Mathematical Function Optimization Using A Novel Algorithm Based On Newtonian Field Theory

Todd Perry
School of Computing
University of Portsmouth
Portsmouth PO1 3HE, UK
Email: todd.perry@myport.ac.uk

Mohamed Bader-El-Den
School of Computing
University of Portsmouth
Portsmouth PO1 3HE, UK
Email: mohamed.bader@port.ac.uk

***Abstract*—Over the past several decades, function optimization has been a growing topic in the field of computational intelligence. This is partly down to the myriad of real world problems that function optimization can be applied to, but also the fact there are a number of issues facing optimization algorithms that are still yet to be solved. Such problems include getting stuck in local optima, and balancing exploration and exploitation. This paper introduces a novel approach to solving the function optimization problem that utilizes the equations of Newtonian field theory to find good solutions. Like a number of existing optimization algorithms, this approach models a number of particles and their positions in solution space. However, the algorithm proposed in this paper introduces a number of interesting behaviours that can help solve some of the aforementioned issues. The algorithm is explained using both formal mathematics and pseudocode, and the emergent behaviours of the algorithm are discussed. In addition to this, the approach is compared to other optimization algorithms using a set of different functions. The results of these experiments, as well as potential improvements to the proposed algorithm are discussed.**

I. Introduction

In real-world applications, resources are always limited. The optimal use of the available resources is a crucial factor for success, therefore, optimisation plays an important role in many applications. In the abstract form, the aim of optimisation is maximizing or minimizing a given real function, for example, minimise energy consumption, maximize the profit and reliability.

Many problems can be solved easily using polynomial time algorithms, however for some problems this is not the case [2]. For these harder problems, it is infeasible to find a solution in a reasonable amount of time. One example of such a problem is finding the highest or lowest point of a mathematical function. A simple solution would be to try every possible combination of parameters within the given upper and lower bounds, however as the granularity of the search and the number of dimensions of the space being searched increases, the number of combinations quickly becomes unworkable. This problem may sound abstract, but it can in fact be applied to a multitude of real world problems. Any problem involving multiple parameters, and one or more qualities that require optimizing falls into this category, for example the design of satellite components [9], the management of nuclear fuel in a power plant [13] and the optimization of machine learning algorithms. [17].

The problem itself involves 'hunting' for either the peaks (maxima) or the troughs (minima) of a function's solution space. Each possible point in solution space represents one possible combination of the functions parameters. For example the point in the solution space of the function $f(x_1, x_2) = x_1 + x_2$ given the inputs 1 and 2 would have the value, or 'fitness' of 3. As the number of parameters increases, so does the number of dimensions in the solution space. High dimensional solution spaces are harder to search [5]. One common problem with optimization algorithms is getting stuck in local optima, this happens when something analogous to a peak surrounded by two troughs exists in the solution space. When the algorithm reaches the peak, it may be unable to escape the local maxima due to the surrounding landscape.

Many optimization algorithms are inspired by natural phenomenon, Genetic Algorithms [1] are inspired by Darwinian Evolution, each 'generation' of solutions has offspring, where high performing individuals are more likely to pass on genetic information, eventually resulting in individuals finding optima. Particle Swarm Optimization [7] mimics the swarming behaviour of a flock of birds, honing in on good solutions. If one individual in the swarm comes across a new best solution, the rest of the individuals will swarm towards it, searching the local area around said solution for other good solutions. Simulated Annealing [12] models the dissipation of energy in a system of particles. Upon initialization, SA's particles have a high energy, and are very likely to jump in and out of peaks and troughs, favouring exploration over exploitation. As the system cools, energy is lost, and individuals converge, and exploit optima.

Classical field theory [14] was proposed in the 17th century, and describes systems of bodies that exert forces upon each other. The force exerted on one particle by another is inversely proportional to square of the distance between the two particles. Examples of systems that use the classical field theory equations are Gravitational and electromagnetic fields. Before Einstein's theory of relativity [8], Newtonian gravitational field theory was the accepted model of gravitational interaction, used to predict the future positions of celestial bodies. In the case of celestial bodies, more massive bodies, for instance a star, have a much stronger gravitational pull than smaller bodies, for instance, a planet. Inspired by this, the algorithm proposed in this paper attempts to find global optima by making good solutions exhibit a higher mass, and in turn exert stronger attractive forces to other particles, resulting in

particles swarming towards good solutions.

This paper proposes a novel optimization algorithm that is inspired by Newtonian field theory. The algorithm seeks to effectively find optima of mathematical functions, and at the same time is built upon the mathematics of classical field theory.

The rest of the paper is structured as follows: Section II gives a brief overview of a number of existing algorithms that solve the function optimization algorithm, as well as a more in depth overview of Newtonian field theory. Section III outlines the proposed algorithm, and provides a formal description of each step, including pseudocode. In addition to this, some emergent behaviours seen in the algorithm's particles are discussed and analysed. In section IV the algorithm is evaluated on a number of test functions, and compared to a number of other algorithms. The results are discussed and analysed, and the advantages and disadvantages of the proposed algorithm are investigated. The paper is concluded in Section V.

II. Related Work

In mathematical optimization, given a function with p parameters, the aim is to find the global optima (either minima or maxima) of the function. Various techniques have been proposed to locate optima. However, the no free lunch theorem [19] showed that, for all search algorithms (including random search), the average computational cost of finding a solution, over all the available problems in the class, is the same. Despite this, some algorithms perform better than others depending on the characteristics of the input function. Some of the simplest optimization algorithms are gradient based methods [4]. Such methods analyse the solution space by differentiating the input function. The derivative allows the gradient to be calculated, and the particle can move to a more optimal position in solution space. Gradient based methods have a major disadvantage in that if the solution space has many peaks and troughs they are prone to get stuck in local optima. In addition to this, due to the fact the input function must be differentiated, these methods do not work on discrete functions. A number of algorithms that overcome these problems have been developed, and a number of them are discussed in this section.

Optimization algorithms based on the positions of particles (solutions) in p dimensional euclidean space are no new paradigm. In 1995 Particle Swarm Optimization (PSO) was proposed by Kennedy et al. [7]. PSO is an optimization algorithm inspired by the movements of natural swarms, such as flocks of birds, or shoals of fish. The implementation is simple, there are a number of particles defined upon initialization, each particle has random coordinates in p dimensional space and a velocity vector of length p. Each iteration, particles move around the solution space. As well as this, each particle accelerates towards the current global best solution, and it's own best historical solution. Over time, the particles should swarm areas with many good optima, and hopefully locate global optima. PSO does have some disadvantages, for example simple PSO algorithms can struggle to escape local optima [20].

Proposed in 1983, another commonly used optimization algorithm is Simulated Annealing (SA) [12]. Like PSO, Simulated Annealing models the movement of a number of particles in an p dimensional solution space. Unlike PSO, SA has an energy parameter, which decreases over time. Each iteration, each SA particle considers it's local region of solution space, and if the energy is high, will have a high chance of moving to a position with a lower fitness, ie. a 'worse' position in solution space. The aim of this is to overcome the problem of getting stuck in local optima. As the system 'cools', these random walks become less and less probable, eventually resulting in the particles saturating in what is hopefully a global optima. Another commonly used algorithm, TabuSearch [10] records a list of previously known solutions, preventing 'loops' from occurring, and when faced with a known local optima, will climb out of it. These algorithms have overcome some of the constraints of simple hill climbing algorithms, but in the process have become more complex.

In addition to the aforementioned particle based algorithms, Genetic Algorithms (GA) are commonly used optimizers [1]. Genetic algorithms model a number of individuals, each of which encode solutions in a 'genome'. In the case of mathematical function optimization, the genome will be representative of an individuals position in solution space. Genomes can be modified via Genetic Operators. The mutation operator randomly changes part of the genome, which as a result will cause the position of the individual to randomly move in solution space. This is utilized by GAs to explore a solution space. Crossover takes two individuals, and 'mixes' their genomes, resulting in offspring with similar traits. This operator is useful when exploiting optima. Each iteration of the algorithm, a number of individuals are selected for mutation and crossover. The higher the fitness of a given individual, the more chance it will be selected. A key advantage of GAs is that they can be used to optimize both continuous and discrete search spaces, which is not the case for PSO, SA and TabuSearch.

Newtonian Field Theory [18], or Classical Field Theory, was proposed by Issac Newton in the 17th century. Newtonian Field Theory is a set of equations that can be used to model real world systems, such as the movements of celestial bodies. In such systems, all bodies exhibit a force on all other bodies, the amount of force exerted on a given body affects it's acceleration, which in turn will affects it's velocity and ultimately it's position. The potency of these forces is proportional to the inverse square of the distance between particles ($1/d^2$). This is easily visible in real world examples, for instance the Andromeda Galaxy, whilst incredibly massive, has little effect on the movement of the Moon around the Earth, due to the inverse square rule. In Gravitational Field Theory, the attractiveness of a body is directly proportional to it's mass. The algorithm proposed in this paper utilizes this by making the mass of a given body proportional to the fitness of the solution it represents.

III. Proposed Algorithm

The proposed method in this paper falls under the category of population (multi-solution) based methods as it makes use of a number of particles/solutions in finding the optima. As mentioned in the previous section, the proposed algorithm utilizes the classical gravitational field theory equations, and uses the fitness of a given particle to work out it's mass. That

mass is used to calculate how a given particle affects all other particles, the desired effect is to attract particles towards good solutions by making said solutions more massive. Each *particle* is a point in search space and is aware of it's p coordinates in the solution space, the fitness of a given particle is evaluated by passing the coordinates into the function being optimized.

The algorithm models the behavior of N particles in p dimensional space, the behaviour of a particle is affected by the gravitational pull of the other particles. This is done by manipulating the position and velocity of each particle based on the forces applied by other particles. For each particle i, the positions are stored in p dimensional matrix s, and the velocities are stored in p dimensional matrix v. For example s_{ix} corresponds to the position of particle i in the xth dimension, and v_{jy} corresponds to the velocity of particle j in the yth dimension.

In addition to this, each particle has a mass, which is stored in a mass vector, m of length N (mass m_i corresponds to the mass of particle i). The mass of a particle is calculated based on the fitness of it's position in the p dimensional solution space. Masses can be both positive and negative, particles with negative masses repel other particles, whilst particles with positive masses attract other particles. Whether or not a mass is evaluated as positive or negative is based on a threshold, if a particle has a fitness below the threshold, it will have a negative mass, and repel other particles. Otherwise, the particle will have a positive fitness, and attract other particles.

During each iteration, new velocities, positions and masses are calculated by taking the gravitational pull of the N particles into account. Each iteration, the following steps are run:

Firstly, the mass of each particle is evaluated using the equation below:

$$m_i = \begin{cases} (f(s_i) - mT)^k & \text{if } f(s_i) \geq mT \\ (f(s_i) - mT)k & \text{if } f(s_i) < mT \end{cases} \tag{1}$$

Where $f(s_i)$ is the result of the input function given input vector S_i, mT is the mass threshold parameter, and k is a parameter used to define how strongly particles should attract or repel one another. To ensure that attractive forces are more powerful than repulsive forces, particles with a positive mass are raised to the power of k, as opposed to being multiplied by k. These parameters are described in greater detail later in this section, but in short, they can be used to tune the behaviour of the algorithm. Regardless of the parameters being used, the important thing is that the mass of a particle is proportional to the fitness, which promotes swarming towards good solutions.

Once the masses are calculated, the new position and velocity are calculated for each *particle* in the population. This is achieved by calculating the gravitational forces exerted on each *particle*. The force particle j exerts on particle i, in the x dimension is defined as:

$$F_{ijx} = \frac{Gm_i m_j(s_{ix} - s_{jx})}{||s_i - s_j||^p} \tag{2}$$

Where G is a gravitational constant, and m_i, m_j are the masses of particles i and j. By summing all $j \in \{1...N\}$, the total gravitational force exerted on particle i in the x dimension can be calculated using the following equation:

$$F_{ix} = \sum_{j=1}^{N} \frac{Gm_i m_j(s_{ix} - s_{jx})}{||s_i - s_j||^p} \tag{3}$$

Using F_{ix} and the equation $F = ma$, it is simple to calculate the x acceleration of particle i:

$$a_{ix} = \frac{F_{ix}}{m_i} \tag{4}$$

Using a_{ix} it is possible to calculate the updated x velocity of i, we know that:

$$\frac{dv}{dT} = a \tag{5}$$

Where dT is the derivative of time, and dv is the derivative of velocity, this can be re-arranged to give:

$$dv = adT \tag{6}$$

This means that the change in rate of velocity can be found by multiplying a by parameter dT. Plugging the x velocity of particle i gives:

$$v_{ix}^{(n+1)} = v_{ix}^{(n)} - a_{ix}dT \tag{7}$$

Where n is the iteration count. Note: as gravity is an attractive force, the acceleration is negative. Finally, the x position of body i can be updated, we know that:

$$\frac{ds}{dT} = v \tag{8}$$

and therefore:

$$ds = vdT \tag{9}$$

ds gives the rate of change in position over time, so the updated position of particle i is:

$$s_{ix}^{(n+1)} = s_{ix}^{(n)} + v_{ix}^{(n+1)}dT \tag{10}$$

Once the positions are updated for all dimensions, the masses are recalculated and the cycle repeats until the max iteration count is reached.

This process describes one iteration of the proposed algorithm. The velocity, position and mass of all particles update each iteration, with the exception of the particle with the highest fitness, which moves at a lower velocity, decided by an algorithmic parameter: cvr, or centroid velocity reduction. The velocities of the most optimal particle are divided by cvr before the positions are updated, therefore is cvr is set to 1, the best particle will move at normal velocity.

The resulting behavior of the algorithm is that particles swarm around known good solutions, a behavior similar to that of PSO. However, in addition to this, a behavior resembling the gravitational slingshot effect [6] is apparent. Particles cluster together around optima, resulting a large number of massive particles in close proximity to each other, and huge forces being exerted on the particles. The result is particles occasionally being slung far away from the particle cluster, which can result in new solutions being discovered.

Another emergent behavior appears in solution spaces with an incline in a given direction. For instance the simple two dimensional function: $f(x_1, x_2) = x_1 + x_2$ has a solution space in which the fitness increases as both x_1 and x_2 tend to infinity. If this function was given to the proposed algorithm, the particles would 'tumble' towards higher values of x_1 and x_2, and build up momentum in the direction of the incline. Effectively, the proposed algorithm has the potential to learn if a given solution space has an incline, and can take advantage of it to find better solutions.

When the algorithm was developed, a piece of software to visualize it was also developed. This software allows 'traces' to be captured, and these traces can be used to analyse the emergent behaviours. In this visualization software, both the two dimensional function, and the paths of the particles are visualized. In the background of the traces, the red pixels represent areas of the solution space with a high fitness, whilst blue pixels represent areas with a negative fitness. In the foreground, the dark green trails represent the previous positions of the particles, and the light green circles represent the particles' current positions.

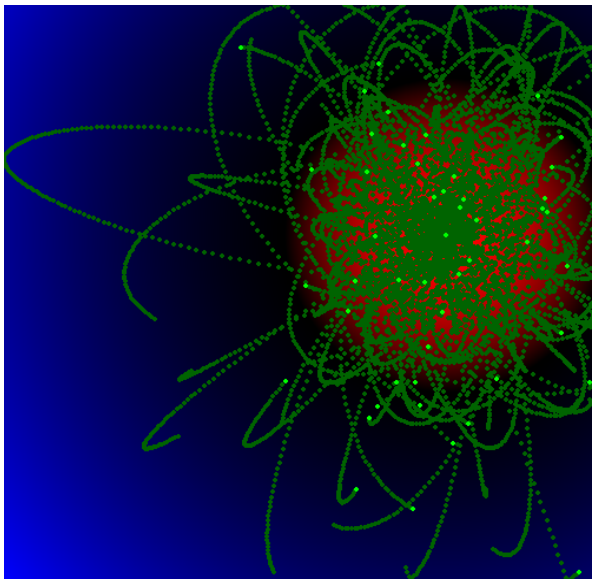

Fig. 1. This figure shows an example of the swarming behavior. Particles can be seen swarming around the optima of $f(x_1, x_2) = (x_1 - 5)^2 + (x_2 + 2)^2 + 20$.

In figure 1 the swarming behaviours can be observed. Particles appear to orbit around a common point of attraction, which in this case is the global maxima of the inverted sphere function. This behaviour allows the algorithm to exploit optima, this can be seen by observing the coverage of the particle's trails. The trails have a wide coverage of the area around the optima, and the coverage increases as the distance to the optima decreases.

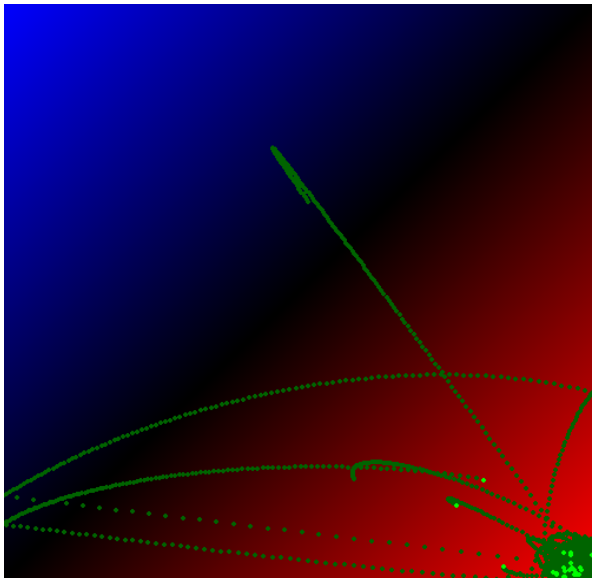

Fig. 2. Here, a trace of the algorithm running on the function $f(x_1, x_2) = x_1 + x_2$ shows the emergent 'slingshotting' behavior.

The 'slingshotting' behavior is a direct result of the swarming behavior. When many particles of high mass come within close proximity of each other, some are fired off at high velocity. This can be seen in figure 2. The space between the points in the trail is proportional to the velocity of the particle. In the figure a dense cluster of particles can be seen in the bottom right, and a number of other particles can be seen being launched from the cluster. These particles are able to jump out of local optima, and potentially find new optima, much like the random walks of Simulated Annealing [12]. It is worth noting that in this implementation, particles that hit the edge of the solution space 'bounce' off with a reduced velocity in the relevant dimension. This velocity reduction is controlled by a coefficient of restitution parameter.

Figure 3 shows an example of the tumbling behavior. From the trail it is apparent that the particles grouped together in a cluster, which then preceded to move towards the general direction of the incline. In this example, the solution space has numerous local maxima, in which many less intelligent optimization algorithms would become stuck. In addition to the tumbling and swarming behaviours visible in figure 3, some of the particles are also exhibiting the slingshotting behaviour, some making it most of the way across the bound solution space.

The proposed algorithm has four parameters that can be used to fine tune performance: G, which controls the strength of gravity, mT, which defines at what point the force becomes attractive or repulsive, k, which controls how much the fitness of a particle affects the mass and finally cvr which controls how much slower the current best particle moves. By tweaking these parameters, the overall behavior of the algorithm changes, for instance a high G or k value causes a lot of singshotting, leading to an exploratory search, whilst a low value of G or k causes slow moving swarms to manifest, allowing for a more exploitative search. It is important to keep

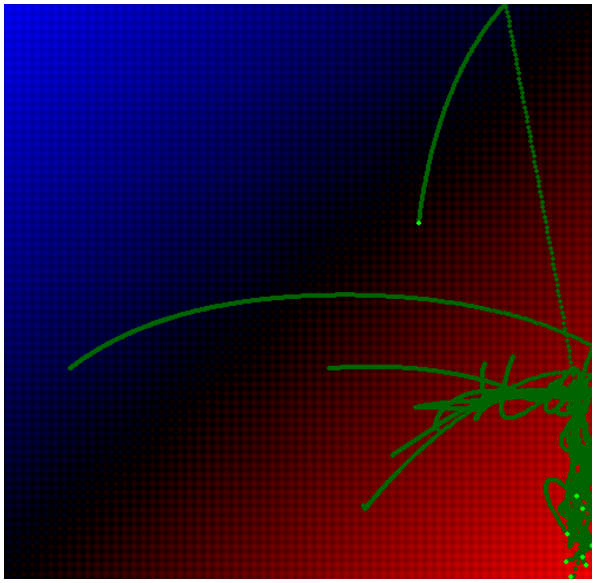

Fig. 3. This figure shows the third emergent behaviour introduced in this section, 'tumbling'. The particles form a moving cluster, that tumbles towards the global optima of in this case the graph $f(x_1, x_2) = \sin(2(x_1 + 0.2)) + \sin(2(x_2 + 0.2)) + 0.1x_1 + 0.1x_2$.

cvr balanced, too low and the particles will glance over optima without exploiting them, too high and proper exploitation becomes difficult as the particle in the ideal place to exploit the current best solution is restrained too much.

During each of the experiments in the following section, the values of G and k were set as 1, the value of mT was -1000, and the value of cvr was 10. -1000 was chosen as the value of mT to ensure that optima were above the mass threshold.

A number of other configurations were tested, however the best results were generated using this configuration. No problem specific optimization took place. The results of this parameter optimization can be seen in table I. These results are the mean of 5 runs at 100 iterations with 50 particles. 3 functions were used as benchmarks, they are all defined in section IV of this paper.

TABLE I. RESULTS OF ITERATIVE GRID SEARCH

G	k	cvr	mT	Sphere	Matyas	Griewank
1	1	1	-1000	-0.030	-0.089	-0.043
1	2	1	-1000	-1.653	-0.116	-0.762
1	1	5	-1000	-0.011	-0.017	-0.012
1	2	5	-1000	-0.346	-0.222	-0.539
1	1	10	-1000	-0.004	-0.008	-0.005
1	2	10	-1000	-0.064	-0.209	-0.037

In addition to the formal description of the algorithm pseudocode is also provided, which can be seen in Algorithm 1. It is worth noting that despite the fact the mathematics describe an n dimensional optimization algorithm, the implementation used in this paper was only capable of optimization in 2 dimensions.

IV. INVESTIGATION

The algorithm proposed in this paper will be compared against a number of other algorithms using a range of functions. The performance metrics used reflect the algorithms ability to converge on global optima and the speed at which the algorithm converges. Two metrics are used: the best maxima found after 100 and the best maxima found after 500 iterations. Exploratory algorithms should be able to find good candidates for optima early on, and should excel at 100 iterations, whilst more exploitative algorithms should perform better at 500 iterations, assuming they did not get stuck in local optima. In addition to these comparison experiments, the effect of allowing the proposed algorithm to run for different number of iterations is also investigated, as well as the effects of increasing and decreasing the number of particles.

Algorithm 1 Gravitational Field Optimization Algorithm

```
Inputs: Input Function, Number Of Particles,
Outputs: Position Of Optima

Set Parameters G, k and mT
Randomly Initialize Positions of Particles
for i = 1 to noIterations do
  for j = 1 to noParticles do
    calculate mass of particle j
    for k = 1 to noDimensions do
      calculate the total force exerted on particle j in
      dimension k
      calculate acceleration of particle j in dimension k
      calculate updated velocity of j in dimension i
    end for
    update position of particle j
    update mass of particle j
  end for
end for
return position of particle with highest mass
```

In this paper, the algorithms are tested using 10 different test functions. The functions used in this paper are defined in the section below, some functions are inverted such that a global maxima search will take place as opposed to a global minima search. As the implementation of the proposed algorithm is only capable of optimizing two dimensional functions, all of the test functions have only two dimensions. All functions are two dimensional. The functions were taken from the following works: [16][15][3][11]

Sphere Function:

$$f(x_1, x_2) = -(\sum_{i=1}^{2} x_i^2) \tag{11}$$

Rosenbrock Function:

$$f(x_1, x_2) = -100(x_1^2 - x_2)^2 - (1 - x_1)^2 \tag{12}$$

Beale Function:

$$\begin{aligned} f(x_1, x_2) = -\ & ((1.5 - x_1 + x_1 x_2)^2 \\ & + (2.25 - x_1 - x_1 x_2^2)^2 \\ & + (2.625 - x_1 + x_1 x_2^3)^2) \end{aligned} \tag{13}$$

Rastrigin Function:

$$f(x_1, x_2) = 20 - \sum_{i=1}^{2}(x_i^2 - 10cos(2\pi x_i)) \quad (14)$$

Schwefel Function:

$$f(x_1, x_2) = -(\sum_{i=1}^{2} -x_i sin(\sqrt{|x_i|}) + 2(418.982887)) \quad (15)$$

Matyas Function:

$$f(x_1, x_2) = -0.26(x_1^2 + x_2^2) + 0.48x_1x_2 \quad (16)$$

Griewank Function:

$$f(x_1, x_2) = -1 - \sum_{i=1}^{2} \frac{x_i^2}{4000} + \prod_{i=1}^{2} \cos(\frac{x_i}{\sqrt{(i)}}) \quad (17)$$

In addition to these benchmark functions, a number of functions first defined in this paper are used. These functions have features that compliment the proposed algorithms unique characteristics which were discussed in the previous section. The functions feature large numbers of local optima in close proximity to each other, and solution spaces that are slightly sloped in one direction. These functions are defined below:

Spherewave Function:

$$f(x_1, x_2) = 10 \sum_{i=1}^{2} x_i^2 + \sum_{i=1}^{2} +0.1x_i \quad (18)$$

PyramidalFunction:

$$\begin{aligned} Pyr(x_1, x_2) =& 10|\sin(x_1) + \sin(x_2)| \\ &+10|\sin(x_2) - \sin(x_1)| + \sum_{i=1}^{2} +0.1x_i \end{aligned} \quad (19)$$

Nested Pyramidal Function:

$$f(x_1, x_2) = Pyr(Pyr(x_1, x_2), Pyr(x_1, x_2)) \quad (20)$$

In the following set of experiments, the proposed algorithm is compared to Particle Swarm Optimization [7], and a Genetic Algorithm [1]. The GA uses a binary string as a chromosome, which evaluates as 2 floating point coordinates. All three algorithms are tested with 20, 50 and 100 particles, and as stated earlier in this section, the best maxima at 100 and 500 iterations are recorded. In each experiment, all algorithms are run 100 times, and the best result is shown. For each function, the two dimensional solution space is bound as follows: $-10 \leq x_1 \leq 10$ and $-10 \leq x_2 \leq 10$. All particles are initialized with random position vectors, with values within the bound solution space. On each run, a new random seed is used.

In addition to the three optimization algorithms, a traditional iterative 'grid' search is also run on each the functions. A 'grid search' consists of a number of points being chosen and evaluated against the input function iteratively, the point yielding the best result is the optima returned by the search. Intuitively, grid searches with a higher number of rows and columns, and therefore a higher granularity, return the best results. Grid searches of 50x50 (2500 fitness evaluations), 100x100 (10000 fitness evaluations) and 500x500 (250000 fitness evaluations) are performed on each functions, and the results are shown in the table II.

TABLE II. RESULTS OF ITERATIVE GRID SEARCH

Function:	50x50	100x100	500x500
Sphere	-0.083	-0.020	-0.008
Rosenbrock	-0.044	-0.088	-0.016
Beale	-0.203	-0.224	-0.058
Rastagrin	37.753	36.085	39.841
Scwefel	-830.083	-830.078	-830.075
Matyas	0.002	0.000	0.000
Griewank	-0.021	-0.005	0.000
Spherewave	1.371	1.440	1.566
Pyramidal	10.399	11.262	11.606
Nested Pyramidal	11.054	11.048	11.881

Table II's results allow the optima of the functions to be gauged, and also provides benchmarks for the optimization algorithms to complete against. As expected, in most cases, the results improve as the granularity increases. In the set of experiments conducted later in this paper, the number of fitness comparisons will be vastly lower (2000 - 50000), compared to the 10000 - 250000 of grid search.

The results in table III show the proposed algorithm is competitive with both PSO and GA. In many cases, especially when 100 particles are used, all 3 algorithms find the global optima. On simple functions, such as Sphere and Matyas, all functions consistently find the global optima. The proposed algorithm seems to struggle when only 20 particles are used, however when this number is increased the algorithm performs better, this will be investigated more thoroughly later in this section. On almost all of the functions, the proposed algorithm outperformed the iterative search. On the three functions introduced in this paper, the proposed algorithm outperforms or is on par with all other algorithms. This shows there exists scenarios in which the proposed algorithm has advantages over the other algorithms tested in this paper.

The results in table IV show a similar trend to the results in table III, in that with the exception of the functions introduced in this paper, the proposed algorithm either performs equally well, or slightly worse than the other algorithms tested. This holds true with the exception of 'Rosenbrock' where like in the previous experiment, the proposed algorithm performs considerably worse than the others. The proposed algorithm performs well on the functions introduced in this paper. On all 3 functions, each of the algorithms appears to find the global maxima when 100 particles are used.

Next, the particle and iteration counts are modified, and the behavior of the algorithm is investigated. First, changing the iteration count is investigated. Using the Beale function and particle counts of 20, 50, 80 and 100, iteration counts between 10 and 1000 are investigated. This experiment is run 50 times and the averages are taken, and the results can be seen in graph 4.

This graph firstly shows that as the number of iterations increases, so does the average performance of the proposed

TABLE III. SHOWS THE PERFORMANCE OF THE PERFORMANCE OF THE PROPOSED ALGORITHM IN COMPARISON WITH GA AND PSO FOR POPULATION SIZE OF 20, 50 AND 100. THE RESULTS ARE BEST OUT OF 50 RUNS AND AFTER 100 GENERATIONS

Function	FO_{20}	PSO_{20}	GA_{20}	FO_{50}	PSO_{50}	GA_{50}	FO_{100}	PSO_{100}	GA_{100}
Sphere	-0.059	0.000	-0.069	0.000	0.000	0.000	0.000	0.000	0.000
Rosenbrock	-0.810	-0.020	-2.561	-0.662	-0.001	-0.422	-0.268	0.000	-0.140
Beale	-0.261	-0.122	-0.219	-0.316	-0.089	-0.228	-0.306	-0.056	-0.338
Rastagrin	37.419	39.592	38.161	40.000	40.000	40.000	40.000	40.000	40.000
Scwefel	-830.078	-830.075	-830.075	-830.079	-830.075	-830.076	-830.075	-830.075	-830.075
Matyas	-0.004	-0.000	0.008	0.000	0.000	0.000	0.000	-0.000	0.000
Griewank	-0.025	-0.002	-0.013	0.000	0.000	0.000	0.000	-0.000	0.000
Spherewave	1.878	1.256	1.837	1.881	1.257	1.877	1.885	1.571	1.881
Pyramidal	11.785	1.471	11.577	11.785	11.782	11.782	11.786	11.786	11.786
Nested Pyramidal	11.789	11.253	11.252	11.881	11.875	11.322	11.881	11.880	11.847

TABLE IV. SHOWS THE PERFORMANCE OF THE PERFORMANCE OF THE PROPOSED ALGORITHM IN COMPARISON WITH GA AND PSO FOR POPULATION SIZE OF 20, 50 AND 100. THE RESULTS ARE BEST OUT OF 50 RUNS AND AFTER 500 GENERATIONS

Function	FO_{20}	PSO_{20}	GA_{20}	FO_{50}	PSO_{50}	GA_{50}	FO_{100}	PSO_{100}	GA_{100}
Sphere	-0.001	-0.000	-0.058	0.000	0.000	0.000	-0.000	-0.000	0.000
Rosenbrock	-0.258	-0.000	-0.000	-0.162	-0.000	0.000	-0.132	-0.000	0.000
Beale	-0.074	-0.056	-0.057	-0.168	-0.056	-0.217	-0.090	-0.056	-0.057
Rastagrin	39.775	40.000	39.000	40.000	40.000	40.000	40.000	40.000	40.000
Scwefel	-830.075	-830.075	-830.075	-830.075	-830.075	-830.075	-830.075	-830.075	-830.075
Matyas	-0.005	-0.000	-0.002	0.000	0.000	0.000	0.000	-0.000	0.000
Griewank	-0.008	-0.000	-0.010	0.000	0.000	0.000	0.000	-0.000	0.000
Spherewave	1.885	1.885	1.885	1.885	1.885	1.885	1.885	1.885	1.885
Pyramidal	11.785	11.785	11.786	11.786	11.786	11.786	11.786	11.786	11.786
Nested Pyramidal	11.831	11.797	11.793	11.875	11.716	11.833	11.881	11.881	11.881

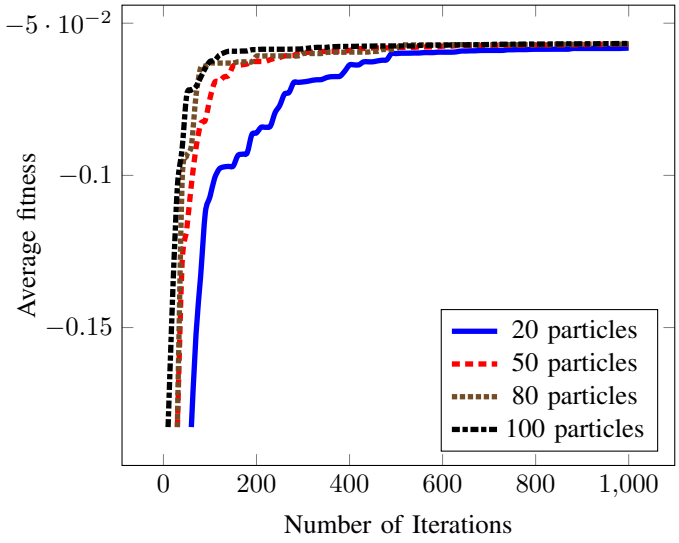

Fig. 4. Show the relation between the number of iterations and the fitness. The y-axis shows the mean fitness of the best particle over 50 runs.

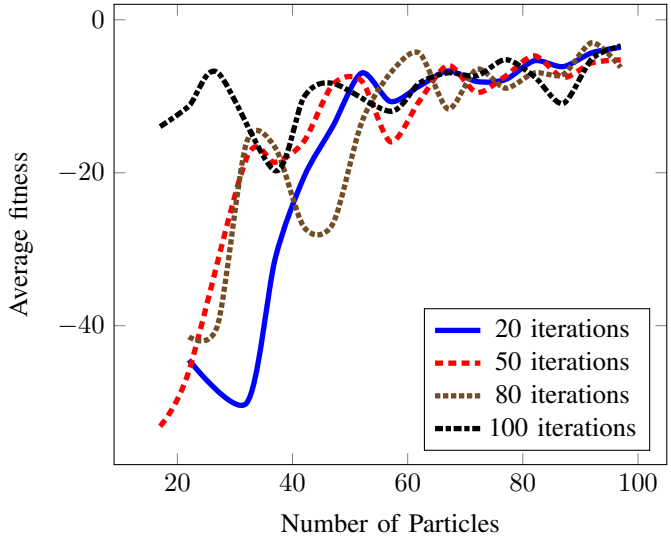

Fig. 5. Show the relation between the number of particles and the fitness. The y-axis shows the average fitness of the best particle over 50 runs.

algorithm, however it does so with diminishing returns. In addition to this, these results also imply that adding extra particles improves both the exploratory and exploitative characteristics of the algorithm. Put simply, as the number of particles in increases, the time it takes to converge to a good solution decreases. This is investigated further in the next experiment, where changing the number of particles in investigated. Iteration counts of 20, 50, 80 and 100 are tested, and particle counts between 2 and 100 are investigated, the results can be see in figure 5.

The results in figure 5 show what has already been demonstrated in the previous experiment, that performance increases as the number of particles increases. In addition to this, a certain amount of inconsistency is apparent. The inconsistency becomes less of an issue as particle counts increase, this can be seen in figure 5 in the form of the lines smoothing out. Interestingly at one point the run at 20 iterations outperforms the run at 100 iterations. The fact this didn't happen in the previous experiment implies changing the particle count has more of an effect on the outcome than changing the iteration count.

In this section, the proposed algorithm was compared to two other commonly used optimization algorithms, Particle Swarm Optimization, and Optimization using a Genetic Algorithm, as well as this, an iterative search was also compared.

The algorithm performed best on the three functions defined for the first time in this paper. This is not surprising, as the functions were developed with the proposed algorithms

previously discussed emergent behaviors in mind, and as such have characteristics that complement the proposed algorithms strong points. On the other functions, the proposed algorithm performed similarly to PSO and GA, with the results at 100 iterations being poorer than the results at 500 iterations.

Despite this, on some of the functions, the proposed algorithm performed poorly compared to the other algorithms, even when the particle count is increased and the algorithm is allowed to run for 500 iterations. For instance on the Rosenbrock and Beale functions the proposed algorithm is repeatedly considerably worse than PSO and GA. One feature of these two functions, the Beale function more so, is a large plateau in the functions' solution space, surrounding a single global optima. This sort of problem would be for simple for even an iterative gradient based optimization algorithm, yet somehow presents a problem for the proposed algorithm. On the contrary, comparing the results of a more 'hilly' function such as the Griewank function shows a different trend; here the proposed algorithm can reliably find the global optima.

When compared to the iterative 'grid search', on all functions other than Rosenbrock and Beale, on which the proposed algorithm performed poorly, the algorithm outperforms the grid search. This is comparing the maximum grid search granularity against the proposed algorithm's results using 100 particles at 500 iterations. This is unsurprising, as the iterative search is unable to focus on promising areas in the solution space. It is worth noting at this point 250000 fitness evaluations were completed for the grid search, yet only 50000 had elapsed for the proposed algorithm.

Overall, the algorithm proposed in this paper is competitive when compared to Particle Swarm Optimization, and Optimization using a Genetic Algorithm. In addition, this work has demonstrated that a set of problems exist in which the proposed algorithm outperforms the two aforementioned algorithms.

V. Conclusion

This paper proposed an algorithm that tackles the function optimization problem using a novel approach. The algorithm is described with formal mathematics, and emergent behaviours are summarized and discussed. The proposed algorithm is compared against a number of other commonly used heuristic optimizers, and a set of functions on which the proposed algorithm outperforms the competition has been identified. The effect of tweaking some of the algorithms parameters was also investigated.

Future work could involve subtly modifying the algorithm to improve performance. Taking inspiration from Simulated Annealing [12], one possible improvement to the algorithm could be to have the particles 'cool down' as the iteration count increases, allowing for an exploratory search early on, and an exploitative search after promising areas have been found. This could be done by reducing the Gravitational Constant (G). In addition to this, the implementation of the algorithm needs to be extended to optimize n dimensional problems. Implementing this will allow the algorithm to be tested against more complex functions, and an investigation into how the algorithm performs on problems with a large dimensionality can be conducted. This would also open up numerous 'real-world' optimization problems for investigation, in contrast to the abstract functions explored in this work. It would also be interesting to explore the benefits of problem specific parameter optimization on the proposed Algorithm.

Continuing to develop novel optimization techniques is important due to the amount of problems that are infeasible using deterministic time algorithms. Even if an algorithm is only capable of pushing the boundaries on a small number of those problems, it could help resolve important issues in the field.

References

[1] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[2] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the p=?np question. *SIAM Journal on computing*, 4(4):431–442, 1975.

[3] Joyita Basak, Sangita Roy, and Sheli Sinha Chaudhuri. Benchmark function analysis of cuckoo search algorithm. In *Information Systems Design and Intelligent Applications*, pages 719–730. Springer, 2015.

[4] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[5] Stephen Chen, James Montgomery, and Antonio Bolufé-Röhler. Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence*, 42(3):514–526, 2015.

[6] John J Dykla, Robert Cacioppo, and Asim Gangopadhyaya. Gravitational slingshot. *American Journal of Physics*, 72(5):619–621, 2004.

[7] Russ C Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[8] Albert Einstein and Francis A Davis. *The principle of relativity*. Courier Corporation, 2013.

[9] Zong Woo Geem and Han Hwangbo. Application of harmony search to multi-objective optimization for satellite heat pipe design. In *Proceedings of*, pages 1–3, 2006.

[10] Fred Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[11] Xing-shi He, Wen-Jing Ding, and Xin-She Yang. Bat algorithm based on simulated annealing and gaussian perturbations. *Neural Computing and Applications*, 25(2):459–468, 2014.

[12] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[13] David J Kropaczek and Paul J Turinsky. In-core nuclear fuel management optimization for pressurized water reactors utilizing simulated annealing. *Nuclear Technology*, 95(1):9–32, 1991.

[14] Lev Davidovich Landau. *The classical theory of fields*, volume 2. Elsevier, 2013.

[15] JJ Liang, PN Suganthan, and K Deb. Novel composition test functions for numerical global optimization. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 68–75. IEEE, 2005.

[16] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 2005.

[17] Todd Perry, Mohamed Bader-El-Den, and Steven Cooper. Imbalanced classification using genetically optimized cost sensitive classifiers. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 680–687. IEEE, 2015.

[18] Davison E Soper. Classical field theory. 1976.

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

[20] Xingquan Zuo, Guoxiang Zhang, and Wei Tan. Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud. *Automation Science and Engineering, IEEE Transactions on*, 11(2):564–573, 2014.