Gradient Guided Differential Evolution-Simulated Annealing Hybrid For Multivariate Differentiable Functions

Bhumrapee Soonjun[a]

[a]*Pusheen Co, Pusheen Road, Pusheen City, 1234, Pusheen, Pusheen*

Abstract

In this term paper, a new metaheuristic hybrid between Differential Evolution and Simulated Annealing algorithm with gradient exploitation for continuous functions is developed and tested against other well-named heuristics. The proposed hybrid algorithm uses Differential Evolution as the "master" program where the main loop resides, while Simulated Annealing's Boltzmann acceptance criteria are used to improve search space exploration and population diversity. Since the nature of our target functions is differentiable, gradient methods and local search techniques are utilized to exploit the nature of the curve to further accelerate convergence speed and the solution quality. The proposed algorithm is compared to metaheuristics including Gradient Evolution, Differential Evolution, Dual Annealing, Artificial Bee Colony, and Particle Swarm Optimization, where it demonstrates promising performance that outperforms other algorithms in terms of solution quality.

Keywords: Pusheen, Cat, Cute

1. Introduction

1.1. Differential Evolution

Differential Evolution (DE), pioneered by Rainer Storn and Kenneth Price, is a population-based stochastic optimization algorithm that belongs to the Evolutionary Algorithms class. Like the Genetic Algorithm (GA), it maintains a population of solutions and gradually increases the population fitness through similar operations called crossover and mutation (Storn and Price (1997)). However, unlike GA where the probability of solution perturbation is predefined by some probability distribution DE performs perturbation on every individual. Moreover, in GA each individual in the population is represented by chromosomes comprising genes which are usually represented in the code by string / Iterable datatype, the population maintained by DE are vectors of real numbers.

With this, Differential Evolution already presents a huge advantage in terms of computation time when compared to GA for optimizing in the real domain because each individual inside DE is kept as a primitive datatype - float / double. In addition, it still enjoys similar benefits that Genetic Algorithm including but not limited to global optimization, navigation in complex search space, and immense easiness in parallelization. However, it also shares some disadvantages with GA such as premature convergence.

1.2. Simulated Annealing

Unlike Differential Evolution, Simulated Annealing (SA) belongs to a class of local search. However, unlike greedy hill climbing where escape from local optima is impossible, it simulates the cooling process of a metal by allowing the algorithm to explore different regions of the solution space in the beginning before setting down on one local neighborhood. Through this process, it is possible that SA will jump off bad local optima to a better one. Internally, to mimic this process, the algorithm maintains the temperature which defines how large is the neighborhood we are going to search.

Although it possesses the ability to jump off the local optima to search more areas, as the name suggests, it still belongs to the class of local search heuristics meaning that it might not converge to the true global optimal solution if the search space is huge. Some of the disadvantages of classical SA include difficulty in parameter tuning - alpha, beta, and temperature, being stuck in local optima, and slower convergence for small alpha Odeniyi et al. (2015).

1.3. The best of both worlds

It is then natural to think of combining the Differential Evolution and Simulated Annealing as they complement each other shortcomings. In the following term project, Differential Evolution will be used as a master program to run the optimization task, and ideas from simulated annealing will be used to conduct population selection in a stochastic way to maintain a variety of solutions; hence, avoiding being stuck in the local optima. By hybridizing the two, the unified algorithm will enjoy both the benefit of global optimization from DE and exploration of search space to prevent premature convergence by SA.

1.4. Traditional Mixins

To speed up local convergence and solution quality, traditional gradient-based methods such as Newton-Raphson, Gradient Descent, Stochastic Gradient Descent, and BFGS will be introduced as another operation to quickly reduce the solution

to the local optimal solution. Furthermore, since these methods are exact - apart from SGD, they can generate a better local optimal solution than the DE and SA if sufficient conditions are met.

The main goal of this term project, then, is to explore the possibility of the new hybrid outlined above against classical heuristics, Continuous Genetic Algorithm, Differential Evolution, and Gradient Evolution in terms of speed of convergence, number of evaluation function invocations, quality of the solution, and robustness of the algorithm.

2. Review of relevant literature

2.1. Storn and Price (1997): Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces

In this original paper, Storn and Price proposed a new heuristic named Differential Evolution that belongs to the Evolutionary Class algorithm. However, unlike GA, mutation - or perturbation in their own term - is performed as the first and then followed by crossover. To perform a mutation, the authors proposed the following way,

1. Let $x_{i,n}$ be the individual i in the generation n, then
2. Choose $x_{i,n}, _{j,n}, _{k,n} \in G_n$ such that $i \neq j \neq k$
3. For some $F \in (0, 2]$, $v_{i,n+1} = x_{i,n} + F \cdot (x_{j,n} - x_{k,n})$

This is performed for every vector in the population. In addition to the mutation, a crossover operation is also applied with some rate CR through the following method

$$u_{i,n+1} = \left\{ \begin{array}{ll} v_{i,n+1}, & randb(j) \leq CR \parallel j = rnbr(i) \\ x_{i,n}, & randb(j) > CR \parallel j \neq rnbr(i) \end{array} \right\} \quad (1)$$

where the vector $u_{i,n+1}$ is called the trial vector. The process can be defined as the following notation,

$$DE/x/y/z$$

Where x is the vector to be mutated which can be rand, best, etc, and y is the number of difference vectors to be used, and z is the criteria - binomial. One such setting they recommended to improve population diversity is,

$$DE/best/2/bin$$

$$u_{i,n+1} = x_{best,n} + F \cdot (x_{j,n} + x_{l,n} - x_{k,n} - x_{m,n}) \quad (2)$$

Lastly, the population selection criteria is greedy which is like the elitist GA. According to them, the DE outperforms all other kinds of Evolutionary algorithms in the First International IEEE Competition on Evolutionary Optimization. However, it placed third after two deterministic methods of "limited application" (Storn and Price (1997)).

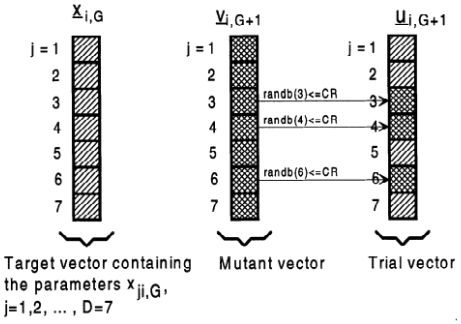


Figure 1: Crossover Illustration from Storn and Price (1997)

2.2. Rafajłowicz (2015): A Hybrid Differential Evolution-Gradient Optimization Method

In the following paper, Rafajłowicz describes a hybrid algorithm between Differential Evolution and the traditional gradient method by using the Newton-Raphson second-order method to update the trial solution obtained in equation 21 through

$$H_f(u_{i,n+1})p = -\nabla_f(u_{i,n+1}) \quad (3)$$

$$u_{i,n+1} = u_{i,n+1} + p \quad (4)$$

Where H is the hessian matrix. In addition, the article also describes the possibility of improvement by adjusting the step size F in the mutation step in each iteration according to the following equation

$$F = \eta + \theta e^{-i/n} \quad (5)$$

where η and θ are parameters, i is the current iteration and is the total number of iterations. They claimed from their results that the new Gradient Based DE outperformed traditional DE convincingly in their test where one of them took only 50 iterations to converge compared to 180 by the original DE, while their DE, SA-inspired took 60 iterations to converge to even a better solution than Gradient DE and DE.

2.3. Kuo and Zulvia (2015): Gradient Evolution

Gradient Evolution (GE) is a metaheuristic that belongs to the evolutionary class algorithm. Instead of computing the gradient using finite difference, it approximates the search (gradient) direction by considering the neighboring solution instead. The update rule used in GE is derived from Taylor expansion on the derivative for first and second order which is given in equation 6 and 7.

$$f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \quad (6)$$

$$f''(x) \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{(\Delta x)^2} \quad (7)$$

Replacing this approximation in Newton's method yields the approximate update rule given in equation 8.

$$x_{t+1} = x_t - \frac{\Delta x_t}{2} \cdot \frac{f(x_t + \Delta x) - f(x_t - \Delta x)}{f(x_t + \Delta x) - 2f(x_t) + f(x_t - \Delta x)} \tag{8}$$

Since computing the actual gradient through numerical differentiation is computationally expensive because it requires multiple objective function evaluations, the author derived a population-based method that uses each vector instead of computing the fitness value. Assume without loss of generality that the objective is minimization, then x_t has two neighbors $x_t - \Delta$ and $x_t + \Delta x$ such that $f(x_t - \Delta x) < f(x_t) < f(x_t + \Delta x)$. Similarly, observe that a vector x_t in the population will have at least two neighbors y_t and z_t such that $f(y_t) \leq f(x_t) \leq f(z_t)$ if x_t is not the best or worst vector in the population. So as an approximation, we replace $f(x_t + \Delta x_t)$ and $f(x_t - \Delta x_t)$ with $f(y_t)$ and $f(z_t)$ instead. The authors further suggest using the vector itself instead of the fitness value because the fitness function can be expensive and their preliminary experiments showed that using the fitness value doesn't significantly affect the quality of the solution. As a result, the vector update rule for GE becomes,

$$x_t = r_g \cdot \frac{|x_t - y_t| + |z_t - x_t|}{4} \cdot \frac{z_t - y_t}{z_t - 2x_t + y_t} + r_a(b_t - x_t) \tag{9}$$

where $r_g, r_a \sim \mathcal{N}(0, 1)$ and b_t is the best vector in the population. In each generation, the above update rule is applied to all vectors in the population to drive the population toward the minimum. Apart from the update rule, the algorithm also comprises of other steps to ensure that the population does not get stuck in the local minimum through vector refreshing and vector jumping in equation 10.

$$x_t = -x_t + \mathcal{N}(0, 1) \cdot (x_t + y_t) \tag{10}$$

This can, however, generate accidental result if the function has minimum about the origin because the difference between two quantity can be near zero which by chance improve the solution. Because of this, as shown in our preliminary experiments, if the function is shifted from the origin, the GE doesn't perform as well.

2.4. Quasi-Newton's Method

Newton's method is a well-known numerical approximation method for finding the root of an equation where the update rule is given by,

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \tag{11}$$

for one dimensional case. To adapt the Newton method to find a local minimum we can simply take the derivative of $f(x)$ to get $f'(x)$ and use Newton on it instead. In total this becomes,

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} \tag{12}$$

In multidimensional space, the rule is generalized to,

$$\hat{x}_{k+1} = \hat{x}_t - \left(\nabla^2 f(\hat{x}_t)\right)^{-1} \nabla f(\hat{x}_t) \tag{13}$$

or in another form,

$$\hat{x}_{k+1} = \hat{x}_t - \left(H_f(\hat{x}_t)\right)^{-1} \nabla f(\hat{x}_t) \tag{14}$$

where H is the Hessian matrix of f. Since computing the inverse of the Hessian is computationally expensive, the Quasi-Newton methods are developed. In general, it is of the following form,

$$\hat{x_{t+1}} = \hat{x}_t - B_k \nabla f(\hat{t}) \tag{15}$$

where B_k is the approximation of the Hessian, different quasi-Newton methods differ in the construction of B_k (Virk (2023)).

2.5. Rodomanov and Nesterov (2022) Rates of superlinear convergence for classical quasi-Newton methods

Rodomanov and Nesterov (2022) showed that the default Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm has a superlinear convergence rate. Let k be the iteration counter, n be the dimension, μ be the strong convexity parameter, and L be the Lipschitz constant, then the convergence rate of BFGS is

$$\left(\frac{nL}{\mu^2 k}\right)^{k/2} \tag{16}$$

while Newton-Raphson is known to possess a quadratic convergence rate. Although BFGS is worse in terms of convergence rate, its computational complexity is only $O(d^2)$ since it doesn't require matrix inversion compared to $O(d^3)$ in the Newton-Raphson method.

2.6. Tibshirani (2013) Gradient Descent and Stochastic Gradient Descent

It was shown by Tibshirani (2013) that gradient descent under convexity satisfies,

$$f(x^{(k)}) - f(x^*) = O\left(\frac{1}{\sqrt{k}}\right) \tag{17}$$

Moreover, if f is differentiable with Lipshcitz continuity, for some step sizes we get that

$$f(x^{(k)}) - f(x^*) = O\left(\frac{1}{k}\right) \tag{18}$$

and under strong convexity with Lipshcitz gradient the convergence rate for GD is,

$$f(x^{(k)}) - f(x^*) = O\left(c^k\right) \tag{19}$$

where $c < 1$, which is linear. For Stochastic Gradient Descent, similar results follow for the convex function, however, when under strong convexity with Lipschitz gradient, SGD yields at best,

$$\mathbf{E}[f(x^{(k)})] - f(x^*) = O\left(\frac{1}{k}\right) \tag{20}$$

which is sub-linear. Lastly, both classical GD and SGD have the computation complexity of $O(knd)$ where k is the number of iterations, n is the number of samples and d is the number of dimensions. However, since we are concerned about a singular point, this reduces to $O(kd)$.

2.7. Atangana (2018): Latin Hypercube Sampling

The Lain Hypercube Sampling is a sampling technique used to initialize the initial population. The sampling algorithm divides the interval for each dimension into non-overlapping N equal partitions with $1/N$ probability of a partition being selected. The algorithm sampled N values from each dimension and the N value from each dimension are paired randomly to form an N-Tuplets. The generated points are in the range $[0, 1)$ for all dimensions. For our use case, this is extended using the provided bounds.

3. Algorithm Design

This section introduces and discusses the design that goes into the algorithm including the population data structure, mutation strategies, gradient methods utilization, and the pseudocode. In order to facilitate this, the following notations are used.

- $f(x)$: Function to be minimized
- N: Number of generations
- P: Population
- P_i: Individual ith
- u_i: Trial vector ith
- $|P|/NP$: Population size
- p_c: Crossover probability
- $GM(x)$: Gradient Method
- p_g: Gradient probability
- F: Differential Factor
- T: Temperature
- α: Cooling rate
- β: α update rate
- E_r: Elitism ratio

3.1. Population Data Structure

To cater to elitism while also maintaining a certain level of freedom of search space exploration, we will maintain a single population that is divided into two parts depending on the elitism ratio E_r. The first part $P[: \lceil E_r \cdot |P| \rceil]$ will be selected in an elitist fashion, where the latter part of the population selection scheme is Boltzmann which chooses to accept a solution with some probability. Concretely, let $\Delta = f(u_i) - f(P_i)$,

$$p = \left\{ \begin{array}{ll} 1, & \text{if } \Delta < 0 \\ e^{-(\Delta)/T}, & \text{otherwise} \end{array} \right\} \tag{21}$$

3.2. Offspring Creation

Here we discuss two methods to create offspring in each generation.

3.2.1. Mutation and Crossover

The first way to create offspring is the normal Differential Evolution style vector mutation method. Specifically, we will use the $DE/rand/1/bin$ scheme. We want to avoid $DE/best/1/bin$ because it can quickly trap the vectors in local minima. Even worse, if the diversity of the vectors is too little, the algorithm cannot proceed because the difference is near zero. Other schemes can also be utilized, but we found this to be the best. Also possible is to swap between $DE/rand/2/bin$ to improve diversity, however, it is not included in this study.

3.2.2. Using Gradient Methods

.

In addition to standard mutation and crossover, we will also introduce another way to produce offspring by applying gradient methods to the target vector. In the article by Rafajłowicz (2015), the author uses the classical Newton method to locally optimize each obtained solution. However, Newton's method is known to be computationally expensive, especially for larger dimensions which is in the order of d^3 because it requires hessian inversion. The results obtained by the author are indifferent because the benchmark function used only has two variables. To overcome the curse of dimensionality, we turn to comparable alternatives such as but not limited to BFGS, L-BFGS, Gradient Descent (GD), Stochastic Gradient Descent (SGD), Nealder-Mead, and Conjugate Gradient.

BFGS, on the surface, has an advantage over GD and SGD in terms of convergence rate. In fact, it possesses a superlinear convergence rate if the underlying function is Lipschitz continuous and strongly convex, both of which are sufficient conditions needed to guarantee convergence. Outside the theory land, BFGS is also shown to perform well in practice even when applied to nonconvex functions and seldom fail (Dai (2002)). Furthermore, its variant, the L-BFGS, also showed promising performance and robustness in optimizing the Deep Reinforcement Learning Model (Rafati and Marcia (2019)) and in other areas.

Although it alleviates the curse of dimensionality to some extent, the algorithm still runs in $O(d^2)$ which is still expensive

if applied in every iteration. To fix this issue, one can buy more GPUs or use alternatives such as Gradient Descent or Stochastic Gradient Descent which runs in $O(d)$ time since it only needs to calculate the gradient at a point.

Under theory, both GD and SGD are superseded by BFGS in terms of convergence rate, where both of them even under strong convexity and Lipschitz gradient are at best linear for GD and sub-linear for SGD (Tibshirani (2013)). In addition, when near local minima, both GD and SGD suffer from a decreasing rate of convergence or overshooting the solution which can lead to huge tolls in performance. However, its computational complexity is exponentially better and has been shown in practice to be very effective at optimizing large ML models with a huge number of parameters.

In our case, although the underlying function does not necessarily need to fit the above convexity assumptions on the global scale, it is possible that in some local neighborhoods where gradient methods are employed, it behaves reasonably enough like a convex function. For instance, suppose that in some local neighborhood, the Hessian is positive definite, then Newton's method can be applied to attain the isolated local minimum. Our choice for the gradient method is BFGS for every experiment presented in this study.

3.3. Proposed Algorithm

The pseudocode of the algorithm is given in Algorithm 1. Notice that when the SA parameters are set to 0 - i.e. we take $T = 0$ or $E_r = 1$, and $p_c = 0$ - the algorithm effectively reverts back to an elitist scheme which is simply Differential Evolution.

Algorithm 1: Gradient DE-SA Hybrid

input : f, N, F, p_c, p_g, GM, E_r, T, α, β
output: P_n: Last Generation Population
P = Initialize Population;
for *i from 1 up to N* **do**
  if *i mod* $\beta == 0$ **then**
    $T = T \cdot \alpha$
  end
  for *j from 1 up to* $|P|$ **do**
    if *Unif*$(0, 1) < p_c$ **then**
      $u_i = GM(P_i)$;
    else
      u_i = Mutate(u_i, F, j);
      u_i = Crossover(u_i, p_c);
    end
  end
  $\Delta = f(u_i) - f(P_i)$;
  if $\Delta < 0$ **then**
    $P_i = u_i$;
  else if *Unif(0, 1)* $< \exp(-\Delta/T)$ *and* $i < E_r \cdot |P|$ **then**
    $P_i = u_i$;
  end
end

4. Experiment and Data Collection

The Differential Evolution-Simulated Annealing hybrid is implemented in Python using NumPy's high-performance array and SciPy's optimization module for gradient methods. The algorithm tested against five other metaheuristics algorithms, namely, Gradient Evolution (GE), Differential Evolution (DE), Dual Annealing (DA), Artificial Bee Colony (ABC), and Particle Swarm Optimization Algorithm (PSO), all of which belongs to the global optimization class. Gradient Evolution and Differential Evolution are implemented in Python according to Kuo and Zulvia (2015) and Storn and Price (1997) respectively, while Dual Annealing is from scipy's package (Virtanen et al. (2020)), ABC is from bee colony package (Oliveira (2021)), and PSO is from pymoo (Blank and Deb (2020)) Packages. For our algorithm, the code is at https://gitlab.com/BhumBhumrapee/heuristic-optimization.git. Lastly, all of the tests are run sequentially on Apple's Macbook M2 Pro Max, with 12 CPU cores and 32 GB of RAM. Lastly, in all of the algorithms, the initialization criteria are the Latin Hypercube Sampling (LHS).

4.1. Test functions

The test bed used to test the DE-SA hybrid and other heuristics comprises several functions with varying underlying natures chosen from Surjanovic and Bingham and Molga and Smutnicki (2005). The complete list along with the formula and bounds for each test function are given in Table 1.

The Sphere, Weighted Sphere, Rotated Hyper Ellipsoid, and Schwefel 1.2 are convex functions; however, the Schwefel 1.2 has a large flat area about the minimum making it hard to search for the minima. Schwefel 2.3, Griewank, Ackley, Rastrigin, and De Jong 5 are highly multi-modal functions. De Jong 3 and Easom are non-convex with one true minima but the search space is relatively flat with steep ridges for global minimum where gradient information is not useful. The minimum is at $f(\mathbf{0}) = 0$ for every test apart from Schwefel 2.3, Rosenbrock, Easom, De Jong 5, and De Jong 3 where the minimum of each function is in Table 2.

For plotting convergence graphs purposes, we have modified the Easom and De Jong 3's functions to have the minimum value of 0 instead. Lastly, the number of dimensions for each test function is chosen to be 30 apart from Easom and De Jong 5 where the number of dimensions is only two by default.

4.2. Parameters Tuning

The parameters for GE, DE, PSO are tuned according to Kuo and Zulvia (2015), Storn and Price (1997), and Shi and Eberhart (1999), respectively. The rest are the default parameters chosen by each respective package. In addition, the population size for GE, DE, ABC, and PSO is chosen to be 40, and the maximum number of generations is set to be 1000. For PSO, the number of particles is chosen to be 100. Note that since DA is not a population-based method, we do not need to set the population size.

The parameters for DE-SA are tuned in the following steps. First, we tune p_g, F, and p_c which are the default DE parameters

Table 1: Test Functions

Name	Formula	Bounds
Sphere	$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	$-100 \leq x_i \leq 100$
Weighted Sphere	$f(\mathbf{x}) = \sum_{i=1}^{D} i \cdot x_i^2$	$-100 \leq x_i \leq 100$
Schwefel 1.2	$f(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j}^{i} x_i\right)^2$	$-500 \leq x_i \leq 500$
Schwefel 2.3	$f(\mathbf{x}) = 418.9892 \cdot D + \sum_{i=1}^{D} -x_i \cdot \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$-500 \leq x_i \leq 500$
Easom	$f(\mathbf{x}) = -(-1)^D \left(\prod_{i=1}^{D} \cos^2(x_i)\right) \cdot \exp\left(-\sum_{i=1}^{D}(x_i - \pi)^2\right)$	$-100 \leq x_i \leq 100$
Rotated Hyper Ellipsoid	$f(\mathbf{x}) = \sum_{i=1}^{D} \sum_{j=1}^{i} x_i^2$	$-65.536 \leq x_i \leq 65.536$
Rosenbrock	$f(\mathbf{x}) = \sum_{i=1}^{D-1} \left((x_i)^2 + 100(x_{i+1} - x_i^2)^2\right)$	$-2.048 \leq x \leq 2.048$
Griewank	$f(\mathbf{x}) = \frac{1}{4000} \cdot \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$
Sum of Different Powers	$f(\mathbf{x}) = \sum_{i=1}^{D} \lvert x_i \rvert^{i+1}$	$-1 \leq x_i \leq 1$
Ackley	$f(\mathbf{x}) = -20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D} \cdot \sum_{i=1}^{D} \cos(2\pi \cdot x_i)\right) + 20 + e$	$32.768 \leq x_i \leq 32.768$
Rastrigin	$f(\mathbf{x}) = \sum_{i=1}^{D} \left(x_i^2 - 10 \cdot \cos(2\pi x_i) + 10\right)$	$-600 \leq x_i \leq 600$
De Jong 5	$f(\mathbf{x}) = \left(0.002 + \sum_{i=1}^{25} \left(i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6\right)^{-1}\right)^{-1}$, where $\mathbf{a} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$	$-65.536 \leq x_i \leq 65.636$
De Jong 3	$f(\mathbf{x}) = \sum_{i=1}^{D} \lfloor x_i \rfloor$	$-3.5 \leq x_i \leq 3.8$

Table 2: Minimum of functions that is not at the origin

Function	Minimum
Schewefel 2.3	$f(420.9687, ..., 420.9687) = 3.82\text{e-}04$
Rosenbrock	$f(1, ..., 1) = 0$
Easom	$f(\pi, \pi) = -1$
De Jong 5	$f(-31.978334., -31.978334.) = 0.99800.$
De Jong 3	$f([-3.5, ..., -3.5] \times [-3.0, ..., -3.00]) = -4D$

with gradient probability. After we obtain a reasonably good enough solution, we tune T, α, β, and E_r next where T and α are adjusted first to get slower convergence but decrease enough to locally search the space later on. β is not really sensitive in most cases. E_r is recommended to be set to 0.8 to get good convergence speed.

For our algorithm, the chosen parameters are presented in Table 3. For Sphere, Weighted Sphere, Easom, Rotated Hyper Ellipsoid, Power Sum, and De Jong 3's functions, the functions are simple enough that the SA aspect does not need to be used. Therefore, T and E_r is set to 0 and 1 respectively. The rest - Schwefel 1.2, Schwefel 2.3, Rosenbrock, Griewank, Ackley, Rastigin's, and De Jong 5's functions - are either highly multi-modal or has relatively flat surface about the minimum. Therefore, accepting worse moves can increase the chance of jumping off the minima or increasing diversity which forces the process from effectively becoming stagnant.

Table 3: Parameters Tuning for DE-SA

	p_g	F	p_c	T	α	β	E_r
Sphere	0.01	0.5	0.6	0	0	0	1
Weighted Sphere	0.00	0.5	0.6	0	0	0	1
Schwefel 1.2	0.01	0.5	0.8	10	0.9	5	0.6
Schwefel 2.3	0.05	1	0.1	100	0.9	10	0.5
Easom	0.0	0.5	0.9	0	0	0	1
Rotated Hyper Ellipsoid	0.01	0.5	0.7	0	0	0	1
Rosenbrock	0.01	0.5	0.8	1000	0.95	10	0.6
Griewank	0.01	0.5	0.2	100	0.95	10	0.5
Power Sum	0.001	0.5	0.8	0	0	0	1
Ackley	0.01	0.5	0.1	1000	0.95	10	0.8
Rastrigin	0.01	0.5	0	100	0.95	10	0.8
De Jong 5	0	0.5	0	100	0.95	10	0.5
De Jong 3	0	0.5	0.1	0	0	0	1

Table 4: Terminating values from each algorithm. Bold entries are the minimum along the row.

Test Function Name	DE-SA	GE	DE	DA	ABC	PSO
Sphere	**1.87e-62 (5.43e-62)**	2.03e-17 (8.06e-18)	5.69e-20 (8.04e-20)	4.65e-13 (6.56e-13)	3.14e-11 (4.13e-11)	1.95e-05 (1.03e-04)
Weighted Sphere	**1.58e-59 (4.12e-59)**	1.03e+01 (4.21e+00)*	5.51e-19 (5.80e-19)	1.86e-09 (2.39e-09)	3.27e-10 (2.96e-10)	3.24e+00 (1.70e+01)*
Schwefel 1.2	**4.84e-17 (1.74e-16)**	7.93e+02 (3.76e+02)*	3.11e+03 (7.94e+02)*	2.02e-08 (1.35e-08)	3.23e+05 (6.35e+04)*	8.00e+03 (4.57e+03)*
Schwefel 2.3	**3.82e-04 (9.77e-12)**	1.07e+04 (3.57e+01)*	4.51e+05 (6.00e+04)*	3.82e-04 (1.60e-07)	3.37e+02 (1.57e+02)*	3.53e+03 (5.01e+02)*
Rotated Hyper Ellipsoid	**6.46e-58 (1.92e-57)**	2.60e-17 (1.34e-17)	1.51e-19 (1.03e-19)	1.29e-09 (1.06e-09)	1.86e-10 (1.72e-10)	1.60e-02 (8.61e-02)
Rosenbrock	**6.34e-15 (1.42e-14)**	2.85e+01 (1.46e-01)*	2.46e+01 (1.91e+00)*	1.33e-01 (7.16e-01)	1.79e+01 (6.20e+00)*	3.20e+01 (1.77e+01)*
Sum of Different Powers	**1.05e-76 (2.67e-76)**	2.39e-17 (3.72e-17)	5.89e-26 (2.71e-25)	3.84e-06 (3.76e-06)	4.01e-11 (7.16e-11)	5.27e-11 (2.61e-10)
Easom	**0.00e+00 (0.00e+00)**	2.13e-05 (2.53e-05)*	**0.00e+00 (0.00e+00)**	5.00e-01 (5.00e-01)	1.14e-02 (2.34e-02)*	**0.00e+00 (0.00e+00)**
Ackley	**1.50e-14 (1.41e-15)**	3.09e-11 (1.07e-10)	6.10e-11 (2.33e-11)	1.96e-08 (4.09e-10)	1.57e-05 (9.05e-06)*	1.60e-01 (3.53e-01)*
Rastrigin	**7.20e-14 (2.51e-14)**	1.14e-11 (3.75e-11)	4.34e-06 (2.58e-06)*	1.33e-01 (3.38e-01)	6.71e-01 (8.62e-01)*	5.91e+01 (3.85e+01)*
Griewank	3.33e-17 (9.13e-17)	**0.00e+00 (0.00e+00)**	2.71e-03 (5.48e-03)	3.29e-04 (1.77e-03)	4.17e-04 (2.24e-03)	6.18e-02 (1.98e-01)
De Jong 5	**9.98e-01 (1.52e-16)**	**9.98e-01 (1.18e-08)**	1.03e+00 (1.78e-01)	**9.98e-01 (1.56e-10)**	**9.98e-01 (1.40e-15)**	**9.98e-01 (1.28e-16)**
De Jong 3	**0.00e+00 (0.00e+00)**	**0.00e+00 (0.00e+00)**	**0.00e+00 (0.00e+00)**	**0.00e+00 (0.00e+00)**	**0.00e+00 (0.00e+00)**	**0.00e+00 (0.00e+00)**

Due to the above fact, T and E_r do not necessarily need to be set to 0 even if the function is convex or unimodal because it can be used to improve diversity within the population which may in turn lead to faster convergence. However, the experiments in this study chose to not include the SA aspects in the aforementioned tests to show that the DE-SA algorithm shows similar or even better performance in terms of convergence speed and solution quality than DE even when the SA aspect is turned off.

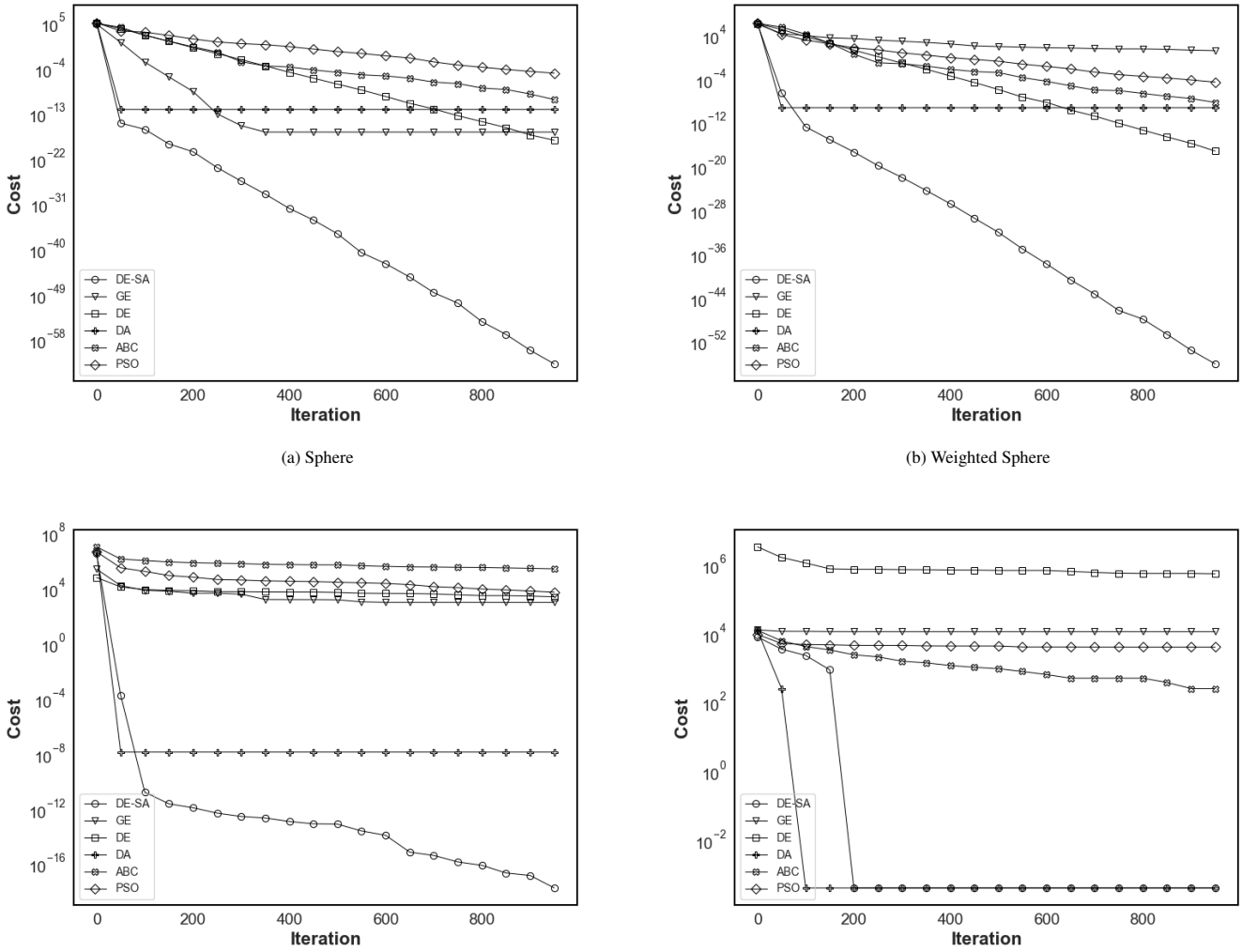
4.3. Experimental Results

The results are presented in the following four tables that show the best value obtained (Table 4), the number of iterations taken to reach the VTR, value to reach, (Table 5), the number of function evaluations taken to reach the VTR (Table 6), and the wall clock time it takes to reach the VTR (Table 7). Each entry is in the following format. Each of the algorithms is run 30 times and the measured values are averaged. Furthermore, each run uses a different seed and the seed for each respective run is the same for all the algorithms.

```
(value)
(std of value){*}
```

The `value` is the value of the measured quantity, `std of value` is the standard deviation of the value, and $*$ is optional depending on whether the algorithm is able to converge to the VTR within 1000 iterations or not. If the algorithm is not able to, the result will be followed by the $*$ character. Bold entries are the best value presented in that row. Each of the algorithms is run for 1000 iterations, and VTR is set to 10e-8 for all test functions apart from De Jong 5 where the VTR is 0.999. In addition, convergence graphs are also plotted for all test functions for each algorithm to track each algorithm's convergence. The convergence graph shows the convergence of the 30th run of the algorithm.

The results in Table 4 suggest that the DE-SA performs much better than other algorithms regarding solution quality. For convex functions such as the Sphere, Weighted Sphere, and Rotated Hyper Ellipsoid, the algorithm converges easily to the global minimum. Moreover, the terminating value is very close to zero compared to other algorithms. For Schwefel 1.2, even though the function is inherently convex the obtained solution is not as good as the other convex functions due to the fact that the area near the minimum is flat, so the search becomes increasingly harder near the minima. However, it is still better than



(a) Sphere

(b) Weighted Sphere

(c) Schwefel 1.2

(d) Schwefel 2.3

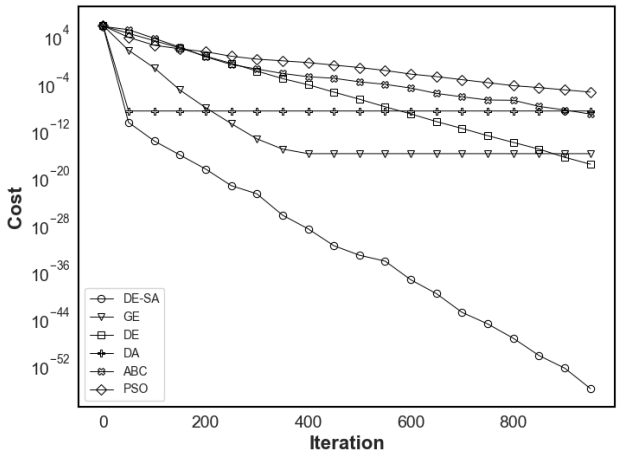
Figure 2: Convergence graph for each test function

most of the algorithms that even struggle to go under 100. Only dual annealing apart from DE-SA was able to achieve a value - 2.02e-08 - near the minimum of 0. This is due to the fact that in SciPy's implementation of DA, the program is also applying local search techniques such as the Gradient Method as well. This allows DA to find the minimum quickly and get a better solution. However, the solution is not as refined as the DE-SA due to DE's nature where the solution can be let to continuously improve if the vectors are near the minimum and there is enough diversity. On the other hand, as time approaches infinity, the rate of temperature decrement is slower. Coupled with the fact that DA chooses a possible visit by sampling from the distribution derived from the Cauchy distribution with a heavier tail and parameterized by T, the sampled visit will likely overshoot the minimum causing it to be stuck because T decreases too slowly.

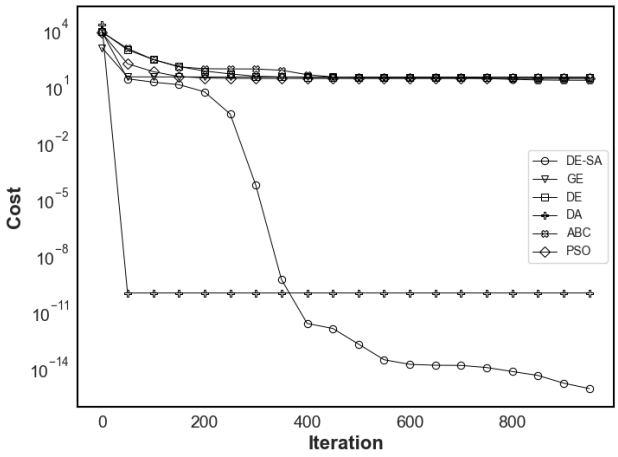
For nonconvex unimodal functions such as the Easom, the DE-SA can also converge to the absolute minimum in all of the runs. Similar results are shown by DE and PSO for the Easom function while the other algorithms struggle to attain the minimum. Similar results for DE-SA are shown in De Jong 3 where gradients are not presented.

The interesting results that distinguish DE-SA from other algorithms are from the Rosenbrock function where DE-SA is the only algorithm that is able to reach the VTR in all runs. Moreover, the standard deviation is also low which implies consistent performance. GE, DE, ABC, and PSO are among the ones that are not able to achieve the VTR in every possible run and DA is able to achieve VTR in some of the runs. This is due to the fact that DE-SA uses the gradient method to improve the search direction of the whole population which leads to better refinement of objective value - and increased speed - in each subsequent generation when compared to DE which can easily get stuck about the minimum because of the narrow valley.

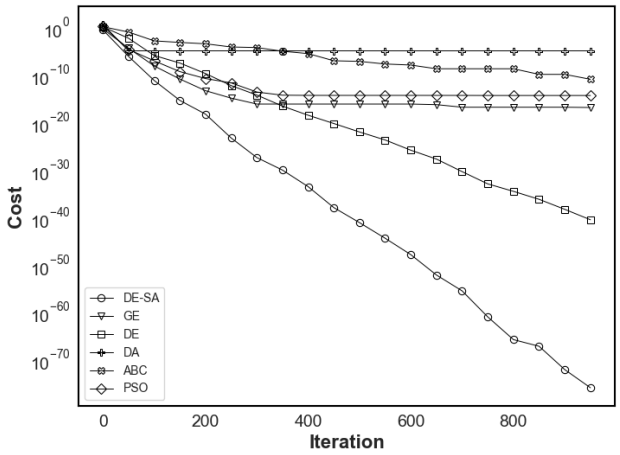
For Schwefel 2.3, which belongs to the multimodal class function, only DE-SA and DA are able to achieve the global



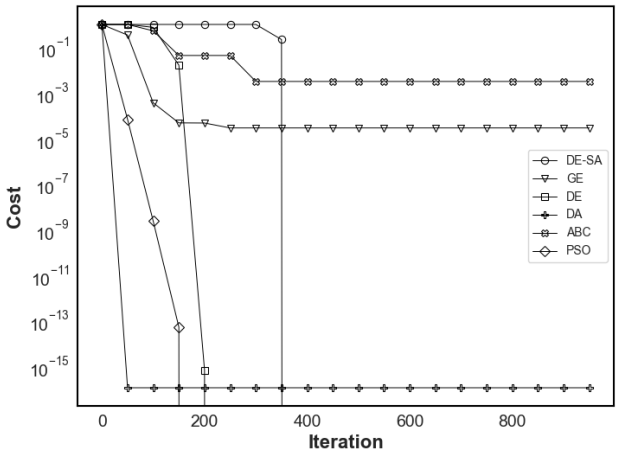
(e) Rotated Hyper Ellipsoid



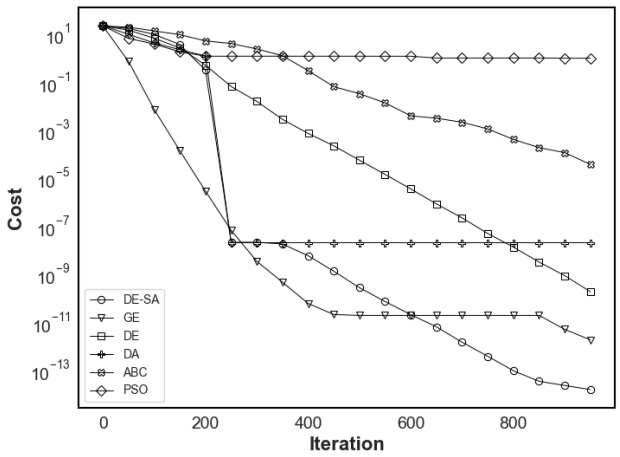
(f) Rosenbrock



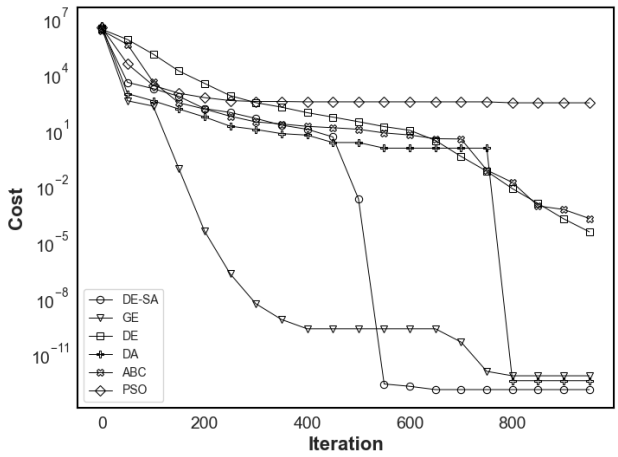
(g) Sum of Different Powers



(h) Easom

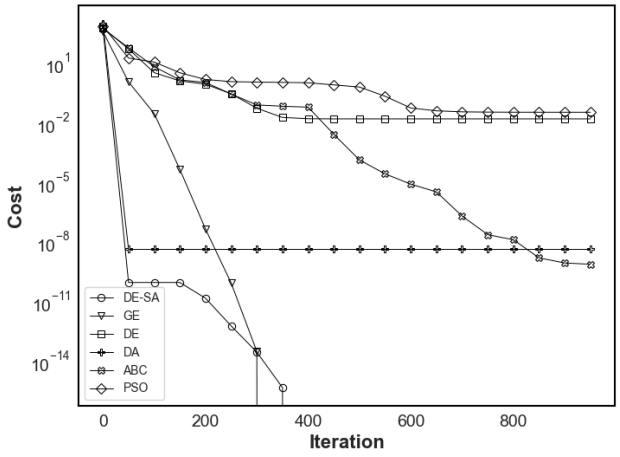


(i) Ackley

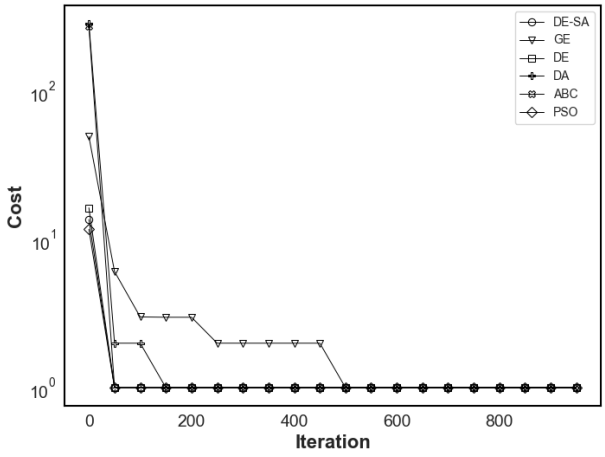


(j) Rastrigin

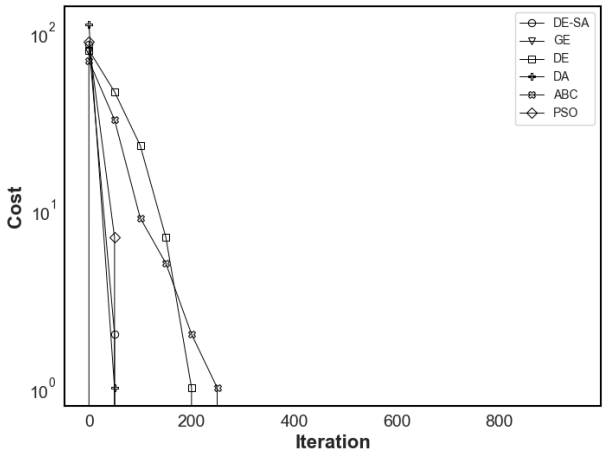
Figure 2: (continued)



(k) Griewank



(l) De Jong 5



(m) De Jong 3

Figure 2: (continued)

minimum of 3.82e-04. Other algorithms are not able to break the 1e+2 barrier with the closest second coming in at 3.36e+02 by the ABC algorithm. Compared to DA which while also always converges to the global minimum, DA has a lesser standard deviation which shows that it also converges more consistently within the minimum basin. Similar results are shown in Ackley, Rastrigin, and Griewank where DE-SA performs the best.

Furthermore, the figure 2 demonstrates that for Sphere, Weighted Sphere, Rotated Hyper Ellipsoid, and Sum of Different Powers, the solution quality obtained by the DE-SA and DE can still be improved because the curve is still heading down unlike other algorithms that plateaued out early. This occurs because the functions are convex and have only one minimum therefore the vectors are already in the minimum basin. With this condition, the vectors can continuously improve themselves due to the differencing of vectors which doesn't often overshoot because if all vectors are near the optimum, the difference (step size with direction) is automatically smaller; in addition, we also scale this process by the differential factor F. This is true for any local minima and can be seen in multimodal cases like Rastrigin's function that contains multiple local minima about the global minimum, and therefore it's harder for the vectors to jump off the local minima into the global minimum basin. Therefore, SA acceptance criteria can be very useful to allow these vectors to do hill climbing in the hope that they will find a better minimum.

Although both DE and DE-SA are able to perpetually refine the solution in the convex case, applying the gradient method can still speed up the convergence by warm-starting the population at a better solution as can be seen in figure 2a, 2b, 2e, and 2g. In any case, initially boosting the population's fitness using the gradient method is still effective, especially for evolutionary algorithms like DE which requires other vectors to be fit as well in order for it to improve.

Consequently, methods that have a fixed step size like GE

Table 5: Number of iterations to reach the VTR

Test Function Name	DE-SA	GE	DE	DA	ABC	PSO
Sphere	3.300 (2.397)	182.333 (9.217)	487.967 (10.959)	**1.000 (0.000)**	713.033 (45.066)	983.433 (23.410)
Weighted Sphere	48.733 (5.876)	1000.000 (0.000)*	532.500 (11.555)	**1.000 (0.000)**	794.700 (33.971)	1000.000 (0.000)*
Schwefel 1.2	60.933 (11.048)	1000.000 (0.000)*	1000.000 (0.000)*	**1.000 (0.000)**	1000.000 (0.000)*	1000.000 (0.000)*
Schwefel 2.3	191.833 (11.685)	1000.000 (0.000)*	1000.000 (0.000)*	**49.867 (15.233)**	1000.000 (0.000)*	1000.000 (0.000)*
Rotated Hyper Ellipsoid	41.333 (4.671)	200.233 (8.747)	515.733 (7.066)	**1.000 (0.000)**	772.300 (42.848)	999.333 (3.590)
Rosenbrock	318.000 (44.184)	1000.000 (0.000)*	1000.000 (0.000)*	**89.667 (211.934)**	1000.000 (0.000)*	1000.000 (0.000)*
Sum of Different Powers	**52.667 (6.257)**	102.667 (6.615)	505.367 (72.522)	933.400 (249.194)	542.233 (65.698)	97.733 (14.635)
Easom	147.867 (161.026)	1000.000 (0.000)*	204.867 (50.534)	729.167 (335.251)	1000.000 (0.000)*	**86.033 (15.394)**
Ackley	237.067 (21.437)	318.033 (20.106)	732.300 (12.413)	**222.467 (22.007)**	1000.000 (0.000)*	1000.000 (0.000)*
Rastrigin	506.433 (22.212)	**296.867 (37.728)**	1000.000 (0.000)*	688.933 (175.510)	1000.000 (0.000)*	1000.000 (0.000)*
Griewank	**16.600 (6.606)**	190.300 (12.144)	652.667 (194.875)	50.600 (196.919)	952.867 (59.658)	994.833 (19.081)
De Jong 5	**15.700 (2.968)**	195.033 (138.168)	44.933 (177.375)	67.733 (55.180)	29.367 (12.955)	54.200 (43.903)
De Jong 3	58.800 (3.945)	**27.067 (8.989)**	225.000 (13.066)	62.767 (15.372)	269.367 (57.097)	153.633 (190.385)

for updating the best vector in the population by moving by approximating the gradient direction using γ or methods that have a dynamic step size but the decrement is too gradual like DA that sample from the visiting distribution that is parameterized by T which decreases too slow later on will fail to converge around the minimum and plateaus out as can be observed in figure 2g, 2i, and 2c. Therefore, for refining solutions, methods like DE show supreme superiority over other heuristics because the step size is dynamic and depends on the solution itself. In addition, for multimodal functions like Ackley, Rastrigin, and Griewank, using gradient methods such as the BFGS aids the DE-SA's vector to move to each local minimum quicker and hence is able to drag other vectors along faster when compared to DE that got stuck for a while before being able to find enjoy logarithmic cost decrement speed in figure 2j.

Also, in figure 2a, 2e, and 2i, we observe that DE-SA initially started out with very fast convergence and took some time before started decreasing again. This is due to only some vector being applied with the gradient method so achieving rapid convergence before the gradient method is no longer applicable because the Jacobian is near zero about the minimum and so it terminates. What is left from the process is very few good individual vectors and an overall population that is not well-fitted. This creates a gap and therefore for the best vector to improve, it needs to wait for the other vectors to descend down as well otherwise the trial vector generated for these early fit vectors will always overshoot leading to it being stuck for a while as observed in preliminary experiments. This is where other mutation strategies also come into play like the $DE/best/1/bin$ and $DE/best/2/bin$ that use the best vector as the base and search from there; hence, overall population convergence will be faster and so the best vector can continue to improve faster as well.

In terms of the number of iterations to reach the VTR, DA mostly performs the best. Note that it only takes one iteration for DA to reach the VTR for Sphere, Weighted Sphere, Schwefel 1.2, and Rotated Hyper Ellipsoid, because the functions are convex and DA internally, runs the gradient method until reaching the minimum every time the algorithm updates the value. While DE-SA is also running gradient methods, it is not guaranteed to run generation because the trigger is determined by the probability p_g. Furthermore, the number of maximum iterations for the gradient method in our implementation of DE-SA is limited to 10 iterations, meaning that after the gradient method is terminated, it is not guaranteed to achieve the local minimum either. These phenomena are shown in the Weighted Sphere, Schwefel 1.2, and the Rotated Hyper Ellipsoid where it

Table 6: Number of function evaluations to reach the VTR

Test Function Name	DE-SA	GE	DE	DA	ABC	PSO
Sphere	525.400 (229.006)	7373.667 (368.829)	19598.667 (438.359)	**185.000 (0.000)**	28541.333 (1802.649)	39337.333 (936.415)
Weighted Sphere	11915.533 (1813.344)	40246.433 (11.427)*	21380.000 (462.198)	**1206.967 (71.360)**	31808.000 (1358.829)	40000.000 (0.000)*
Schwefel 1.2	14297.633 (2055.951)	40378.267 (11.051)*	40040.000 (0.000)*	**4456.800 (492.396)**	40020.000 (0.000)*	40000.000 (0.000)*
Schwefel 2.3	109502.633 (7923.498)	40365.867 (26.248)*	40040.000 (0.000)*	**6869.033 (1058.233)**	40020.000 (0.000)*	40000.000 (0.000)*
Rotated Hyper Ellipsoid	10316.500 (1873.697)	8089.900 (349.943)	20709.333 (282.642)	**1177.000 (72.921)**	30912.000 (1713.916)	39973.333 (143.604)
Rosenbrock	77172.833 (12335.363)	40948.900 (12.828)*	40040.000 (0.000)*	**11654.367 (12893.346)**	40020.000 (0.000)*	40000.000 (0.000)*
Sum of Different Powers	5128.367 (2162.492)	4187.000 (264.917)	20294.667 (2900.896)	60254.067 (14964.637)	21709.333 (2627.926)	**3909.333 (585.417)**
Easom	5954.667 (6441.058)	40589.333 (26.642)*	8274.667 (2021.372)	**2945.467 (1331.302)**	41010.233 (0.616)*	3441.333 (615.747)
Ackley	35099.400 (3493.922)	**12805.767 (805.361)**	29372.000 (496.510)	27106.800 (1800.587)	40020.000 (0.000)*	40000.000 (0.000)*
Rastrigin	135240.333 (10776.965)	**2990.500 (378.056)**	40040.000 (0.000)*	58992.533 (10757.227)	40020.000 (0.000)*	40000.000 (0.000)*
Griewank	4576.933 (1782.871)	7692.333 (485.971)	26177.333 (7778.375)	**3948.400 (12085.961)**	38134.667 (2386.333)	39793.333 (763.227)
De Jong 5	668.000 (118.727)	7954.867 (5590.001)	1876.000 (7087.802)	**423.833 (237.003)**	1200.600 (522.038)	2168.000 (1756.129)
De Jong 3	2392.000 (157.785)	**1162.667 (359.545)**	9080.000 (522.660)	4539.933 (941.725)	10795.067 (2284.431)	6145.333 (7615.384)

takes much more iteration to reach the VTR than the DA.

For the Rosenbrock function, the DE-SA takes about 3 times more number of iterations to reach the VTR than the DA. However, the standard deviation of the number of iterations is much less than the DA which shows that the DE-SA converges consistently quicker than the DA which has the standard deviation of 211.934. Moreover, DA doesn't converge sometime unlike the DE-SA which always achieves the VTR. Note that GE was able to achieve the VTR and fast convergence only due to the fact that most of the test functions have minimum about the origin. This is because the vector jumping mechanism causes the difference of vectors to become near zero. However, if we shift these functions to have the minimum of any other points, GE struggles to perform as preliminary results showed. That aside, DE-SA comes in second next to GE for power sum at just around 120 iterations, and similarly for Ackley's function where DE takes around 10 iterations more than DA. For Griewank and De Jong 5, DE-SA performs the best and converges the fastest. Although DE-SA does not appear to converge to VTR the fastest, it still ranks second and converges to a much better solution on average as can be seen in Table 4 with better consistency as well.

However, since DE-SA is a population-based method that also uses the gradient method on top, the number of function evaluations will be very high due to the need to compute the Jacobian matrix multiple times. For this reason, DE-SA places very badly in terms of the number of function evaluations. This can be seen in harder test function that requires more iterations to reach the VTR such as the Rosenbrock, Rastrigin, and Schwefel 2.3. This, however, can be optimized. If a gradient method is applied to the solution recently, then we can programmatically block gradient method calls because the solution just arrived at the minima. However, some of these methods require a lesser number of function evaluations due to the fact in some runs they didn't reach the VTR. If we were to let the algorithm run until it reached the VTR, then it might be possible that the DE-SA will use a lesser number of function evaluations due to better exploration and exploitation of search space. As there is a direct positive correlation between the number of function evaluations with the time complexity, DE-SA inevitably takes more time to reach the VTR than DA in most cases. However, the difference is not too large and in practice, this is negligible with only ± 1.5 seconds difference for $d = 30$. Especially for the Rastrigin and Rosenbrock function where the minima are narrow, more line search iterations are needed to be performed leading to more computations. Therefore, the tradeoff is rea-

Table 7: Number of seconds to reach the VTR

Test Function Name	DE-SA	GE	DE	DA	ABC	PSO
Sphere	0.010 (0.004)	0.323 (0.026)	0.486 (0.025)	**0.004 (0.000)**	0.623 (0.041)	2.013 (0.062)
Weighted Sphere	0.136 (0.017)	1.644 (0.034)*	0.391 (0.009)	**0.012 (0.001)**	0.389 (0.020)	1.807 (0.018)*
Schwefel 1.2	**0.770 (0.110)**	13.678 (0.159)*	11.750 (0.050)*	1.277 (0.139)	12.470 (0.084)*	12.668 (0.144)*
Schwefel 2.3	1.956 (0.138)	1.812 (0.058)*	11.694 (0.045)*	**0.150 (0.027)**	0.880 (0.014)*	2.302 (0.169)*
Rotated Hyper Ellipsoid	3.071 (0.561)	2.689 (0.198)	6.075 (0.111)	**0.352 (0.023)**	9.958 (0.532)	13.239 (0.230)
Rosenbrock	1.430 (0.222)	1.922 (0.042)*	1.041 (0.015)*	**0.236 (0.321)**	0.628 (0.013)*	2.086 (0.027)*
Sum of Different Powers	**0.118 (0.040)**	0.193 (0.017)	0.559 (0.081)	1.637 (0.417)	0.478 (0.059)	0.213 (0.034)
Easom	0.080 (0.088)	0.964 (0.026)*	0.099 (0.029)	**0.055 (0.025)**	0.389 (0.006)*	0.141 (0.026)
Ackley	1.115 (0.107)	**0.725 (0.056)**	1.116 (0.027)	0.892 (0.060)	1.422 (0.023)*	2.562 (0.023)*
Rastrigin	2.642 (0.212)	**0.129 (0.018)**	1.232 (0.018)*	1.449 (0.288)	0.969 (0.019)*	2.142 (0.018)*
Griewank	0.212 (0.083)	0.547 (0.040)	1.415 (0.422)	**0.200 (0.636)**	1.892 (0.114)	3.127 (0.070)
De Jong 5	0.015 (0.003)	0.325 (0.240)	0.045 (0.181)	**0.012 (0.006)**	0.020 (0.009)	0.113 (0.093)
De Jong 3	0.054 (0.003)	**0.047 (0.017)**	0.189 (0.013)	0.105 (0.023)	0.144 (0.031)	0.392 (0.472)

sonable for a huge improvement in solution quality.

Moreover, for higher dimensions, DA will take more function evaluations and time to reach the VTR due to the fact that in one iteration it is making at least $2D$ steps hence function evaluations. Therefore, for higher dimensions like 100, it will be making at least 200 evaluations which can increase the number very easily unlike DE-SA, DE, and GE which doesn't create a neighborhood depending on the number of dimension itself. However, since DE-SA is using the gradient method, time complexity will be contributed from the curse of dimensionality up to some order.

4.4. Statistic Test Results

Wilcoxon Signed Rank Test has been used to compute the statistical test results in a pairwise manner between the DE-SA and other algorithms. Let x^* and y^* be the solutions obtained from DE-SA and another algorithm respectively, then the hypotheses for our statistical test are the following.

$$\begin{aligned} H_0, &\quad \text{Med}(f(x^*)) = \text{Med}(f(y^*)) \\ H_a, &\quad \text{Med}(f(x^*)) < \text{Med}(f(y^*)) \end{aligned} \tag{22}$$

The statistical test results are shown in table 8 for the best value that the algorithms are able to achieve after it has been running for 1000 iterations. For most of the cases, we are able to reject the null hypothesis that both algorithms performed equally and claim that the DE-SA performed better than the compared algorithm; for some that we cannot reject, it is due to both being able to obtain the global optimal solution apart from DE-SA vs GE on Griewank where GE is able to perform better but DE-SA takes lesser iterations to converge.

5. Improvement and Further Research

The following section discusses the possible improvement to the current algorithm which includes the gradient method application criteria, mutation strategies, and the cooling schedule.

5.1. Gradient Method Application

Instead of applying the gradient method directly to a vector in the population, we can check for the application criteria and apply it after the crossover instead. This ensures that we do not make redundant gradient descent on the vector that reaches the minimum because the trial vector will be a new point that might not yet reach the minimum. This increases the chance that the algorithm will find new and better minima than the current one,

Table 8: Statistic Test Table for VTR. Pairwise comparison of DE-SA algorithm and other algorithms in terms of last terminating value.

Test Function Name	GE	DE	DA	ABC	PSO
Sphere	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Weighted Sphere	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Schwefel 1.2	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Schwefel 2.3	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Rotated Hyper Ellipsoid	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Rosenbrock	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Sum of Different Powers	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Easom	9.31e-10	1.00e+00	2.80e-06	9.31e-10	1.00e+00
Ackley	9.31e-10	9.31e-10	9.31e-10	9.31e-10	9.31e-10
Rastrigin	2.46e-06	9.31e-10	1.81e-06	9.31e-10	9.31e-10
Griewank	1.00e+00	8.57e-03	9.31e-10	9.31e-10	9.31e-10
De Jong 5	9.31e-10	2.46e-01	9.31e-10	9.31e-10	1.03e-01
De Jong 3	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00

and moreover stop making redundant function evaluations that don't improve the quality of the solution.

In addition to the above modification, a table can also be added to keep track of which vector in the population has been recently updated with the gradient method. If a vector is updated using the gradient method recently, then we can defer the update so that gives the vector enough time to move out of the local minima before applying the gradient method again. This serves two purposes. First, if the gradient method is applied too early, then there's a chance that the vector will slide back down the local minima, countering the hill climbing part from Simulated Annealing. Second, it prevents excessive gradient computation - which is time-consuming - when the Jacobian is already near zero.

Lastly, instead of setting the maximum number of iterations for the gradient method, we can allow it to run until the algorithm terminates instead so we are sure that it will reach the local minimum or terminate due to precision loss, both of which are reasonable criteria to stop.

All of these ideas can be encapsulated in the form of a table that is maintained when each vector is applied with the gradient method. If the vector is applied with the gradient method within some defined timeframe - called the cooldown period, then we will skip the process. Furthermore, instead of having a new parameter for the cooldown period, we can make the cooldown period a function of p_g instead, where it is equal to $\lceil 1/p_g \rceil$ which is the expected value of iteration before the application of gradient method to some vector. This can tremendously improve the time complexity of the algorithm.

5.2. Mutation Strategies

To maintain diversity and increased convergence speed, the current mutation strategy, $DE/1/rand/bin$ can be swapped out in favor of something else depending on each function instead. In addition, a combination of mutation strategies could also be used to improve convergence speed. For example, there is a $1/3$ chance of using the $DE/1/best/bin$ strategy and a $2/3$ chance of using $DE/1/rand/bin$. As mentioned in earlier sections, this will help close the gap between the whole population and the best individual earlier in the optimization phase because the best individual often has a much better fitness value due to being applied with the gradient method. This will improve DE convergence speed because the best individual cannot be improved unless there are at least 3 other vectors that are nearly as good as itself due to DE's mutation nature. However, since we do not want to impact the diversity which is another huge factor in convergence, we also apply the same random mutation strategy based on $DE/rand/1/bin$.

5.3. Cooling Schedule

Another improvement that can be made is to borrow the cooling schedule of the Fast Simulated Annealing where T is allowed to be decreased like the inverse of time. Consequently, this makes the Fast Simulated Annealing, and so the Generalised Simulated Annealing and Dual Annealing, much faster and more efficient Tsallis and Stariolo (1996). This fast cooling schedule is possible due to the semi-local search visiting distribution that is derived from the Cauchy-Lorentz distribution that makes frequent local jumps and occasional long jumps. In our case, the gradient method and DE mutation strategy will make a long jump initially and so converge quickly so it might be possible to adopt DA's scheme instead of the classical Boltzmann machine's. Let E be the event that the new solution is accepted, and $\Delta = f(x_{t+1}) - f(x_t)$, then the borrowed acceptance probability is defined in equation 23 and T is updated per equation 24.

$$P(E) = \left\{ \begin{array}{ll} 1, & \text{if } \Delta < 0 \\ \left[(1 + (q_A - 1)(\Delta)/T)^{1/(q_A-1)}\right]^{-1}, & \text{otherwise} \end{array} \right\} \quad (23)$$

$$T = T_0 \cdot \frac{\exp((q_V - 1) \cdot \ln 2) - 1}{\exp((q_v - 1) \cdot (t + 2)) - 1} \quad (24)$$

where q_A and q_V are the acceptance and visit parameter that controls how likely are we to accept a worse solution and how fast the temperature decreases respectively. In the Dual Annealing algorithm, the visiting distribution is a function of T which makes shorter steps as time progresses; to mimic that idea, the differential factor F can be made as a function of T as well, where initially F is high and gradually decreases over time to search locally which is similar to Rafajłowicz (2015). Moreover, since T is now allowed to decrease as a function of time, we can also introduce re annealing mechanism. This can possibly allow for faster convergence and also decrease the number of parameters that need to be tuned.

6. Summary and conclusions

In this study, we proposed new hybrid metaheuristics that generalize the Differential Evolution and Simulated Annealing and exploit gradient information of the underlying function to improve the convergence speed and quality of the solution. The backbone of the algorithm is based on Differential Evolution where it employs DE's mutation and crossover strategy while maintaining diversity through semi-elitist selection criteria from Simulated Annealing Boltzmann selection scheme. On top of that, gradient methods are also introduced as another mutation step to boost convergence, refine the solution, and give a better search direction for DE's vector mutation. Moreover, when the gradient method and SA aspect are turned off, the algorithm reverts back to DE which guarantees that the DE-SA will perform as well as or better than the original.

Conducted experiments demonstrated that the DE-SA hybrid outperforms other heuristics - GE, DE, DA, ABC, and PSO - in terms of solution quality significantly for selected functions with varying characteristics. For convex functions, the DE-SA converges quickly and consistently to the optimum. For harder multimodal functions, the DE-SA is also able to perform well and achieve the VTR in every run. For relatively flat search space, the DE aspect of the algorithm allows it to search the search space widely and achieve the global minimum as well. Furthermore, the Wilcoxon Signed Rank Test is used to perform a pairwise comparison between DE-SA and each algorithm to confirm that DE-SA performs better and no worse than other algorithms in terms of solution quality. However, optimizations are also needed to decrease unnecessary function evaluations and improve the algorithm's speed; moreover, to show that DE-SA performs well in practical settings, it is imperative to use more complicated test functions. Lastly, more research and experiments are needed to improve the performance by bringing in DA's fast cooling schedule and acceptance criteria, yet for high-level metaheuristics where one could take and implement specifically for a problem, this is sufficient.

Acknowledgements

Thanks to Nattakorn Massaya-anon (NCSU) and Peeranat Tanasatitchai (CU) for supplying me with literature that I cannot afford.

References

Atangana, A., 2018. Chapter 2 - principle of groundwater flow, in: Atangana, A. (Ed.), Fractional Operators with Constant and Variable Order with Application to Geo-Hydrology. Academic Press, pp. 15–47. URL: https://www.sciencedirect.com/science/article/pii/B9780128096703000023, doi:https://doi.org/10.1016/B978-0-12-809670-3.00002-3.

Blank, J., Deb, K., 2020. pymoo: Multi-objective optimization in python. IEEE Access 8, 89497–89509.

Dai, Y.H., 2002. Convergence properties of the bfgs algoritm. SIAM Journal on Optimization 13, 693–701. doi:10.1137/S1052623401383455.

Kuo, R., Zulvia, F.E., 2015. The gradient evolution algorithm: A new metaheuristic. Information Sciences 316, 246–265. URL: https://www.sciencedirect.com/science/article/pii/S0020025515002996, doi:https://doi.org/10.1016/j.ins.2015.04.031. nature-Inspired Algorithms for Large Scale Global Optimization.

Molga, M., Smutnicki, C., 2005. Test functions for optimization needs. Test functions for optimization needs 101, 48.

Odeniyi, O., Omidiora, O., Olabiyisi, O.S., Aluko, A., 2015. Development of a modified simulated annealing to school timetabling problem. International Journal of Applied Information Systems 8, 16–24. URL: https://api.semanticscholar.org/CorpusID:16419715.

Oliveira, S.C.P., 2021. beecolpy. - -, –. doi:-.

Rafajłowicz, W., 2015. A hybrid differential evolution-gradient optimization method, in: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (Eds.), Artificial Intelligence and Soft Computing, Springer International Publishing, Cham. pp. 379–388.

Rafati, J., Marcia, R.F., 2019. Deep reinforcement learning via l-bfgs optimization. arXiv:1811.02693.

Rodomanov, A., Nesterov, Y., 2022. Rates of superlinear convergence for classical quasi-newton methods. Mathematical Programming 194, 159–190. URL: https://doi.org/10.1007/s10107-021-01622-5, doi:10.1007/s10107-021-01622-5.

Shi, Y., Eberhart, R., 1999. Empirical study of particle swarm optimization, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), pp. 1945–1950 Vol. 3. doi:10.1109/CEC.1999.785511.

Storn, R., Price, K., 1997. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359. doi:10.1023/A:1008202821328.

Surjanovic, S., Bingham, D., . Virtual library of simulation experiments: Test functions and datasets. Retrieved November 24, 2023, from http://www.sfu.ca/~ssurjano.

Tibshirani, R., 2013. Gradient descent: Convergence analysis.

Tsallis, C., Stariolo, D.A., 1996. Generalized simulated annealing. Physica A: Statistical Mechanics and its Applications 233, 395–406. URL: https://www.sciencedirect.com/science/article/pii/S0378437196002713, doi:https://doi.org/10.1016/S0378-4371(96)00271-3.

Virk, Z., 2023. Gradient descent and quasi-newton methods.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17, 261–272. doi:10.1038/s41592-019-0686-2.