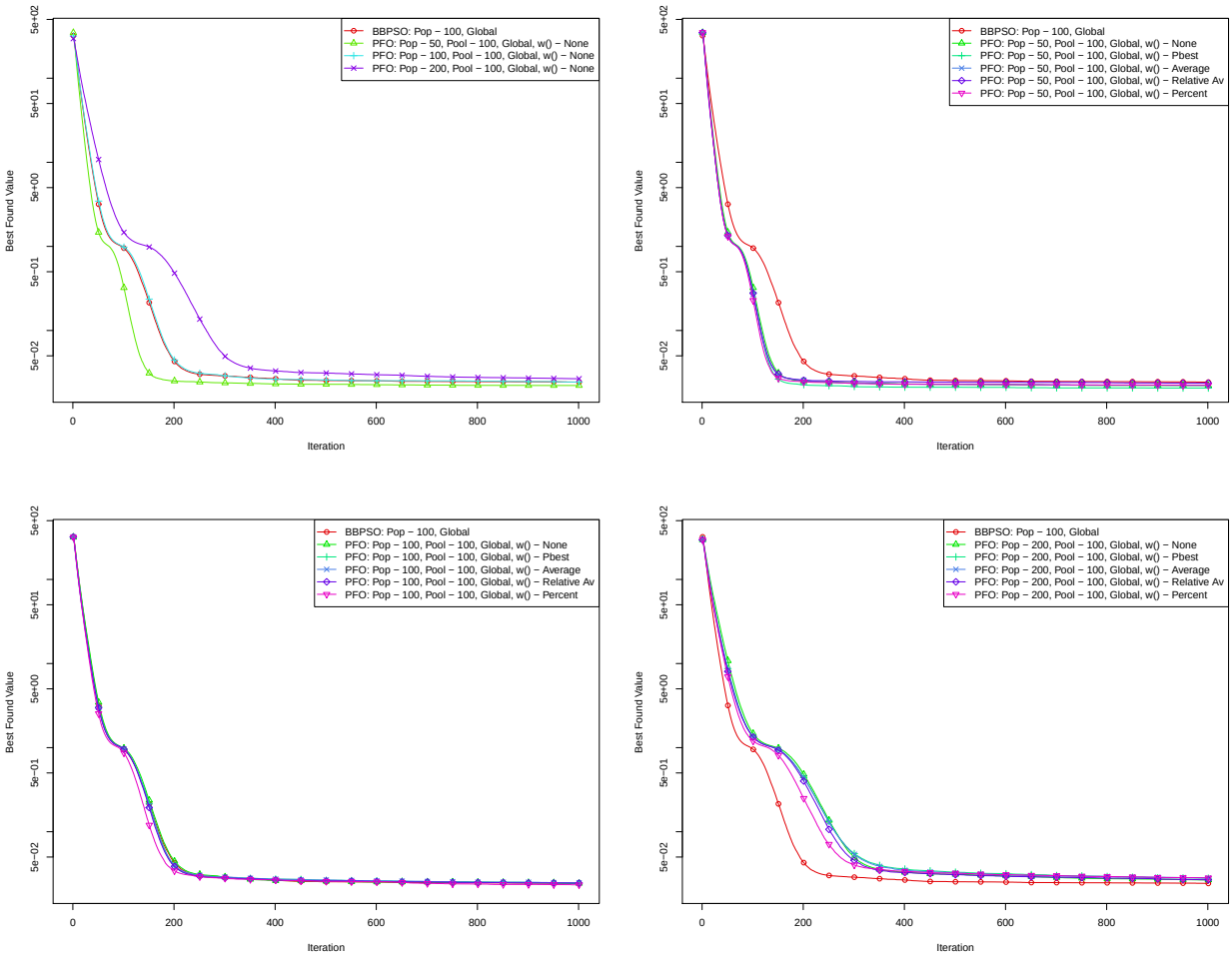
Fig. 6: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 2.



We can summarize the overall results as below:

1. Among these results, a clear pattern emerges, suggesting a significant correlation between the particle field population size and the behavior of the PFO algorithm. On all the test functions, with the exception of the Griewank 10 and 20 functions, the final results are divided by the particle field population size into distinct clusters. With the pool size parameter set as a single value for each function, the difference in the PFO configurations within these clusters is only the weighting scheme used. This suggests that the particle field population sizes chosen for each test "dominated" the choice of the weighting scheme, as there was no overlap between these groups.
2. In addition to this distinct grouping, a correlation between the relative order of these groups and the particle field population size can be observed. With the exception of the Sphere function, a larger particle field population size was correlated with a better best value found. On the Sphere function, the opposite is observed, where a smaller population size is correlated with a better best value found. In addition to this exception we observe the previously-mentioned exceptions of the Griewank 10 and 20 functions, on

Fig. 7: The performance results of the various algorithms for the Griewank function when the number of dimensions was 20.



which the particle field population size did not show such a distinct clustering on the final results. These exceptions can be explained when considering the plotted test results showing the average best value found at each iteration. Rather than make a correlation between the particle field population size and the final result, it is more accurate to correlate the particle field population size with the "exploratory" behavior of the algorithm. A smaller population size displayed a more "exploitative" behavior, performing well on "simple" functions, while a larger population size displayed a more "exploratory" behavior and performed well on more "complex" functions. This correlation is in line with results on *all* test functions, including the Sphere and Griewank 10 and 20 functions.

3. In the case of the Sphere function, a higher "exploratory" behavior would show poor performance compared to a more "exploitative" behavior, due to the simplicity of the function. This is in line with the plotted results, as configurations with smaller particle field population sizes were observed to converge much faster than configurations with larger population sizes.
4. The results on the Griewank 10 and 20 functions also suggest this correlation, though not as clearly. The plotted results show this correlation in the early iterations, although the final values are not distinctly

separated by the particle field population sizes. In this context, we mention that it has been shown that the Griewank function becomes "easier" as the number of dimensions are increased, and if we take this into account, these results, which are otherwise unexplainable on the Griewank 10 and 20 functions, can be explained. Results on the Griewank 2 function support the correlation between the particle field population size and the "exploratory" behavior, with larger population sizes yielding dramatically better final results. On the Griewank 10 function, the final results are not distinctly separated by the population size, with some configurations using a smaller population size outperforming configurations using larger population sizes and vice-versa. However, when grouping the results based on the population sizes, and comparing the average final result between these groups, it is still the case that the configurations with larger population sizes perform better (even though this superiority is sometimes marginal), on average, than those with smaller population sizes.

5. When comparing these observations with the results of tests on the Griewank 20 function, a possible pattern emerges which explains the observed exception. Similar to the Griewank 10 function, the results on the Griewank 20 function are not distinctly separated by particle field population sizes. However, the observed relation between the average final value for each group is, in fact, the opposite of the observed relation on the Griewank 10 function. In this case, the PFO configurations with a smaller population size performed, on average, slightly better than those with a larger population size. With the knowledge that the Griewank function becomes "easier" as the number of dimensions are increased [9], and recalling the results of tests on the Sphere function, a possible explanation for the results can be made. As the number of dimensions increases, the Griewank function becomes "easier", and so as the number of dimensions increases, "exploitative" behavior becomes more valuable than "exploratory" behavior. With a correlation between larger population sizes being much more "exploratory", and smaller population sizes being much more "exploitative", the difference in performance between the two on the Griewank function as the dimension increases would, intuitively, shrink, until the "exploitative" behavior of small populations begins to yield better results than the "exploratory" behavior of large populations. This is in line with the observed results on the Griewank function. Results on the Griewank 2 function show "exploratory" large populations performing much better than "exploitative" small populations. Results on the Griewank 10 function show the performance difference approaching zero. Finally, results on the Griewank 20 function show the "exploitative" small populations out-performing the "exploratory" large populations by a slight margin.
6. The weighting scheme was expected to have a large impact on the PFO algorithm's behavior. The test results, however, show that the weighting scheme chosen had a relatively small impact when compared with the particle field population size parameter. By grouping PFO configurations according to the particle field population sizes, the effects of the weighting schemes can be compared within these groups.
7. In addition to the tested weighting schemes, PFO configurations without a weighting scheme were tested [3] to provide a baseline point of comparison. Without a weighting scheme, each particle field individual is chosen with equal probability. The effectiveness of the weighting schemes varied greatly across the test functions and population size groups. On some functions there was a significant variation in performance between the weighting functions, and on some the variation was very small. Also, the relative performance of the weighting schemes within their groupings was inconsistent over the different test functions, with no significant patterns suggesting any clear correlation between the weighting schemes and specific behaviors.

In addition to this, there were many cases where some weighting schemes performed worse than the configuration with no weights (i.e., where the weighting did not depart from the default or from equal weighting), and in some of these cases, this configuration was observed to perform significantly better than all of the weighting schemes. These observations suggest that the effects of the weighting scheme on the behavior of the algorithm is quite un-intuitive, and so deciding on what weighting scheme is to be used for a particular problem may require an extensive trial and error approach.

5 Conclusions and Future Work

Particle swarm algorithms have been a popular topic of research and development since the inception of the original Particle Swarm Optimization (PSO) algorithm in 1995. Many different strategies have been explored to change, or improve, the PSO algorithm, with distinct motivations and results. The initial stages of our research into these various strategies led us to study the Bare Bones Particle Swarm (BBPS) algorithm, which simplified and abstracted the PSO algorithm. Upon discovering the potential for a further level of abstraction, we have been able to determine an even higher level of abstraction on which the particles were replaced by "fields", and to investigate the implications of the new perspective.

This work was written with two primary objectives in mind. The first objective was to present an abstracted perspective of the behavior of the BBPS algorithm. With this abstracted perspective came new opportunities for the development of the algorithm. The second objective was to present the newly created PFO algorithm, which was designed by exploring some of the new opportunities presented by this abstracted perspective of the BBPS algorithm.

In this paper, we have thoroughly achieved and explored both of these objectives. The abstracted perspective of the behavior of the BBPS algorithm presents opportunities for development that are not evident with the traditional particle swarm perspective. With this perspective as a foundation, we have explored a number of these avenues, applying a number of changes and additions to the BBPS algorithm, resulting in the distinct Particle Field Optimization (PFO) algorithm. The PFO algorithm has been shown to be effective in the optimization of a variety of different functions of high and low dimensionalities and differing complexities. It also presents a rich framework for further development and exploration of the abstracted perspective.

5.1 Application of Particle Swarm Developments to the PFO Algorithm

The PFO algorithm has an overall behavior which is distinct from traditional particle swarm algorithms, but the core particle swarm framework is still present within the population of particle field individuals. Because of this, it is possible to incorporate many facets, originally created for traditional particle swarm algorithms, into the PFO paradigm.

Introducing the concept of dynamically controlling the parameters of the PSO algorithm has been a popular topic of research and development, and has resulted in considerable performance increases in some cases. A basic example of this is a standard PSO algorithm with a dynamically controlled inertial weighting parameter. In this algorithm, the PSO system is initialized with a high inertia value, which is decreased over the course of the simulation. The motivation for this is to promote exploratory behavior in the early stages of the algorithm,

but to then shift toward exploitative behavior in the later stages. It is very likely that applying a similar control to the PFO algorithm's particle field population size parameter would result in a similar increase of performance. Empirical testing performed on the PFO algorithm suggests that there is a direct correlation between the particle field population size and the balance between the exploratory and exploitative behaviors, and so a dynamic "pruning" of the particle field population is likely to provide a similar shift in behaviour.

5.2 Future Research Avenues

With regard to future research avenues, first of all, it would be fascinating to see how a Thompson's sampling process could be incorporated into the PFO paradigm. If this can be formally achieved, it would also permit a more formal analysis to a research domain where most of the work is experimental in nature.

Another avenue for future research would pertain to weighting schemes. The weighting parameter of the PFO algorithm was originally intended as a way to introduce a dramatic change in the search behavior without interfering directly with the underlying particle swarm principles at work. However, the impact of the different weighting schemes tested was much less significant than we had hoped. The advantages gleaned were also very problem specific. Because of this, the improvement of the weighting scheme portion of the PFO algorithm presents a clear opportunity for further research and development. The observed effects of the weighting schemes were surprising, and so it is not yet clear *why* these different weighting schemes did not show the expected impact. Further investigation into the weighting schemes themselves is required to understand this.

Finally, with regard to experimentation, we propose the possibility of considering more general functions. In the section where we referred to the experiments that we had conducted, we had alluded to an overly generic "No Free Lunch" (NFL) argument. Actually, as suggested by one of the anonymous Referees, it turns out that it is quite difficult to come up with a problem set that strictly fulfils the NFL criteria. Thus we suggest that the PFO should also be tested with more general functions, where the the restrictions introduced by the NFL theorem are relaxed, or even removed.

References

1. P. J. Angeline. Using selection to improve particle swarm optimization. *IEEE International Conference on Computational Intelligence*, 1998.
2. M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, Vol. 6 pp. 58–73, 2002.
3. N. Bell. *Swarm Optimization Using Agents Modeled as Distributions*. MCS Thesis, Carelton University, 2014.
4. R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Congress on Evolutionary Computation*, Vol. 1 pp. 84–88, 2000.
5. N. Higashi and H. Iba. Particle swarm optimization with gaussian mutation. *IEEE Swarm Intelligence Symposium*, pp. 72–79, 2003.
6. J. Kennedy. Bare bones particle swarms. *IEEE Swarm Intelligence Symposium*, pp. 80–87, 2003.
7. J. Kennedy and R. Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks*, Vol. 4 pp. 1942–1948, 1995.
8. J. Kennedy and R. Mendes. Population structure and particle swarm performance. *Congress on Evolutionary Computation*, Vol. 2 pp. 1671–1676, 2002.
9. M. Locatelli. A note on the Griewank test function. *Journal of Global Optimization*, Vol. 25 pp. 169–174, 2003.

10. M. Lovbjerg, T. K. Rasmussen, and T. Krink. Hybrid particle swarm optimiser with breeding and subpopulations. *Genetic and Evolutionary Computation Conference*, pp. 469–476, 2001.
11. R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, Vol. 8 pp. 204–210, 2004.
12. C. K. Monson and K. D. Seppi. Exposing origin-seeking bias in pso. *Conference on Genetic and Evolutionary Computation*, GECCO '05, pp. 241–248, 2005.
13. E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. *Intelligent engineering systems through artificial neural networks*, Vol. 8 pp. 253–258, 1998.
14. E. Ozcan and C. K. Mohan. Particle swarm optimization: Surfing the waves. *Congress on Evolutionary Computation*, Vol. 3 pp. 1939–1944, 1999.
15. A. Ratnaweera, S. Halgamuge, and H. C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, Vol. 8 pp. 240–255, 2004.
16. M. Settles and T. Soule. Breeding swarms: a ga/pso hybrid. *Conference on Genetic and Evolutionary Computation*, GECCO '05, pp. 161–168, 2005.
17. Y. Shi and R. Eberhart. Empirical study of particle swarm optimization. *Congress on Evolutionary Computation*, Vol. 3 pp. 1945–1950, 1999.
18. J. Snoek, H. Larochelle and R. P. Adams. Practical Bayesian Optimization of Learning Algorithms. *Neural Information Processing Systems*, Vol. 25 pp. 2951–2959, 2012.
19. W. M. Spears, D. T. Green, and D. F. Spears. Biases in particle swarm optimization. *Int. J. Swarm. Intell. Res.*, Vol. 1 pp. 34–57, April 2010.
20. W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples *Biometrika*, Vol. 25 pp. 285–294.
21. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 1 pp. 67–82, 1997.
22. Z. H. Zhan, J. Zhang, and H. S. H. Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 39 pp. 1362–1381, 2009. *IEEE Swarm Intelligence Symposium*, pp. 80–87, 2003.