Binary Particle Swarm Optimisation for Continuous Functions

Candidate Number: 014381

ABSTRACT

The particle swarm algorithm classically operates in a continuous space through adjusting a population of "particles" within the objective space by using information about each particle's previous best performance, and the best performance from the whole population. Trajectories were defined as changes in position for a number of dimensions. The binary particle swarm redefines this in order to work in a discrete objective space by considering trajectories as changes in the probability that a coordinate will take a discrete value. The efficacy of this approach is empirically analysed using the De Jong testbed.

KEYWORDS

Particle Swarm Optimisation, Metaheuristics, Evolutionary Computation, Discrete Optimisation

ACM Reference Format:
Candidate Number: 014381. 2021. Binary Particle Swarm Optimisation for Continuous Functions. In *Evolutionary Computation and Optimisation.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Nature inspired optimisation algorithms have seen a growing interest in recent decades. Numerous approaches based on natural phenomenon have been shown by researchers that such optimisers are effective tools for solving difficult computational problems. Metaheuristic approaches which have been explored by researchers include: genetic algorithms, tabu search, simulated annealing, ant colony and particle swarm optimisation (PSO). PSO is classified as a swarm intelligence and was originally developed by Kennedy and Eberhart in 1995 [4], who aimed to simulate social behaviour of artificial life such as bird flocking and fish schooling. PSO has since been found to be capable of generating superior quality solutions within shorter computation times on some optimisation problems. Originally PSO was designed for use in the continuous space however many optimisation problems are set in discrete spaces, typically represented with binary strings as they are easy to encode, and apply operators such as crossover or mutation. This makes the problem combinatorial in difficulty as the algorithm must find the optimal allocation of ones and zeros within a fixed length string. This means the solution space quickly becomes intractable for larger problems, hence there is a clear need to employ meta-heuristics.

ECMM423P '21, April 26–26, 2021, ECMM423P, Exeter

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Particle swarm is typically applied to real-valued continuous problems where it is successful as it adjusts trajectories through the manipulation of each particle's coordinates. This enables it to "overfly" known local optima, allowing it to explore beyond as well as between them. To do this in a discrete space, the key concepts of the algorithm such as trajectory, velocity, and betweenness will need to be suitably redefined.

This makes PSO interesting when used on binary strings as every particle is now expressed as a binary vector, and the velocity of a particle is now a probability vector, where each probability element determines the likelihood of that binary variable taking the value of one. The challenge is then to determine whether the optimiser is still able to converge correctly.

2 LITERATURE REVIEW

Optimisation problems are set in a space featuring discrete, qualitative distinctions between variables. Such problems typically require the ordering or arranging of discrete elements such as binary digits. Any discrete or continuous problem can be expressed in a binary notation, therefore it is advantageous to investigate optimisers which operate on such problems [5].

PSO originates from the simulation of social behaviour. A population of individuals is updated through an operator which pertains to the fitness information obtained from the environment in order for the individuals to move towards better solution areas [6].

PSO is similar to evolutionary algorithms in that it searches using a population of particles, where each particle represents a candidate solution to the problem at hand. Instead of reproducing with each other to form new solutions (such as in genetic algorithms), the particles change their positions by traversing a multidimensional search space until a relatively unchanged position is encountered, or a limit is reached [4].

To optimise the given objective function, each particle knows its own best value so far (p_best_i) and the corresponding position. Each particle also knows the best value so far from the entire population (g_best). Particles then modify their position using the distance between their current position and p_best; and the distance between their current position and g_best. This concept is represented by velocity (v_i) which is calculated for each particle every generation. A particles new position is their previous position plus their velocity.

The velocity and the position of each particle in a d-dimensional space is calculated by:

$$v_{id}(t+1) = w.v_{id}(t)+c_1.rand(p\_best_{id}-x_{id})+c_2.rand(g\_best_d-x_{id}) \quad (1)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (2)$$

where $rand()$ is a random functions uniformly distributed in the range $[0, 1]$, c_1, c_2, and w are positive constants. $x_i = (x_{i1}, x_{i2}, ..., x_{id})$ represents the position of particle i. The velocity of particle i is represented by $v_i = (v_{i1}, v_{i2}, ..., v_{id})$. The best previous position of

particle i is shown by:
$p\_best_i = (p\_best_{i1}, p\_best_{i2}, ..., p\_best_{id})$.

When a particle moves through the search space it compares its fitness value at the current position to p_best_i.

Some of the success of PSO can be attributed to it "overflying" known local optima, allowing it to explore beyond as well as between them. Therefore the concepts of trajectory, velocity, and betweenness need to be defined for the discrete space [5].

In a binary search landscape, a particle moves to nearer and further corners of the hypercube by flipping different numbers of bits. Thus for binary PSO (BPSO) the velocity of the particle overall can be described by the number of bits changed per iteration, or by the Hamming distance between the particle at time t and at $t+1$. A particle where none of the bits have been flipped will not move whereas a particle with every bit flipped will move the "furthest".

To define v_{id} a trajectory needs to be defined in terms of changes of probabilities that a singular bit will be in one state or the other [5]. On each dimension, a particle should only be allowed to move in a state space restricted to zero or one on each dimension where every v_{id} represents the probability of bit x_{id} being 1. If $v_{id} = 0.20$, then there is a twenty percent chance that x_{id} will be a one, and an eighty percent chance it will be a zero. It follows that p_{id} and x_{id} are integers in $\{0, 1\}$, whereas v_{id} is a probability hence it is constrained to the interval $[0.0, 1.0]$.

A logistic transformation $S(v_{id})$ can be used to update the position appropriately where

$$S(v_{id}) = Sigmoid(v_{id}) = \frac{1}{1 + e^{-v_{id}}} \tag{3}$$

$$if\ rand() < S(v_{id}(t+1))\ then\ x_{id}(t+1) = 1\ else\ x_{id}(t+1) = 0 \tag{4}$$

where $S(v_{id})$ is a sigmoid limiting transformation and *rand* is a quasi-random number selected from a uniform distribution in $[0.0, 1.0]$. The continuous version of PSO limits v_{id} by the value v_{max}. In BPSO v_{id} is limited in the range $[-v_{max}, v_{max}]$. Usually $v_{max} = 6.0$ [5]. This limits the probability to $[0.0025, 0.9975]$ but results in better convergence characteristics. v_{max} is partly used to set a limit to further exploration after the population has converged.

In [6], it was found that when using BPSO on a binary string problem, the average cost function initially improves but struggles to converge to the global optima.

In [5], it was found that BPSO is capable of solving the De Jong [2] testbed functions rapidly. It struggled on problems with good local optima as it does not use crossover which better permits larger leaps across the search space.

In [7], they found BPSO to outperform a traditional genetic algorithm for finding optimal solutions to the lot sizing problem. They found BPSO advantageous due to there being very few parameters to deal with, and the large number of dimensions which enable it explore the solution space effectively. Conversely, it converges to a solution very quickly therefore should be used cautiously for combinatorial optimisation problems which inherently have very large search spaces.

There are two main disadvantages to BPSO [6]. Firstly, when the algorithm is approaching the optimal solution, the particles diverge and instead may become trapped in a local optima. This is caused by the probability of changing the position of the particle not being near to zero. Instead the sigmoid function means the position will change with probability of 0.5 when the velocity is 0. Secondly, in the position updating equation, the previous position is not considered when updating to the new position.

3 FORMALISATION OF THE PROBLEM AS OPTIMISATION PROBLEM

In [2], five test functions were designed in order to test a carefully constructed set of dimensions of performance of an optimiser. The problems include instances of continuous, discontinuous, convex, non-convex, unimodal, multimodal, quadratic, non-quadratic, low-dimensional and high-dimensional functions as well as a function with Gaussian noise.

Each function was restricted to the bounded subspace of R^n of the form $a_1 \leq x_1 \leq b_1$, $i = 1, ..., n$. Within R^n each function was made discrete by specifying a resolution factor Δx_1 for each axis.

Test function F1 is given by:

$$F1(x) = \sum_{i=1}^{3} x_1^2 \tag{5}$$

F1 is a continuous, convex, unimodal, low-dimensional quadratic function with a maximum of 78.6 at the domain's bounds. F1 was restricted to the space A defined by $-5.12 \leq x_1 \leq 5.12$, $i = 1, 2, 3$ with a resolution factor 0.01 along each axis. The search space A consists of $(1024)^3 \cong 10^9$ solutions.

Test function F2 is given by:

$$F2(X) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \tag{6}$$

F2 is a continuous, non-convex, unimodal, low-dimensional quartic function with a maximum of 3905.93 at the domain's lower bounds. It is a difficult minimisation due to it having a deep parabolic valley along the curve $x_2 = x_1^2$. F2 was restricted to the space A defined by $-2.048 \leq x_1 \leq 2.048$, $i = 1, 2$ with a resolution of 0.001 on each axis. The search space A consists of $(4096)^2 \cong 10^6$ solutions.

Test function F3 is given by:

$$F3(X) = \sum_{i=1}^{5} \lfloor x_1 \rfloor \tag{7}$$

F3 is a 5-dimensional step function. It is a discontinuous, non-convex, unimodal function of moderate dimension which is piecewise constant. It is used as a test for handling discontinuities. F3 was restricted to the space A defined by $-5.12 \leq x_1 \leq 5.12$ with a resolution of 0.01 along each axis. The search space A consists of $(1024)^5 \cong 10^15$ solutions.

Test function F4 is given by:

$$F4(X) = \sum_{i=1}^{30} ix_i^4 + \text{GAUSS}(0, 1) \tag{8}$$

F4 is a continuous, convex, unimodal, high-dimensional quartic function with Gaussian noise. F4 was restricted to the space A defined by $-1.28 \leq x_i \leq 1.28, i = 1, ..., 30$ with a resolution of 0.01 along each axis. The search space A consists of $(256)^30 \cong 10^72$ solutions.

Test function F5 is given by:

$$F5(X) = \{0.002 + \sum_{i=-2}^{2} \sum_{j=-2}^{2} [5(i+2)+j+3+(x_1-16j)^6+(x_2-16i)^6]^{-1}\}^{-1} \quad (9)$$

F5 is an interesting multimodal function. It is continuous, non-convex, non-quadratic, two-dimensional function. F5 was restricted to the space A defined by $-65.536 \leq x_i \leq 65.536$ with a resolution of 0.001 along each axis. The search space A consists of $(131,072)^2 \cong 10^1$ solutions.

4 DESCRIPTION OF SEARCH ALGORITHM

Algorithm 1: Pseudocode for Binary Particle Swarm

```
Input  : ProblemSize, Population_size
Output : P_g_best
Population ← ∅;
P_g_best ← ∅;
for i = 1 to Population_size do
    P_velocity ← RandomVelocity();
    P_position ← RandomPosition(Population_size);
    P_p_best ← P_position;
    if Cost(P_p_best) ≤ Cost(P_g_best) then
        P_g_best ← P_p_best;
    end if
end for
while ¬ StopCondition() do
    foreach P ∈ Population do
        P_velocity ← UpdateVelocity(P_velocity, P_g_best,
          P_p_best);
        P_position ← UpdatePosition(P_position, P_velocity);
        if Cost(P_position) ≤ Cost(P_p_best) then
            P_p_best ← P_position;
            if Cost(P_p_best) ≤ Cost(P_g_best) then
                P_g_best ← P_p_best;
            end if
        end if
    end foreach
end while
return P_g_best
```

The swarm's aim is to have its particles locate the optima in a multi-dimensional hyper-volume. First, random positions are assigned to all particles in the space with small initial random velocities. The algorithm then simulates the swarm by advancing the position of every particle based on its velocity, the best known global position in the problem space and the best position known to a particle. The objective function is then sampled after each position update. As the swarm explores and exploits good positions within the search space, the particles will cluster or converge together around a global optima, or several local optima.

5 APPLICATION OF ALGORITHM TO PROBLEM

The fitness function is designed to optimise the objective function by encapsulating the decision variables into a singular fitness score. This is done by converting the binary string representation operated on by the algorithm into the real valued decision variables, passing these to the objective function, and then calculating the absolute value between the obtained value and the global optima. This difference is then minimised by the optimiser.

BPSO operates on binary strings thus the real valued decision variables should be represented as such. This is done by first calculating the size of the search space for each decision variable so that the minimal number of bits is used to encode each variable. Using too many bits will make the search space exponentially larger which would in turn would increase the difficulty of the problem unnecessarily. The objective search space A is bounded by the domain of each problem, and the precision used. A binary string representing a base two number can then be mapped to a solution using the following formula:

$$x_i = a_i + (\Delta x.x_i^*) \quad (10)$$

where x_i represents the decision variable, a_i represents the lower bound, Δx represents the precision, and x_i^* represents the encoded base two value of the decision variable. This allows real valued numbers to be represented as positive binary integers which can be manipulated inside the given constraints of the problem. Representing the decision variables by using the IEEE Short Real format directly would have the disadvantage of the sign and exponent bits having vastly more influence than the other bits. The optimiser however would not have knowledge of this difference in influence and therefore would not be able to take it into account.

Each binary string is then mapped to a vector of velocities such that each bit corresponds to a velocity. Every particle in the swarm has its own velocity vector. A velocity, after the sigmoid transformation function, represents a probability that a bit will be either zero or one. The binary bits themselves do not have a value until they are evaluated, therefore the actual position of the particle in D dimensional space is ephemeral. This means an individual could take a different position at every iteration.

In order to calculate the social and cognitive values, the traditional PSO uses Equation 1. This equation is designed for continuous values whereas our representation is discrete. To counter this, we calculate the Hamming distance between the previous best performance and the current coordinate for the cognitive value, and we calculate the Hamming distance between the global best position and its current position for the social value. By using the Hamming distance we can see the number of bits changed per iteration, which is in turn the velocity of the particle.

A binary string representation is meaningful for solving discrete problems as it allows for the manipulation of individual bits. This makes it easier to perform changes to parts of the encoding.

6 EXPERIMENTAL SETUP

To investigate the effectiveness of BPSO for solving real valued problems represented as binary strings, it is important to evaluate the optimiser on some benchmark functions. De Jong's test bed

of benchmark functions will be used to empirically analyse the performance of BPSO. These functions were chosen because they represent the common difficulties in optimisation problems in an isolated manner. By using these functions to make comparisons, one can see the strengths and weaknesses of an optimiser.

In order to evaluate the robustness of BPSO, comparisons will be drawn against a Random Search algorithm, and a Stochastic Hill Climbing algorithm (SHC) [3]. Random search was chosen as it does not utilise any guiding mechanisms for exploring the problem space therefore if an optimiser produces worse results than Random Search, it can considered detrimental to the optimisation of a problem. SHC iterates the process of randomly selecting a neighbour for a candidate solution, and only accepts it if there is an improvement. Neither algorithm require derivatives of the search space.

In order to draw fair comparisons between the optimisers, an equal number of fitness function evaluations should be performed. Therefore the population size and iteration number can vary between optimisers provided the number of fitness function evaluations is consistent. BPSO will have a population of 20 for 200 iterations making for 4000 fitness function evaluations. A parameter tuning experiment was performed in order to find the optimal step size for each benchmark problem when using the SHC. The step size is a distance within the search space which determines how close the new point can be from the old point. The step size should be big enough to allow fitter close points in the search space to be located, and small enough so that the search does not jump out of regions that contain the local optima. For each problem, the algorithm was run with a step size parameter s where $0 \leq s \leq 1$, where s increased by 0.1 each iteration. The difference between the result and the optima was averaged over 100 runs, with the parameters causing the least error for problem selected. The parameters were $[1.0, 1.0, 0.9, 0.7, 0.8]$ for problems $[F1, F2, F3, F4, F5]$ respectively.

BPSO has three parameters, the inertia, cognitive, and social coefficients. These coefficients control the trade-off between exploration and exploitation. Exploitation is the ability of particles to target the best solutions found so far. Exploration on the other hand, is the ability of particles to evaluate the entire objective space. It is therefore important to find a good balance between the two. The parameter w determines the ability of the swarm to change its direction. The particles inertia is proportional to this coefficient. The lower the coefficient w, the stronger the convergence thus a low w encourages the exploitation of the best solutions found so far whereas a high coefficient w encourages exploration around existing solutions. The inertia weight w sets the balance between the exploration and the exploitation of the best solutions found so far. To find these existing solutions, the acceleration coefficients $c1$ and $c2$ need to be suitably set. The $c1$ parameter lets the group be influenced by the best personal solutions found hitherto, conversely the $c2$ parameter lets the group be influenced by the best global solution found hitherto. The coefficients of $c1$ and $c2$ are complementary thus a combination of the two increases both exploration and exploitation.

According to the paper [1], the best static parameters for PSO are $w = 0.72984$ and $c1 = c2 = 2.05$. It was decided to use these parameters rather than perform an experiment similar to the SHC parameter tuning because the combinatorial nature of potential parameters did not make the experiment feasible.

In order to see how well the algorithm has optimised the given function, we shall use two performance measures. Firstly, we examine the absolute difference between the obtained optima and the true optima. This shows us how well the objective function has been optimised in the decision space. This allows us to directly compare to values other optimisers have achieved. Secondly, we will examine the Hamming distance between the obtained binary string and the binary string which represents the most optimal value. This lets us see how well the problem has been optimised in the objective space. If there is a Hamming distance of 0 in the objective space, but the solution is not optimal in the decision space then we know that the encoding is the reason an optimal solution has not been found.

In order to analyse the strengths and weaknesses of BPSO, as well as draw comparisons with random search and stochastic hill climbing, each optimiser will be run on each benchmark function. The optimisers will be run 30 times each, with the mean of the best performing member from each population taken at each time step. It is important to see results throughout the duration of the optimisation procedure as this lets us observe how long it takes for the population to converge, which would not be possible by only examining the end result.

7 ANALYSIS

Figure 1 shows that BPSO converged quickly towards the optimum and did not diverge for the remaining iterations. Table 1 shows that it found the optimal binary encoding on all 30 runs, however it never found the exact solution despite its quick convergence. This is likely due to the encoding not being precise enough as the shown by the optimal Hamming distance being found for every run. Random search was shown to outperform the SHC. The SHC had a much greater standard deviation (STD) suggesting it was consistently getting stuck in local optima.

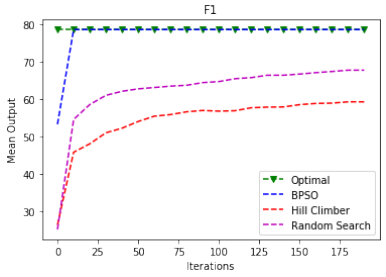

Figure 1: A comparison between Binary Particle Swarm, Stochastic Hill Climber, Random search and the optimal solution on the test problem F1.

F2 was more challenging for the particle swarm as it was unable to find the optimum on any of the runs. It produced the largest STD for any of the benchmark functions. The SHC performed the best with a mean error of 6.3565. This was significantly better than BPSO and RandomSearch whose means were both over 100.

Table 1: Table of results for the optimiser being run on the benchmark functions F1, F2, F3. The mean and standard deviation (STD) are calculated from the difference between the found value and the optimal value over 30 runs. Opt shows how many times from 30 runs the maximum of the objective function was found while Ham-Opt shows how many times out of 30 the optimal encoding was found by the Binary Particle Swarm.

	F1				F2				F3			
	Mean	STD	Opt	Ham-opt	Mean	STD	Opt	Ham-opt	Mean	STD	Opt	Ham-opt
Binary Particle Swarm	0.0432	0.0	0	30	127.66	99.346	0.0	0.0	0.0	0.0	30.0	30
Stochastic Hill Climber	10.145	14.377	0	-	6.3565	7.8827	0.0	-	19.5	6.3966	0.0	-
Random Search	5.4661	1.8032	0	-	174.65	77.360	0.0	-	6.1333	1.0242	0.0	-

Table 2: Table of results for the optimiser being run on the benchmark functions F1, F2, F3. The mean and standard deviation (STD) are calculated from the difference between the found value and the optimal value over 30 runs. Opt shows how many times from 30 runs the maximum of the objective function was found while Ham-Opt shows how many times out of 30 the optimal encoding was found by the Binary Particle Swarm.

	F4				F5			
	Mean	STD	Opt	Ham-opt	Mean	STD	Opt	Ham-opt
Binary Particle Swarm	154.81	23.582	0	0	0.0002	0.0	0	0
Stochastic Hill Climber	0.2368	0.1908	0	-	0.79735	2.2205	0	-
Random Search	744.61	15.346	0	-	0.0002	0.0	0	-

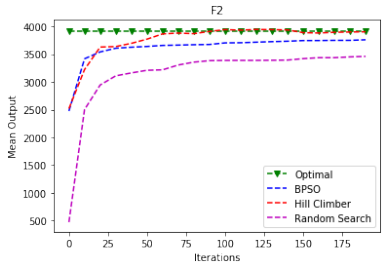

Figure 2: A comparison between Binary Particle Swarm, Stochastic Hill Climber, Random search and the optimal solution on the test problem F2.

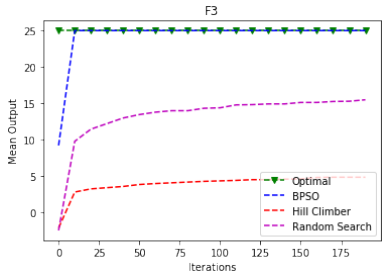

Figure 3: A comparison between Binary Particle Swarm, Stochastic Hill Climber, Random search and the optimal solution on the test problem F3.

BPSO was able to attain the target value on every run for F3. This was optimal for both the objective and decision space. Figure 3 shows that the particle swarm was able to quickly converge within the first 25 iterations. Random Search performed significantly better than SHC, having a mean error of 6.1333 compared to 19.5. Neither of the two were able to find the optimal solution in any of the runs.

The benchmark function F4 introduces gaussian noise to the objective function. This had an impact on the results as the SHC was now the best performing. It was not able to find the optimal solution however it had a mean error of 0.2368, with a similarly low STD of 0.1908 showing it could adequately handle the gaussian noise. In comparison, BPSO appeared to get stuck in a local optima as its convergence towards the optima stalled after the first 50 iterations. This suggests it became stuck in a local optima which it was unable to escape from. The random search performed poorly. It had the largest mean error and struggled to get close to the optima.

F5 proved an interesting challenge for the BPSO as it immediately converged towards a near optimal solution, however it was unable to find the optimal solution in both the objective and decision space.

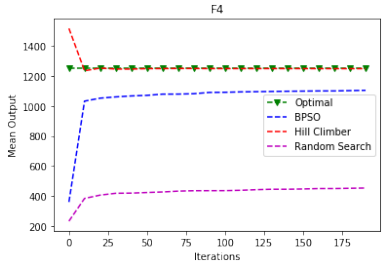

Figure 4: A comparison between Binary Particle Swarm, Stochastic Hill Climber, Random search and the optimal solution on the test problem F4.

This suggests the inaccuracy was in the optimiser itself rather than the encoding used. Interestingly, the Random Search was also able to find a near optimal solution, and never converged on the true optima. BPSO and Random Search shared the same mean error and

STD. SHC performed poorly in comparison, but still achieved a mean error of 0.79735.

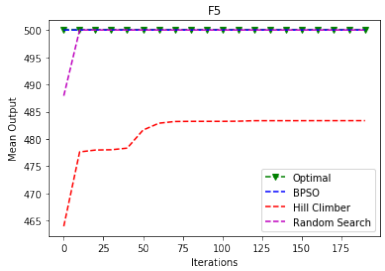

Figure 5: A comparison between Binary Particle Swarm, Stochastic Hill Climber, Random search and the optimal solution on the test problem F5.

The test function F4 had the largest objective search space with 10^{72} solutions. For comparison, the second largest was F3 with 10^{15} potential solutions. The SHC greatly outperformed BPSO for F4 showing that it was not sensitive to a larger search space whereas BPSO and Random Search are. BPSO was not only sensitive to a larger search space however as it struggled to optimise F2 which had the smallest objective search space. This difficulty can instead be explained by the existence of very good local optima in regions which are distant from the best known optimum. This shows the search is very unlikely to move from one of the local optima to the global optimum. These results demonstrate that BPSO struggles to make large jumps through the objective space.

8 DISCUSSION

The experimentation found that BPSO was very effective at solving some of the optimisation problems as it was able to very quickly converge, however it tended to get trapped in local optima close to the global optima. BPSO was not sensitive to whether the objective function was continuous or discontinuous; convex or non-convex; nor to whether it was uni or multi-modal.

The experimentation showed that the encoding played an important role in whether BPSO could find the true global optima rather than an approximate answer. This was highlighted in F1 where the optimum was never reached in the decision space but was achieved in the objective space as shown by a Hamming distance of 0 for all 30 runs. This shows that the effectiveness of the optimiser depends heavily on how well the problem can be encoded. This would depend on the bounds of the problem and the precision used. Overall, BPSO could be seen to effectively solve continuous problems encoded with discrete variables.

Problems F2 and F4 were particularly challenging as they contained deceptive local optima, and a large search space, respectively. The particle swarm remembers past successes, and therefore it tends to converge towards regions of the search space which previously had a good fitness. It does not have a mechanism for orchestrating large leaps from one region to another which would explain why it struggled on F2 which has fit local optima, and on F4 which has a very large search space which would require larger leaps in order to explore properly. BPSO struggles to make large leaps because a larger probability does not equate to a larger change in position. The algorithms inability to converge to the global optima can be explained by there being a probability of 0.5 when the velocity is 0 thus it tends to diverge around the global optima.

Future work should be done to investigate the effect of changing the parameters of BPSO in order to greater explore the search space. Ideally the parameters would be dynamic in order to adapt to the problem throughout the iterations. This would allow the swarm to encourage exploration initially, and then as the search progresses it would be able to exploit the best results after exploration by converging towards the global optimum.

9 CONCLUSIONS

PSO aims to replicate swarm behaviour seen in nature such as flocks of birds and schools of fish. An iterative simulation of this mimicked behaviour enabled the original PSO algorithm to successfully optimise continuous valued functions. The approach was then modified in order to optimise problems that exist in a discrete space. This meant aspects of the algorithm had to be re-engineered, specifically the trajectory and velocity. The velocity is represented as the number of bits changed per iteration, while the trajectories of each bit were defined as probabilities. The algorithm was tested on De Jongs benchmark functions against a random search and a SHC algorithm. The experiments found two key findings. Firstly, the representation of a real-valued variables as a binary string was instrumental in whether the algorithm was able to deduce the global optima rather than an approximation. The encoding was not always found to be sufficient. Secondly, it was found that the algorithm struggled to converge when large leaps in the fitness landscape were required. This was attributed to greater probabilities not equating to greater changes in position. Overall, the empirical analysis supported the previous findings [5], [6], that BPSO was able to converge very quickly towards near optimal solutions. It consistently outperformed random search and often the SHC thereby proving it a useful optimisation tool, however its performance was very dependent on the fitness landscape. The problems it could optimise, it would do so every time, whereas the problems it struggled with proved challenging. Problems that require large jumps would therefore be better suited for algorithms which employ a genetic crossover operator.

REFERENCES

[1] M. Clerc and J. Kennedy. 2002. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* 6, 1 (2002), 58–73. https://doi.org/10.1109/4235.985692

[2] J. De and A. Kenneth. 1975. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* Ph.D. Dissertation. USA. AAI7609381.

[3] S. Forrest and M. Mitchell. 1993. Relative Building-Block Fitness and the Building-Block Hypothesis. In *Foundations of Genetic Algorithms*, L. DARRELL WHITLEY (Ed.). Foundations of Genetic Algorithms, Vol. 2. Elsevier, 109–126. https://doi.org/10.1016/B978-0-08-094832-4.50013-1

[4] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, Vol. 4. IEEE, 1942–1948.

[5] J. Kennedy and R. Eberhart. 1997. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, Vol. 5. IEEE, 4104–4108.

[6] H. Nezamabadi-pour, M. Rostami-Shahrbabaki, and M. Maghfoori-Farsangi. 2008. Binary particle swarm optimization: challenges and new solutions. *CSI Journal on Computer Science and Engineering* 6, 1 (2008), 21–32.

[7] M Fatih Taşgetiren and Yun-Chia Liang. 2003. A binary particle swarm optimization algorithm for lot sizing problem. *Journal of Economic and Social Research* 5, 2 (2003), 1–20.