Buggy Pinball: A Novel Single-point Meta-heuristic for Global Continuous Optimization

Vasileios Lymperakis[1](✉) and Athanasios Aris Panagopoulos[2]

[1] Technical University of Crete, Chania, Greece
vasilis@lyberakis.gr
[2] California State University, Fresno, CA, USA

Abstract. In this work, we propose a fundamentally novel single-point meta-heuristic designed for continuous optimization. Our algorithm continuously improves on a solution via a trajectory-based search inspired by the pinball arcade game in an anytime optimization manner. We evaluate our algorithm against widely employed meta-heuristics on several standard test-bed functions and various dimensions. Our algorithm exhibits high precision, and superior accuracy compared to the benchmark, especially when complex configuration spaces are considered.

1 Introduction

Continuous optimization problems rely on optimization variables that draw their values from a non-countable set—typically a range of real numbers [20]. This is in contrast to discrete (combinatorial) optimization where the optimization variables draw values from a countable set. Many problems in various domains can be formulated and tackled as continuous optimization tasks. These range from image processing [4], and chemical engineering [27] to finance [24], and biology [26]. Additionally, machine learning techniques heavily rely on continuous optimization to optimize model parameters in order to, typically, minimize an error function [22]. Continuous optimization has drawn considerable attention [13].

Continuous optimization methods can be classified into either local or global ones. Local methods, such as naive Gradient Descent and Continuous Hill Climbing, move locally over the optimization space and aim to precisely locate the locally optimal solution. As such, they tend to be quite fast and have been used widely in numerous applications (e.g.,[3, 12]). However, despite the aforementioned advantages, such methods converge to local optima when operating on non-convex configuration spaces. On the other hand, global methods aim to find the global solution, typically by moving in a less restricted manner over the optimization space. Such approaches range from meta-heuristics to modified local search, such as gradient descent using momentum [21] and adaptive subgradient methods such as Adagrad [6]. In the absence of analytical solutions, such methods approximate the global optimum—in contrast to local methods

L. Rutkowski et al. (Eds.): ICAISC 2022, LNAI 13589, pp. 264–276, 2023.
https://doi.org/10.1007/978-3-031-23480-4_22

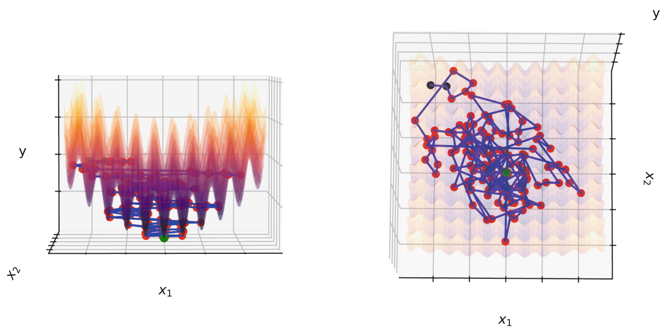

Fig. 1. Search trajectory (blue line) and coalitions (red dots) of BP on Rastrigin 3D (Color figure online)

that precisely locate a local one. Given that the objective space is usually non-convex and that one is typically interested in an approximation of the optimal solution, global methods are of great interest and widely used in practice [2].

Meta-heuristics have long been used for global continuous optimization tasks (e.g.,[5,25]). They can avoid convergence to local optima and scale to multidimensional problems, while typically not requiring a derivative of the objective function, which is a restrictive requirement of gradient-based optimization. Meta-heuristics can be broadly classified into single-point or population-based ones. The former focus on improving a single solution, while the latter on improving a collection of points based on population characteristics. Single-point meta-heuristics are generally regarded as less-fit for continuous optimization and related research is limited when compared to population-based approaches. Most single-point meta-heuristics—such as simulated annealing (SA) [15] and threshold accepting (TA) [7]—have been developed for discrete optimization problems. As such, their deployment in continuous optimization often requires non-straightforward tuning. On the other hand, population-based approaches have received considerably more attention for continuous optimization. Many such approaches have been proposed for continuous optimization over the past years, such as grey wolf optimizer [18], whale optimization [17] and particle swarm optimization (PSO) [14]. Such approaches typically demonstrate fast convergence and an ability to converge to the global optima with high precision. Nevertheless, they can still fall into local optima especially in complex configuration spaces.

Against this background, we propose a fundamentally novel single-point meta-heuristic algorithm, namely *buggy pinball* (BP), designed specifically for global continuous optimization. Our algorithm is inspired by the movement of a ball's collision and descent in the well-known pinball arcade game. Importantly, BP, is an *anytime* algorithm: it always improves over the solution while ensuring exploration. We evaluate our approach against three well-known single-point and population-based meta-heuristics on several commonly employed optimization test-bed functions and a number of dimensions. We show that BP is able to find the global optima with high accuracy and precision and in a shorter time than the benchmark. Especially, when more complex functions are considered, we show consistency on high performance, unlike any other meta-heuristic in our experiments. We believe that the superiority of our approach, and the fact that a solution is guaranteed to improve over time makes it a better choice compared to the benchmark for continuous optimization tasks, and especially those of high complexity, such as the ones that typically emerge in machine learning optimization tasks. Notably, BP's superior performance also highlights the potential of single-point meta-heuristics compared to population-based ones for continuous optimization. We sum up our main contributions as follows:

- We propose BP, a fundamentally novel single-point anytime meta-heuristic, inspired by the pinball game, which is tailored for continuous optimization.
- We experimentally show that BP ensures exploration without having to accept worse solutions, which is a common practice in meta-heuristic search.
- We evaluate our approach against widely employed meta-heuristics, on several test-bed functions and a number of dimensions.
- We show that BP performs better than both the single-point and population-based approaches considered, especially in complex configuration spaces—which proves the efficiency of our novel way of searching.

The rest of the paper is structured as follows. We first discuss background material and related work. Then, we detail our approach and discuss core motivational aspects. Subsequently, we conduct a systematic evaluation, discuss the evaluation results, and finally, conclude and present directions for future work.

2 The Buggy Pinball (BP) Algorithm

Our work is motivated by the ball's movement in the pinball game. In this game, a ball is thrown to the highest point, and by moving inside a glass-covered cabinet, it heads towards the lowest point. The player uses paddles to evade the ball from falling at the lowest point and collects points by hitting various targets. The main challenge originates from the multiple collisions of the ball, which eventually lead the ball to the lowest point. We imitated this movement by creating a trajectory-based search method, where the "ball" is moving until a collision with the objective function takes place, which occurs at the common point between the objective function and the trajectory segment—see Fig. 1. The trajectory segments start almost horizontally and become steeper with time. A trajectory segment corresponds to one round of our search algorithm. The intuition behind this movement is that when the ball is moving almost horizontally, the probability of getting into a local optimum is small, as shown in Fig. 2a, while steeper segments speed up convergence to an optimum as time progresses. Anytime algorithms are algorithms that increase the quality of the output as time progresses [10]. In BP, the trajectory segments that are progressively created, direct the search towards values that can only better optimize the cost axis. So, every new point that is detected, is guaranteed to be better than the current. Thus, Buggy Pinball is anytime, since every new segment can only provide a better solution.

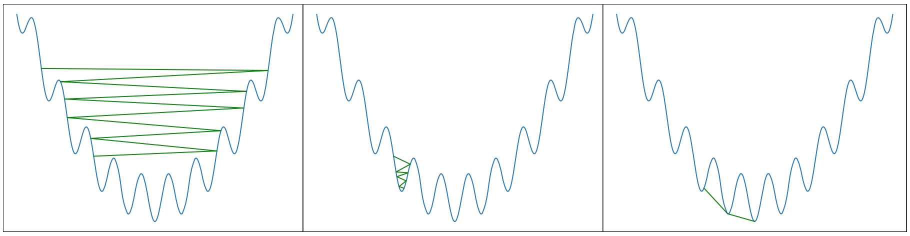
(a) Trajectory Movement (b) Entering local minimum (c) Avoiding local minimum

Fig. 2. Description of trajectory movement

Why Buggy? Even though there is a chance of getting into a local optimum solution, it is existent. To significantly decrease the possibility of converging to a local optimum, we identified the need for the pinball game to be "buggy". Instead of bouncing away, as in the real pinball, the "ball" in our algorithm is capable of continuing the search underneath the configuration space. This way, not only are local optima effectively escaped, but there is also no need to accept worse solutions to ensure exploration, unlike most meta-heuristic approaches. Thus, BP is an any-time algorithm. An illustration of being trapped into a local minimum is shown in Fig. 2b. In Fig. 2c, we can also see how BP creates trajectories underneath the configuration space and avoids convergence to a local minimum. This behavior allows BP to reduce significantly the probability of getting stuck into local optima. The BP effectiveness in avoiding local optima is verified empirically by its increased accuracy, demonstrated in our results.

BP Overview. As discussed, BP searches the space by creating trajectories. These progress via steps in a continuous simulation manner until a collision between the function and the ball occurs. Once we find that a collision occurs between two steps, a routine to precisely locate the point of collision begins. When the exact point is identified, we create a new trajectory segment starting from that point and repeat the procedure. The trajectories are created with a random direction to ensure that the configuration space is searched adequately. The elevation angle though, (i.e., the angle of descent or ascent of the segment, depending on whether we minimize or maximize a function) follows a predefined schedule: it starts at an almost horizontal level, in order to avoid local optima and becomes steeper over time to achieve faster convergence rates. By contrast, the step size is reduced over time, to achieve higher precision. As we get to bigger configuration spaces, we choose higher starting step values and smaller elevation angles. As mentioned, the trajectories are even able to advance the ball underneath the configuration space of the function. An example of a BP search on the 3D version of the Rastrigin function (Fig. 1.h in the Appendix) can be seen in Fig. 1. Each blue line represents one trajectory segment, while each red point is the common point between the function and the segment.

The BP in Detail. BP (Algorithm 1) is composed of five main parts: (1) initialization, (2) trajectory segment creation, (3) stepping forward, (4) recursive refining, and (5) cooling schedules. We note that each round corresponds to one trajectory segment, which progresses in steps. Also, an objective function is characterized by its cost (i.e. the y values) and its variables (i.e. the x_i values). Every x_i is an axis in the configuration space (see Fig. 1 for an example with two x_i variables).

Initialization. The first part (i.e., Algorithm 1, lines 1-2) is that of initializing our hyper-parameters. First, we set the original step and elevation angle, a. The original step size should be set in such a way, that we do not make too large steps in the configuration space at the beginning of the process. We have empirically found that the choice of an original step size at ∼10% of the average variable range performs well (in general, the step size should be bigger as the configuration space becomes bigger). We note here that the sign of the step (variable *stepSize* in Algorithm 1) should be negative for minimization tasks and positive for maximization.

The elevation angle is defined with respect to the plane perpendicular to the y axis. For the elevation angle we always begin with a value close to 0 (e.g., 0.1) in order to perform a near-horizontal movement.[1] We have also identified that its value should be smaller in large spaces (in contrast to that for step) since a more horizontal movement is required, to effectively avoid local optima. The same "smaller values" rule applies as the number of dimensions increases.

In line 1, we also set the number of rounds, which corresponds to the number of trajectory segments to be created as well as the number of steps to be executed in each segment. The number of rounds should be set as high as possible, considering the optimization time constraints. With respect to the number of steps, we have found that a reasonable choice is one such as the product of the number of steps and the step size is three to five times bigger than the average variable range. Finally, the initialization of a random starting point takes place in line 2, and the procedure of the algorithm begins from line 3. As BP is able to escape local optima the algorithm is not very sensitive on the initial points selected regarding the convergence to a global solution. Nevertheless, a convenient initial solution can still speed up the algorithm, as further discussed in Results.

Trajectory Segment Creation. Each round corresponds to the creation and execution of a trajectory segment, starting from the current point and ending when the maximum number of steps is reached or a collision is detected. At the beginning of each round, the segment's direction is set randomly,[2] in order to ensure the best exploration of the configuration space with only the elevation angle being fixed and following a predetermined schedule. Thus, the step component for each variable is set randomly to a value within [-1, 1]. The step-towards-optimum component for the y axis that respects the elevation angle can be computed as: $y_{step} = \sqrt{\frac{\sin^2 a \sum_{j=1}^{d} x^2_{step,j}}{1-\sin^2 a}}$ where d is the number of variables—i.e., the problem's

[1] The sign of the elevation angle is irrelevant as the square of its sin value is considered.

[2] We experimented with trajectories alternating from one direction to another; however we found this took a toll in exploration. Thus, the algorithm does not behave exactly like a pinball, however, the final trajectories do resemble a pinball movement.

Algorithm 1. Buggy Pinball

```
1: set $stepSize = step_{max}$; $a = a_{min}$; $\#rounds, \#steps$
2: $\boldsymbol{x}, y \leftarrow$ initialize randomly
3: while $i$ in $\#rounds$ do
4:     $\boldsymbol{x}_{step}$ = random(-1,1)
5:     $y_{step} = \sqrt{\frac{\sin^2 a \sum_{j=1}^{d} x_{step,j}^2}{1-\sin^2(a)}}$
6:     $\boldsymbol{x}_{step} = z\boldsymbol{x}_{step}$; $y_{step} = zy_{step}$
7:     while $j$ in $\#steps$ do
8:         if crossing_detected($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}, j$) then
9:             $\boldsymbol{x}, y$ = recursive_refining($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}, j$)
10:            break
11:    $a$ =elevation_cooling($a_{min}, i, \#rounds$)
12:    $stepSize$ =stepSize_cooling($step_{max}, i, \#rounds$)
13: return $\boldsymbol{x}, y$
```

Algorithm 2. crossing_detected ($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}, j$)

```
1: $A = y + (j-1)y_{step} - f(\boldsymbol{x} + (j-1)\boldsymbol{x}_{step})$
2: $B = y + jy_{step} - f(\boldsymbol{x} + j\boldsymbol{x}_{step})$
3: return $(A > 0 \wedge B < 0) \vee (A < 0 \wedge B > 0)$
```

dimensions. It is easy to derive this equation with the following procedure. We know for the elevation angle that: $\sin a = \frac{|\mathbf{nu}|}{|\mathbf{n}||\mathbf{u}|}$, where n is the vector perpendicular to the fundamental plane, i.e. parallel to the y axis $(0, 0, ...0, 1)$. Vector u is the trajectory's segment vector $(x_0, x_1, ..., x_{d-1}, y)$. Values $x_0, x_1, ..., x_{d-1}$ and the elevation angle are known. Solving for y gives us the y_{step} equation above.

With the above procedure, we have set the direction of the segment. What remains is to readjust the dimension-wise step components to respect the predetermined overall step size. In order to achieve this, we multiply all step components with a common factor, z, calculated as $z = \frac{stepSize}{\sqrt{x_0^2+x_1^2...+x_{d-1}^2+y_{step}^2}}$ (used in line 6 of Algorithm 1). Once we have completed this procedure, our step for the current round is ready. We then apply it for the number of steps stated or until a crossing of the objective function is detected.[3]

Stepping Forward. In this part (i.e. Algorithm 1, lines 7-10 and Algorithm 2 and 3), we start our trajectory segment search by applying the step's values on each axis, and we continue until one of the two conditions of stopping is met. As the segment proceeds in the configuration space, it moves downwards concerning the y axis, as the segment value of y decreases in every step. This results in accepting only better values (i.e. closer to the global optimum) of the current position. The crossing of the objective function by the segment is determined by checking the last two steps taken. As shown in the crossing detected function (Algorithm 2), if

[3] Crossing of the objective function means that a common point of the objective function and the current trajectory segment has been detected.

Algorithm 3. recursive_refining ($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}, j$)

```
1: $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{x}_{step} j$; $y = y + y_{step} j$
2: if $y - f(\boldsymbol{x}) \approx 0$ then
3:     return $\boldsymbol{x}, f(\boldsymbol{x})$
4: else
5:     $\boldsymbol{x}_{step} = \frac{\boldsymbol{x}_{step}}{2}$; $y_{step} = \frac{y_{step}}{2}$
6:     $\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_{step}$; $y = y - y_{step}$
7:     if crossing_detected($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}$,0) then
8:         return recursive_refining($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}, 0$)
9:     else
10:        return recursive_refining($\boldsymbol{x}, y, \boldsymbol{x}_{step}, y_{step}, 1$)
```

the difference between the y value and the function evaluation is of different sign between these steps, we know that a point on the objective function is "internal" to the last trajectory segment drawn—and "recursive refining" is triggered.

Recursive Refining. This is a process of iterative refinement, shown also in pseudo-code in Algorithm 3, used to locate the exact point where a trajectory segment crosses the objective function. It is activated only when a crossing of the configuration space is detected, otherwise, that part is skipped. In that case, a loop begins, where the point in the middle between the last two steps is examined, to determine whether this is the common point between the segment and the function (or a very good approximation). If it is not, then we choose whether we continue the loop on the upper or the lower half of the examined part, depending on where the crossing is identified, according to the signs of the points. We continue by examining the point in the middle of that part as before, and the same process is repeated until the exact location of the common point is found. Once we find this point, we stop the iteration of the current round, as we have reached the closest point to the global optimum so far.

Cooling Schedules. The last part takes place at the end of each round (i.e. Algorithm 1, lines 11–12). It determines the values of the desired step size and the angle for the upcoming round. It simply applies the cooling schedule function determined for each of the parameters. In our experiments, we used a simple linear cooling schedule, where the final values are a fraction of 1 for the step size, so we have increased precision no matter the structure of the problem, and from $0.1°$ for highly complex many-local-optima functions up to $89°$ for simple slope no-local-optima functions. This seems to work satisfactorily for each problem tested so far, but further research on the topic is desirable for future work.

3 Experiments

Background and Related Work. As discussed, in our experiments we consider various widely-employed meta-heuristics, both single-point and population-based,

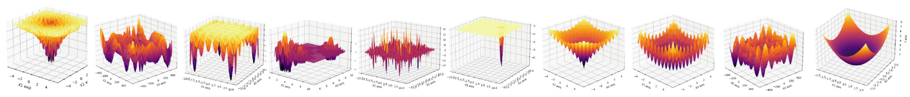
(a) Drop-wave (b) Egg holder (c) Holdertable (d) Langermann (e) Shubert (f) Easom (g) Ackley (h) Rastrigin (i) Schwefel (j) Sphere

Fig. 3. Objective functions' graphs

in order to thoroughly evaluate our approach. Simulated annealing, SA, [15] is a famous single-point, commonly employed meta-heuristic (e.g., [1,16]). Numerous SA variants have also been proposed [23]. Threshold accepting (TA) [7] is proposed as a variant of SA that is also receiving recent attention (e.g., [8,9]). One of the most famous population-based meta-heuristic approaches is particle swarm optimization, PSO [14], initially designed for continuous optimization.

To evaluate our approach, we compare BP against SA, TA, and PSO into the minimization of several benchmark optimization functions that are commonly employed in optimization evaluation [19], and on different dimensions. In more detail, the benchmark functions consider a range that spans from relatively simple uni-modal ones (i.e., Sphere and Easom) to more complex multi-modal ones (i.e., Rastrigin, Ackley, Eggholder, Schwefel, Shubert, Holdertable, Langermann, and Dropwave). As such, all algorithms are evaluated on a diverse range of optimization spaces. The function graphs and equations are reported in Fig. 3 and Table 1 of the Appendix, respectively, for concreteness. All benchmark functions are considered in three dimensions to support straightforward visualization of the results. The benchmark functions, which are also directly defined on a higher number of dimensions (i.e., Sphere, Rastrigin, Ackley, and Schwefel), have also been considered in two, four, five, and six dimensions. This range allowed us to evaluate the scalability of all approaches, while still ensuring fairness and statistical significance. To ensure statistical significance, each algorithm was executed for each function and dimension one hundred times.

In order to ensure a fair comparison, a predefined time allowance was selected and was made available for all approaches, while all experiments were run on the same machine (a 40-CPU Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80 GHz processor, with 64GB RAM). The predefined time allowance was selected to be one second, five seconds, one minute, five minutes and twenty minutes for two, three, four, five and six dimensions respectively. This time allowance was selected before the experiments were executed to avoid favoring any approach.

Ensuring fairness is a particularly prominent challenge when comparing different meta-heuristics, as they operate differently and typically depend on a number of different hyper-parameters [11]. A fair comparison should consider the "same level" of optimization/calibration with respect to the hyper-parameters for all meta-heuristics considered (e.g., the cooling schedule for SA and TA). In order to ensure fairness among all approaches, we performed a thorough exhaustive grid search—for *each algorithm, on every benchmark function, and on every dimension*—to identify the best hyper-parameters for every setting (i.e., the

Table 1. Testbed functions

Function	Equation
Dropwave	$-\frac{1+\cos(12\sqrt{x_1^2+x_2^2})}{0.5(x_1^2+x_2^2)+2}$
Eggholder	$-(x_2+47)\sin(\sqrt{\lvert x_2+\frac{x_1}{2}+47\rvert})$ $-x_1\sin(\sqrt{\lvert x_1-x_2-47\rvert})$
Holdertable	$-\lvert\sin(x_1)\cos(x_2)e^{\lvert 1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\rvert}\rvert$
Langermann*	$\sum_{i=1}^{5} c_i e^{-\frac{1}{\pi}\sum_{j=1}^{d}(x_j-A_{ij})^2}$ $\cos(\pi\sum_{j=1}^{d}(x_j-A_{ij})^2)$
Shubert	$(\sum_{i=1}^{5} i\cos((i+1)x_1+i))$ $(\sum_{i=1}^{5} i\cos((i+1)x_2+i))$
Easom	$-\cos(x_1)\cos(x_2)e^{-(x_1-\pi)^2-(x_2-\pi)^2}$
Ackley	$-20e^{-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}}$ $-e^{\sqrt{\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)}}+20+e^1$
Rastrigin	$10d+\sum_{i=1}^{d}(x_i^2-10\cos(2\pi x_i))$
Schwefel	$418.9829d-\sum_{i=1}^{d}x_i\sin(\sqrt{\lvert x_i\rvert})$
Sphere	$\sum_{i=1}^{d}x_i^2$

*where $c=(1,2,5,2,3)$ and $A^T=\begin{pmatrix}3&5&2&1&7\\5&2&1&4&9\end{pmatrix}$

hyper-parameters that lead to the best performance within the predefined time allowance). The optimal hyper-parameters were used for our evaluations, ensuring a fair comparison among all approaches. The dimension range used in our evaluation enabled us to evaluate the scalability of all approaches while performing this demanding search in feasible time.

When comparing the performance of meta-heuristic approaches, it is crucial to use appropriate metrics. In this work, we use both a precision and accuracy. We calculate the accuracy percentage as the ratio of the trials, where the algorithm converged to an approximation of the global minimum over the total number of trials. We consider an algorithm to have converged to an approximation of the global minimum, if the solution discovered is better than the second-best (local) minimum. To evaluate precision, for those solutions that have converged to an approximation of the global optimum, we calculated the difference between that approximation and the global optimum itself. That is, we calculated the Mean Absolute Error (MAE) as: $MAE=\frac{\sum_{i=1}^{n}\lvert y-x_i\rvert}{n}$—where n is the number of trials, y the global minimum, and x_i the proposed solution on a given trial.

4 Results

Our results are shown in Table 2 for each function, algorithm, and dimension considered, with the best results in each occasion noted in bold. A higher accuracy

Table 2. Evaluation Results

Function		**Accuracy (%)**				**Precision (MAE)**			
		BP	SA	TA	PSO	BP	SA	TA	PSO
Dropwave	3D	**100%**	84%	**100%**	**100%**	1e-4	0.008	0.002	**1e-17**
Eggholder	3D	**100%**	73%	77%	48%	0.872	0.454	4.634	**1e-5**
Holdertable	3D	**100%**	**100%**	**100%**	56%	0.006	0.01	0.008	**1e-6**
Langermann	3D	**100%**	5%	59%	41%	2e-6	0.001	0.007	**2e-15**
Shubert	3D	**100%**	**100%**	**100%**	88%	0.032	0.021	0.274	**8e-6**
Easom	3D	**100%**	**100%**	**100%**	99%	0.002	0.001	7e-4	**5e-17**
Ackley	2D	**100%**	**100%**	**100%**	**100%**	2e-4	0.009	0.002	**4e-16**
	3D	**100%**	**100%**	**100%**	**100%**	7e-5	0.017	0.013	**4e-16**
	4D	**100%**	**100%**	**100%**	**100%**	**8e-5**	0.03	0.026	0.021
	5D	**100%**	**100%**	**100%**	**100%**	9e-5	0.044	0.07	**2e-15**
	6D	**100%**	**100%**	**100%**	**100%**	**3e-4**	0.062	0.115	0.063
Rastrigin	2D	**100%**	**100%**	**100%**	**100%**	7e-7	0.005	5e-4	∼0
	3D	**100%**	**100%**	**100%**	**100%**	8e-7	0.01	0.01	∼0
	4D	**100%**	**100%**	**100%**	99%	6e-7	0.035	0.044	∼0
	5D	**100%**	**100%**	**100%**	82%	7e-7	0.116	0.139	∼0
	6D	**100%**	**100%**	**100%**	46%	7e-7	0.331	0.36	∼0
Schwefel	2D	**100%**	95%	**100%**	95%	2e-4	6e-3	0.11	**1e-5**
	3D	**100%**	90%	93%	91%	3e-4	0.02	0.9	**2e-5**
	4D	**100%**	81%	91%	84%	3e-4	0.142	3.634	**3e-5**
	5D	**100%**	72%	86%	62%	4e-4	0.875	8.69	**3e-5**
	6D	**94%**	71%	77%	60%	3e-4	2.68	16.72	**4e-5**
Sphere	2D	**100%**	**100%**	**100%**	**100%**	0.036	5e-6	2e-12	**1e-117**
	3D	**100%**	**100%**	**100%**	**100%**	0.003	1e-6	2e-7	**4e-16**
	4D	**100%**	**100%**	**100%**	**100%**	8e-4	2e-5	9e-6	∼0
	5D	**100%**	**100%**	**100%**	**100%**	7e-4	1e-4	6e-5	∼0
	6D	**100%**	**100%**	**100%**	**100%**	7e-4	3e-4	2e-4	∼0

percentage indicates better performance (the algorithm discovers and approximates the global optimum more often compared to the rest of the algorithms evaluated), while lower MAE indicates better precision. An MAE of ∼0 is practically zero. Visual inspection indicated that the evaluation results are not normally distributed. A Shapiro Wilk Test with a p-value of 0.05 confirmed that we cannot assume a normal distribution. The statistical significance of all results is tested using a non-parametric Kruskal-Wallis H test and follow-up Conover's tests (along with the step-down method using Bonferroni adjustment for p-value adjustment). A p-value threshold of 0.05 is used for statistical significance. All statistical significance results are included in the Appendix.

As seen in Table 2, BP shows higher or equal accuracy rates compared to all other algorithms in all settings. The starting point for each trial and each algorithm is random. BP has no trouble approximating the global minimum from any possible starting position. Notably, it is the algorithm with the most times to achieve a 100% accuracy ratio. There are some cases, however, where all algorithms achieve almost 100% accuracy. This occurs for the "less complex" functions Ackley, Easom, Rastrigin, Schwefel (2D), Sphere, Holdertable, Shubert, and Dropwave, where all or almost all algorithms exhibit almost 100% accuracy (we elaborate below). The more "complex" functions are Schwefel (3D, 4D, 5D, 6D), Eggholder, Langermann. In those, BP always outperforms its competitors. BP reaches 100% accuracy in all these results, and its superiority to others is statistically significant, except results against TA for the Schwefel 3D, 4D cases.

Now, regarding precision, we clarify that it is calculated only for the points that have converged to an approximation of the global minimum. Thus, high precision demonstrates how well the global optimum is approximated, if the algorithm did not get stuck to a local optimum in the first place. As such, a *high precision-low accuracy* performance is *not* suitable for global optimization—since, although precise, the algorithm is not discovering the global optima often enough. That said, an adequate performance concerning precision is definitely required for global optimization algorithms. As can be seen, BP's precision is high; and it is higher than that of the other two single-point algorithms considered (SA, TA) in most cases. Follow-up tests confirm statistically significant better BP precision against SA and TA in all cases. The precision of BP seems to be lower than that of PSO (which is also developed for continuous optimization tasks), and follow-up tests indicate that this difference is statistically significant. Notably, however, PSO frequently has the worst accuracy among all algorithms, and thus is a poor choice for global optimization in these cases.

As noted already, the advantage of BP becomes greater when the more complex functions are considered, i.e., Eggholder, Schwefel and Langermann, and when we move to higher dimensions. These functions have many and deep local minima, but only a single global one. The Eggholder and Langermann are also non-symmetric. These facts make them harder to optimize in a global optimization manner, while not getting stuck in a local minimum. The higher dimensionality introduces further challenges for global optimization. That said, BP manages to achieve a 100% accuracy in all occasions except Schwefel for 6D (where an 94% is achieved). The accuracy results are also always better compared to the rest of the algorithms considered and the improvement ranges to up to 20 times better (i.e., compared to SA for Langermann 3D). When simpler functions and lower dimensions are considered the differences between the algorithms become less prominent. For instance, the Dropwave, Ackley, and Rastrigin have few local minima that are relatively shallow, while Shubert and Holdertable have many global optima. As such, most algorithms reach 100% accuracy except

SA in Dropwave 3D, PSO in Rastrigin 4D, 5D and 6D, and PSO in Holdertable 3D and Shubert 3D. Finally, when the unimodal functions, Sphere and Easom are considered, not surprisingly, all algorithms achieve a 100% accuracy.

5 Conclusions and Future Work

In this paper, we introduced a fundamentally novel single-point meta-heuristic tailored for global continuous optimization problems. Our algorithm, *buggy pinball*, is inspired by the pinball arcade game and is able to discover the global optimum in an any-time optimization manner. We evaluated our algorithm against widely-employed meta-heuristics on standard test-beds. We showed that it has a better performance compared to all benchmark approaches, especially when complex optimization functions and multiple dimensions are considered.

Future and ongoing work includes various extensions of the Buggy Pinball algorithm. For instance, investigating different cooling schedules for the step length and elevation angle parameters could improve the algorithm's effectiveness. Another valuable extension could be adapting BP for discrete optimization problems, to benefit from the perquisites of the algorithm in that domain as well.

References

1. Abdel-Basset, M., Ding, W., El-Shahat, D.: A hybrid harris hawks optimization algorithm with simulated annealing for feature selection. Artif. Intell. Rev. **54**(1), 593–637 (2021)
2. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Global Optim. **31**(4), 635–672 (2005)
3. Biehl, M., Schwarze, H.: Learning by on-line gradient descent. J. Phys. A: Math. Gen. **28**(3), 643 (1995)
4. Chambolle, A., Pock, T.: An introduction to continuous optimization for imaging. Acta Numer **25**, 161–319 (2016)
5. Dhouib, S., Kharrat, A., Chabchoub, H.: A multi-start threshold accepting algorithm for multiple objective continuous optimization problems. Int. J. Numer. Meth. Eng. **83**(11), 1498–1517 (2010)
6. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(7) (2011)
7. Dueck, G., Scheuer, T.: Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. J. Comput. Phys. **90**(1), 161–175 (1990)
8. Frausto-Solis, J., Hernández-Ramírez, L., Castilla-Valdez, G., González-Barbosa, J.J., Sánchez-Hernández, J.P.: Chaotic multi-objective simulated annealing and threshold accepting for job shop scheduling problem. Math. Comput. Appli. **26**(1), 8 (2021)
9. Geiger, M.J.: Pace solver description: A simplified threshold accepting approach for the cluster editing problem. In: 16th International Symposium on Parameterized and Exact Computation (IPEC 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)

10. Grass, J., Zilberstein, S.: Anytime algorithm development tools. ACM SIGART Bulletin **7**(2), 20–27 (1996)
11. Halim, A.H., Ismail, I., Das, S.: Performance assessment of the metaheuristic optimization algorithms: an exhaustive review. Artif. Intell. Rev. **54**(3), 2323–2409 (2021)
12. Hochreiter, S., Younger, A.S., Conwell, P.R.: Learning to learn using gradient descent. In: Dorffner, G., Bischof, H., Hornik, K. (eds.) ICANN 2001. LNCS, vol. 2130, pp. 87–94. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44668-0_13
13. Jeyakumar, V., Rubinov, A.M.: Continuous Optimization: Current Trends and Modern Applications, vol. 99. Springer Science & Business Media (2006). https://doi.org/10.1007/b137941
14. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN 1995-International Conference On Neural Networks, vol. 4, pp. 1942–1948. IEEE (1995)
15. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
16. Lin, S.W., Cheng, C.Y., Pourhejazy, P., Ying, K.C.: Multi-temperature simulated annealing for optimizing mixed-blocking permutation flowshop scheduling problems. Expert Syst. Appl. **165**, 113837 (2021)
17. Mirjalili, S., Lewis, A.: The whale optimization algorithm. Adv. Eng. Softw. **95**, 51–67 (2016)
18. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv. Eng. Softw. **69**, 46–61 (2014)
19. Molga, M., Smutnicki, C.: Test functions for optimization needs. Test Funct. Optim. Needs **101**, 48 (2005)
20. Munoz, M.A., Kirley, M., Halgamuge, S.K.: The algorithm selection problem on the continuous optimization domain. In: Computational Intelligence In Intelligent Data Analysis, pp. 75–89. Springer (2013). https://doi.org/10.1007/978-3-642-32378-2_6
21. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural Netw. **12**(1), 145–151 (1999)
22. Shalev-Shwartz, S., Ben-David, S.: Understanding machine learning: From theory to algorithms. Cambridge University Press (2014)
23. Siddique, N., Adeli, H.: Simulated annealing, its variants and engineering applications. Int. J. Artif. Intell. Tools **25**(06), 1630001 (2016)
24. Taylan, P., Weber, G.W., Yerlikaya, F.: Continuous optimization applied in mars for modern applications in finance, science and technology. In: ISI Proceedings of 20th Mini-euro Conference Continuous Optimization and Knowledge-based Technologies, pp. 317–322. Citeseer (2008)
25. Vanderbilt, D., Louie, S.G.: A monte carlo simulated annealing approach to optimization over continuous variables. J. Comput. Phys. **56**(2), 259–271 (1984)
26. Weber, G.W., Özöğür-Akyüz, S., Kropat, E.: A review on data mining and continuous optimization applications in computational biology and medicine. Birth Defects Res. C Embryo Today **87**(2), 165–181 (2009)
27. Xiong, Q., Jutan, A.: Continuous optimization using a dynamic simplex method. Chem. Eng. Sci. **58**(16), 3817–3828 (2003)