Paper ID: 607

ABSTRACT

Particle Swarm Optimization (PSO) has been a popular meta-heuristic for black-box optimization for almost two decades. 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 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.

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 viewed as "function optimization" problems.

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 non-specific definition, 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, multi-modal complexity, and elaborate constraints on the input values, render "standard" optimization methods impossible or infeasible, and so currently, new methods must be created and tailored to 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, algebraic manipulations (such as computing derivatives) are impossible. Thus, 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 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 new behaviour to the algorithm. 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

*Acknowledgements will come here.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May, 4–8, 2015, Istanbul, Turkey.*

swarm systems. Taking advantage of this new perspective, we propose a novel search algorithm known as Particle *Field* Optimization (PFO), which, to the best of our knowledge, is both unique and pioneering.

The rest of the paper is organized as follows. Section 2 contains a fairly comprehensive overview of the field including that of black-box optimization, meta-heuristics and the PSO. 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: 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 straightforward 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 an adjacent solution 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 such functions, 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 potentially-infinite size of 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 extremes. *Exploitation* is necessary to find the locally optimal solution in some area, and *exploration* is necessary to escape local optima and locate more areas for 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.

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 $\vec{X}_i$ within the solution space, a velocity $\vec{V}_i$, and a personal best found point $\vec{P}_i$. 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 $\vec{V}_i$, and position $\vec{X}_i$, according to the equation:

$$\vec{V}_i = \omega\vec{V}_i + \vec{U}[0,\phi_p] \otimes (\vec{P}_i - \vec{X}_i) + \vec{U}[0,\phi_g] \otimes (\vec{P}_g - \vec{X}_i)$$
$$\vec{X}_i = \vec{X}_i + \vec{V}_i.$$

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

2.3 Variations/Extensions of the PSO

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 to various levels of sophistication over the core algorithm to improve useability [15], [17], [19]. Other strategies include hybridizing the PSO algorithm with other population-based ideas such as genetic algorithms [1], [5], [10] [16].

Although the high level behavior of the PSO algorithm is very complex and difficult to analyze, analyses of simplified versions have been reported [2], [8], [13], [14]. This 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 [18].

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, observations of a *single* particle using the standard velocity strategy, with the system in a state of swarm stagnation (i.e. when the personal and global best points remain constant), produced a histogram which displayed a distinct bell-curved shape. This phenomenon led 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 the observed 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 variance is set as the absolute distance between the global and personal best points for each dimension. Given a particle i with position $\vec{X}_i$, whose personal best point is $\vec{P}_i$ and for which the global best point is $\vec{P}_g$, the particle's next position is determined according to:

$$\vec{P_m} = \frac{\vec{P}_i + \vec{P}_g}{2}, \qquad \vec{X}_i = \vec{\mathcal{N}}(\vec{P_m}, \vec{\sigma^2}),$$

where $\vec{\mathcal{N}}$ represents a function which creates a Gaussian random vector dimension-by-dimension, centred on $\vec{P_m}$. $\vec{\sigma}$ is a vector of standard deviations for each dimension, set to be proportional to the absolute values of the difference between the $\vec{P}_i$ and $\vec{P}_g$ points 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, the algorithm still involves a population of particles which move through the solution space, and which are influenced by the memories of their best found positions and the communication of the global best found position.

3. THE PARTICLE FIELD OPTIMIZATION (PFO) ALGORITHM

3.1 The Need for Further Abstraction

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.

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. The algorithm's behavior with this abstracted model is unchanged, but the metaphors and concepts of the original PSO may require re-examination.

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 best point is a memory that the particle maintains of the best point it has evaluated so far, and the global best position which can be thought of as being 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.

3.2 How the PFO Abstraction is Affected

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 exactly one new solution. However, this population distribution can be used to guide the search in a different way.

Candidate solutions are instead generated by sampling this population-level distribution. Although the population-level distribution is complex, the sampling process is simple. The population-level distribution is a finite mixture distribution with known component distributions. Therefore, sampling this complex population-level distribution 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 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.

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.

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 criteria 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 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 criteria is met, typically the maximum number of iterations. Each iteration consists of two phases. In the first phase, candidate solutions are generated. These candidate solutions are generated by sampling the mixture distribution defined by the population of particle field individuals. For each point in the point pool we do the following: A particle field individual is selected at random from the population, according to some weighting scheme. Then, the point is generated by sampling the random distribution defined by the selected individual. This random distribution is constructed using the individual's personal best, and the global (or neighborhood) best points in the same way as the BBPS method does. Given a particle field with personal best point $\vec{P}_i$ and for which the global (or neighborhood) best point is $\vec{P}_g$, the position of the candidate solution point, $\vec{c}$, is determined according to:

$$\vec{P_m} = \frac{\vec{P}_i + \vec{P}_g}{2}, \qquad \vec{\sigma^2} = |\vec{P}_i - \vec{P}_g|, \qquad \vec{c} = \vec{\mathcal{N}}(\vec{P_m}, \vec{\sigma^2}).$$

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 this 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 criteria have been met, the global best found point is returned as output of the algorithm. The formal algorithm follows in Algorithm 1.

Algorithm 1 Particle Field Optimization Algorithm

Input:
Function $f()$ to be optimized
Initialization range $lbound$, $ubound$
Particle field population size n_{pop}
Candidate solution point pool size n_{pool}
Weighting function $w()$
Output:
Point $\vec{P_g}$ 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 $i \in P$ **do**
  $\vec{P_i} \Leftarrow \vec{U}[lbound, ubound]$
  if $f(\vec{P_i}) < f(\vec{P_g})$ **then**
    $\vec{P_g} \Leftarrow \vec{P_i}$
  end if
end for

while termination criteria not met **do**
  for each candidate solution point $\vec{c} \in C$ **do**
    select particle field $i \in P$ with probability $\frac{w(i)}{\sum_{p \in P} w(p)}$
    $\vec{c} \Leftarrow \vec{\mathcal{N}}\left(\frac{\vec{P_i}+\vec{P_g}}{2}, |\vec{P_i} - \vec{P_g}|\right)$
    $S_i \Leftarrow S_i \cup \vec{c}$
  end for
  for each particle field $i \in P$ **do**
    choose point $\vec{c_{min}}$ from S_i which minimizes $f(\vec{c_{min}})$
    if $f(\vec{c_{min}}) < f(\vec{P_i})$ **then**
      $\vec{P_i} \Leftarrow \vec{c_{min}}$
      if $f(\vec{P_i}) < f(\vec{P_g})$ **then**
        $\vec{P_g} \Leftarrow \vec{P_i}$
      end if
    end if
    $S_i \Leftarrow \emptyset$
  end for
end while
return $\vec{P_g}$

4. THE PFO: EXPERIMENTAL RESULTS

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

Table 1: The set of the test functions used in the test 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	$\sum_{i=1}^{d} x_i^2$
Rastrigin	10	$10d + \sum_{i=1}^{d} \left[x_i^2 - 10cos(2\pi x_i)\right]$
	20	
Griewank	2	$1 + \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right)$
	20	

4.2 Overview of Testing Strategy

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. Consequently, a number of different values were chosen for each parameter, and tests were performed for each combination. 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 functions, dimensions, tests and parame-

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

Function	Dim	BBPS	Best PFO	Worst PFO
Sphere	2	$5.5282E^{-66}$	$1.3299E^{-89}$	$4.2189E^{-40}$
Rosenbrock	2	0.0170	$8.1423E^{-5}$	0.4980
Schaffer F6	2	0.0100	$7.6178E^{-3}$	0.0328
Rastrigin	10	4.4663	2.7609	6.5798
	20	17.5572	11.0465	21.3381
Griewank	2	0.0129	$5.6518E^{-3}$	0.2599
	20	0.0242	0.0206	0.0282

ter settings can be found in [3][1]. 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.

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 (i.e., the 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 schemes.

4.3.2 Rosenbrock Function

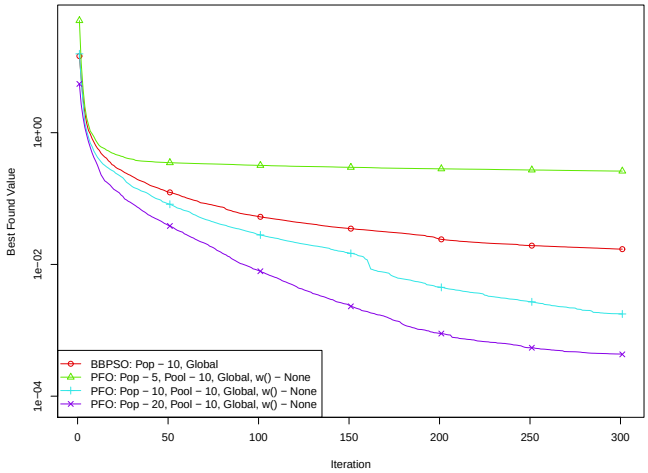
For the Rosenbrock function, PFO configurations using a population size of half the pool size performed poorly when compared to the baseline BBPS. Despite the Rosenbrock function being unimodal, these configurations quickly became stagnant and got stuck in a sub-optimal area within the first 50 iterations. 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 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.

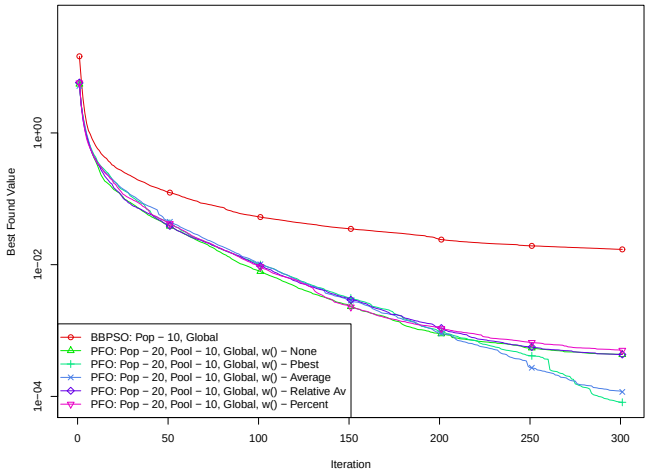
4.3.3 Schaffer F6 Function

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 and to twice the pool size, the PFO showed, on average, a small improvements over the baseline – which was a result unique to the Schaffer F6 function.

[1]The name of the first author, the title of his thesis [3] and his university have not been declared to preserve the double-blind nature of this submission.

Figure 1: A selection of performance results for the Rosenbrock-2 function





4.3.4 Rastrigin Function

Between the 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 the performance was, on average, similar to the baseline BBPS and with a population size greater than the pool size, performance was significantly improved over the baseline BBPS. The impact of the weighting scheme on performance was relatively insignificant.

4.3.5 Griewank Function

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

On the 20-dimensional Griewank function, the configurations with different relative population sizes performed about the same as one another. However, the smaller rela-

tive 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.

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 20 function, 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 20 function, on 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. This correlation is in line with results on *all* test functions, including the Sphere and Griewank 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

Figure 2: A selection of performance results for the Rastrigin-10 function.

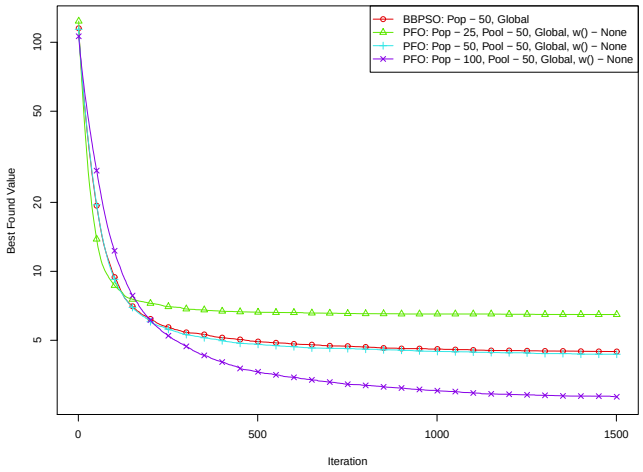


Figure 3: A selection of performance results for the Griewank-2 function.

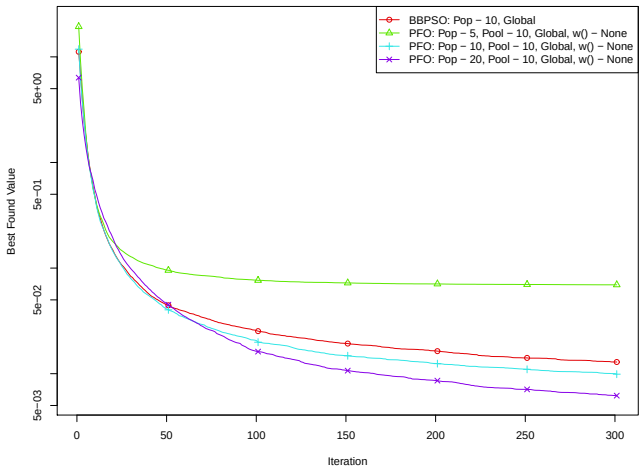
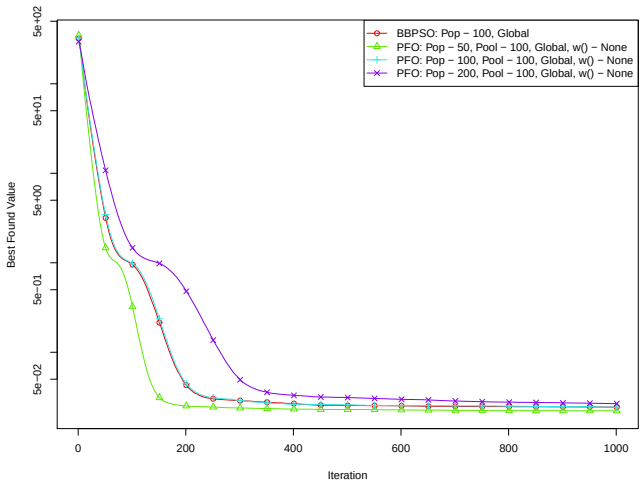


Figure 4: A selection of performance results for the Griewank-20 function.



sizes were observed to converge much faster than configurations with larger population sizes.

4. The results on the Griewank function 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 connection, 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 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.

5. When comparing these observations with the results of tests on the Griewank 20 function, a possible pattern emerges which explains the observed exception. 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 2 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, explaining the PFO's behavior that its "exploitative" nature of small populations begins to yield better results than its "exploratory" nature for large populations.

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 with no weighting scheme were tested [3] to provide a baseline point of comparison. 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. This is addressed in more detail in Section 5.2 which deals with the potential for more research in this area.

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. As far as we know, such an abstraction is both novel and pioneering in the area of swarm-like algorithms, and in the field of AI, in general.

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 many 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 many new opportunities for development which are not evident with the traditional particle swarm perspective. With this perspective as a foundation, we have explored a number of these new opportunities, applying a number of changes and additions to the BBPS algorithm, resulting in the distinct Particle Field Optimization (PFO) algorithm. The PFO algorithm itself is a novel contribution, proving to be effective in the optimization of a variety of different functions of high and low dimensionalities and differing complexities. The PFO algorithm also presents a rich framework for further development and research of the new and abstracted perspective.

5.1 Application of PSO Concepts to the PFO

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. It is thus possible to incorporate many PSO-based facets, into the PFO paradigm. The PFO can also be applied in all the problem domains where PSO schemes are applicable.

5.2 Improvement of Weighting Schemes

The weighting scheme 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, then, is required to understand this.

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] A. Author. *Thesis of the First Author*, Undeclared 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] 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.

[19] 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.