Table of Contents

Title and Abstract

Title: Comparing the Lamarckian and Baldwinian Approaches in Memetic Optimization

Abstract: Memetic optimization (MO) combines local and global search in optimization in non-monotonic, 'rugged' search spaces. In particular, MO extends genetic optimization, where global search is implemented via the crossover operator and local search is performed as random mutation. MO uses more elaborate techniques to implement local search and to combine it with its global counterpart.

This study is set to compare two ways of memetic optimization: one motivated by the Baldwinian effect, while the other by the Lamarckian theory of evolution.

Both the Baldwinian and Lamarckian theory suggest that behaviors of individuals are not only passed on to offspring by crossover and mutation, but also through lifetime learning. In Baldwin approach, learned behaviors affect the mapping of genotypes to phenotypes, which ultimately results in changes to the fitness landscape. In Lamarck approach, not only will the learned behaviors affect the fitness landscape in the same way as the former does, but they will also be passed on to offspring through phenotypes. Whilst the biological plausibility of these perspectives is questionable, they offer a valuable structure for constructing memetic optimization algorithms.

Before exploring Lamarckian and Baldwinian approaches, a baseline framework where offspring can only inherit the characteristics of the parent through crossover and mutation (i.e., genetic optimization) is considered as a global optimization problem. Based on the baseline framework, we develop various implementations of the Lamarckian and Baldwinian approaches exploring several local search procedures to study the potential contributions of the studied approaches. Our experiments will be performed on the CEC-BC-2017 test functions for optimization.

Keywords: Baldwinian evolution; Lamarckian evolution; Memetic optimization; Fitness landscape; Learning; CEC-BC-2017.

Genetic algorithm and Memetic algorithm

Genetic algorithm

Genetic Algorithm (GA) has become one of the critical methods behind solvers capable of addressing large-scale real-world optimization problems. It has been positively explored in research institutes as well as being actively employed in industry. GA is a population-based stochastic algorithm originated from Darwinian theory of evolution, belonging to the broader category of Evolutionary Algorithms. It consists of three principal bio-inspired operators: selection, crossover, and mutation [1-3]. Inspired by that only superior individuals in the population have the chance to produce offspring and pass on their genes, selection operator only selects the fittest individuals, and this selectivity lays the most important foundation for GA to be applied to solve optimization problems. Owing to the contribution of the selection operator, GA always maintains the best solution in each generation and evolves towards better solutions. Part of selection operators such as local selection, fuzzy selection, fitness uniform selection, linear rank selection, steady-state reproduction can be found in [4-8]. Crossover operator allows two parents to swap their genes with a certain probability, thereby producing a solution between two points [1-3]. A variety of ways to implement crossover operators such as uniform crossover, half uniform crossover, three parent's crossover, partially matched crossover, cycle crossover, order crossover and position-based crossover are introduced in the literature [9-15]. Different from crossover, mutation operator randomly alters some genes of an individual by chance, resulting in the production of another new solution, which improves the diversity of the whole population [1-3]. Some mutation operators such as Gaussian, shrink, supervised mutation, uniqueness mutation, varying probability of mutation can be accessed in [16-20].

As mentioned earlier, only individuals that are adapted to their environment will survive. How well an individual is adapted to its environment is determined according to a fitness function, i.e., the objective function. Each individual's genotype is a solution to the fitness function. Therefore, the fitness value of each individual is the result of the calculation performed by bringing the genotype into the fitness function. A Fitness landscape is applied to help visualize the relative locations between genotypes and a fitness function. A fitness landscape is a (N+1)-dimensional view of a fitness function while a genotype is a N-dimensional point. The height of a fitness landscape represents fitness value of an individual [21-22].

Figure 1(left part) presents the process of a simple genetic algorithm. The evolution starts from a population with randomly generated individuals. GA measures the fitness value of each individual and selects the most adapted individuals, then creates new offspring by crossover and mutation with a certain probability. New individuals will join the population and participate in the next iteration. After continuous iteration, GA moves towards better population, but it is not guaranteed to find the global optima every time, depending on multiple factors such as the number of iterations and the complexity of the objective function.

Memetic algorithm

With the influence of R. Dawkins' notion of 'memes' [23], Pablo Moscato first introduced memetic algorithm (MA) in 1989 and he proposed that memetic algorithm is similar to a mixed population-based genetic algorithm combined with an individual's learning procedure that enables local fine-tuning [24]. Broadly speaking, MA is an extension of GA by introducing local search heuristics into the framework of stochastic global search techniques [25-27], which decreases the probability of the premature convergence of GA to a certain level [27-28]. Local search procedure (LSP) is a type of optimization method that explores a small space nearby the current solution and replaces the current solution by a better one if exists [29].

In searching for the optimal value of a fitness function, crossover operator gives individuals the ability to jump across the landscape of a fitness (objective) function while mutation operator enables individuals to explore local environment in fitness. MA further reinforces local search on the basis of mutation.

Figure 1(right part) presents one way of implementing a simple memetic algorithm.

In genetic algorithms, each individual has its own genotype, which is a solution to an objective function. With the help of a local optimizer, MA generates a new solution in the vicinity of the fitness landscape where the genotype is located. The better of genotype and the new solution will be involved in the later operations.

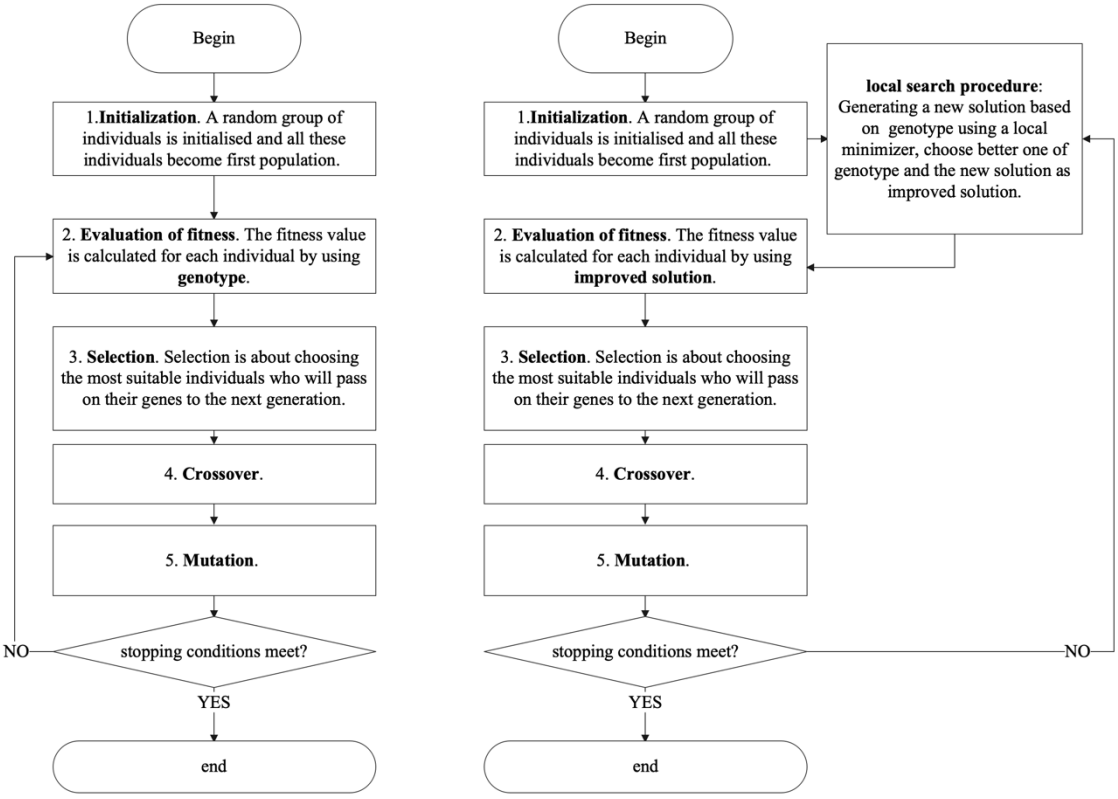


Figure 1 Flowchart of Genetic algorithm(left) and Memetic algorithm(right)

While in principle the enhanced exploitation of candidate solutions by adding a local search procedure should lead to an improvement of the algorithm, this is not always the case [25]. For example, if the population is almost converged and has reached a local optimum, then no local search procedure will find a better solution than the current one, and in this case, it simply plays no role. Our expectation is that memetic algorithms will create collaborative effects that usefully bridge local and global search.

Baldwinian effect and Lamarckian evolution

The Baldwinian effect and Lamarckian evolution are both memetic algorithms which focus on how individual's lifetime learning affects the evolutionary direction [30-36]. Individuals can develop a phenotype based on its own genotype through lifetime learning, this ability allows individuals to do some local exploration around the current location (i.e., where the genotype is located). In computation, genotypes and phenotypes are typically high-dimensional vectors while their biological significance can be conceived as the DNA of an individual and the physical body of an individual [37].

Both the Baldwinian effect and Lamarckian evolution suggest that behaviors of individuals are not only passed on to offspring by crossover and mutation, but also through lifetime learning. In Baldwin approach, learned behaviors affect the mapping of genotypes to phenotypes, which ultimately results in changes to the fitness landscape. In Lamarck approach, not only will the learned behaviors affect the fitness landscape in the same way as the former does, but they will also be passed on to offspring through phenotypes. Whilst the biological plausibility of these perspectives is questionable, they offer a valuable structure for constructing memetic optimization algorithms [37-40].

For instance, a genotype is a point on the fitness landscape and the height of the point represents the fitness value of an individual. Our goal is to find the global maximum of the fitness function, which means that a higher fitness value for an individual corresponds to a higher position on the fitness landscape. If the fitness value of the phenotype is higher, then the fitness value of this individual is replaced by the fitness value of the phenotype. The mapping of genotype to phenotype can be appreciated as a lifelong learning outcome for the individual, which leads to a change on the fitness landscape.

The difference is that phenotype is not inherited to the offspring in Baldwin approach, but it can be in Lamarck approach. Both approaches have their strengths and weaknesses, and it is difficult to judge which is better than the other. Baldwin approach has been found to perform better than Lamarck approach in dynamic environments [41-42], while Lamarck approach is found to be better at solving optimization problems of convex functions [43-44].

Contribution of this work

We offer two versions of memetic algorithms: one motivated by the Baldwinian effect, while the other by the Lamarckian theory of evolution, namely as Baldwin algorithm and Lamarck algorithm, respectively. Before implementing the two algorithms, we create a baseline called Steady-state genetic algorithm (SSGA) as global optimization, then we develop Baldwin and Lamarck algorithms by adding local search procedures into SSGA as a combination of memetic optimization and global optimization. The first goal of this paper is to evaluate the effectiveness of local search procedures in Baldwin and Lamarck algorithms. The second goal of this paper is to compare the performance of Baldwin and Lamarck algorithms by using benchmark functions CEC-BC-2017 [45]. It includes unimodal functions, multimodal functions, hybrid functions and composition functions with high dimensions, etc., which can be used to universally assess proposed Baldwin and Lamarck algorithms. This is what makes this study unique.

Methods

Fitness function

CEC-BC-2017 contains 23 benchmark functions with different characteristics and levels of difficulty, which can be used to assess the performance of new algorithms. The performance of our algorithms is evaluated based on CEC-BC-2017 test functions for optimization. Table 1 shows 23 test functions with details. The header of Table 1 includes: function, dimension, domain, and the optimal. The number of dimensions varies from 2 to 50. Each function has a domain, which explains the upper and lower bounds of the function. A global optimal value is the smallest value that a function can achieve within its domain.

Figure 2 shows the first two-dimensional view of fitness landscape for part of CEC-BC-2017 benchmark functions, considering the layout, two-dimensional view of fitness landscape for all functions is not shown, but one can get a two-dimensional view of fitness landscape for all the functions in the appendix. Figure 2 has 12 subpictures, and each subpicture is a fitness

landscape view of a fitness function in two dimensions with a title specifies function's name. Of all 23 functions, the smallest number of dimensions is 2. For normal high dimensional functions, the first and second dimensions are picked up for generating this view. For high-dimensional and dimension-fixed functions, such as F15, F19, F20, F21, F21, F22 and F23, the first and second dimensions are picked while other dimensions still exist, but values of other dimensions are set to zero.

Steady-state Genetic Algorithm

Steady-state genetic algorithm (SSGA) is implemented as a baseline framework in order to demonstrate the progress and improvements which Lamarckian and Baldwinian approaches provide. SSGA maintains a stable population size by generating only one new offspring based on the best individual in the population, while discarding the least adapted individual. The individual holding the lowest position in the fitness landscape is defined as the best individual, as our goal is to find the global minimum. In contrast, the individuals who are least adapted to the environment have the highest position. For all individuals in SSGA, genotypes and phenotypes are vectors of equal dimensions and equal values at corresponding positions in every test function of CEC-BC-2017. Fitness value of an individual is computed based on its phenotype. The genotypes of the primordial populations are randomly created with every gene located within the domain of a test function.

Figure 2 two-dimensional view for part of test functions

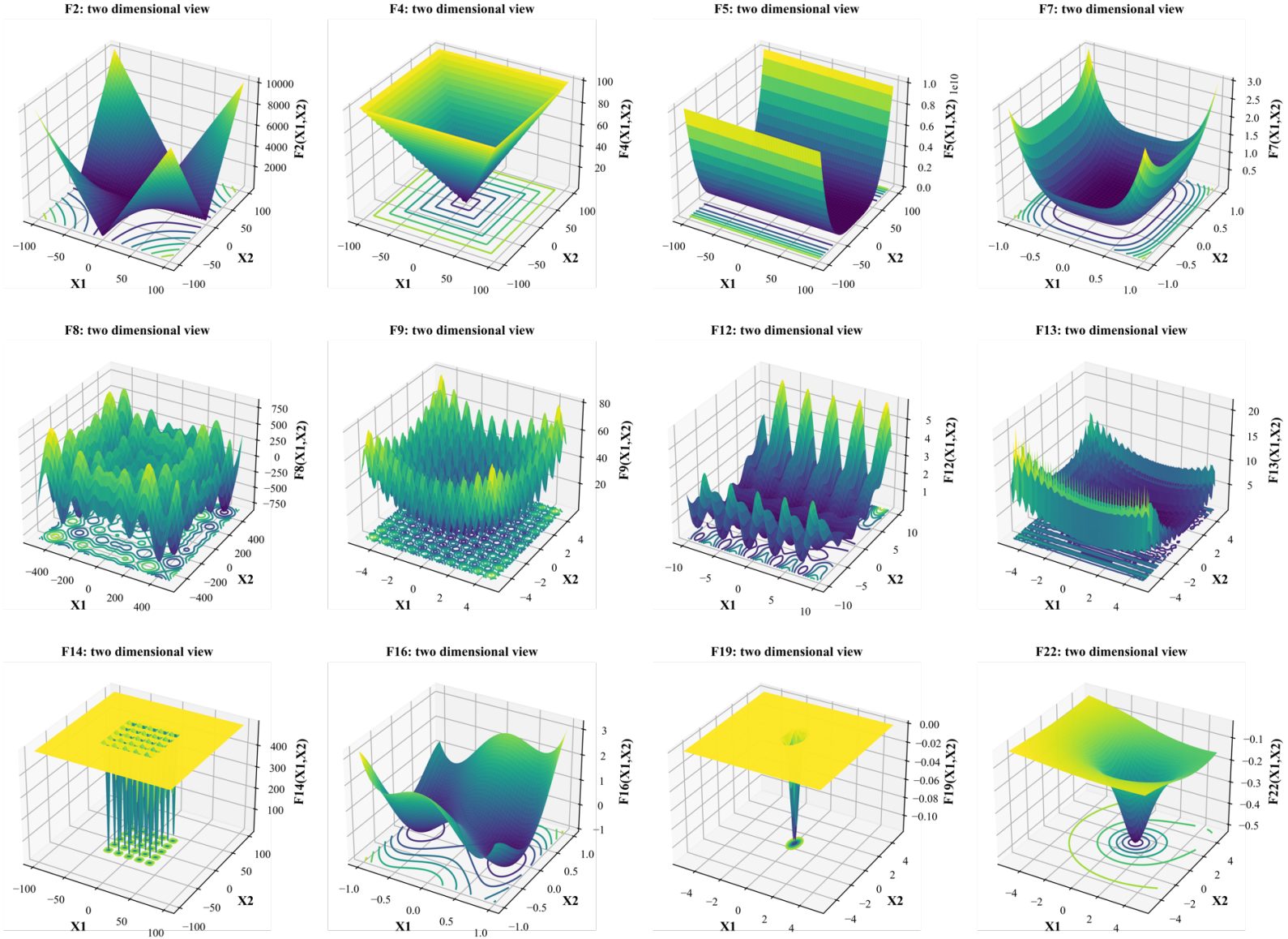


Table 1 CEC-BC-2017 benchmark functions

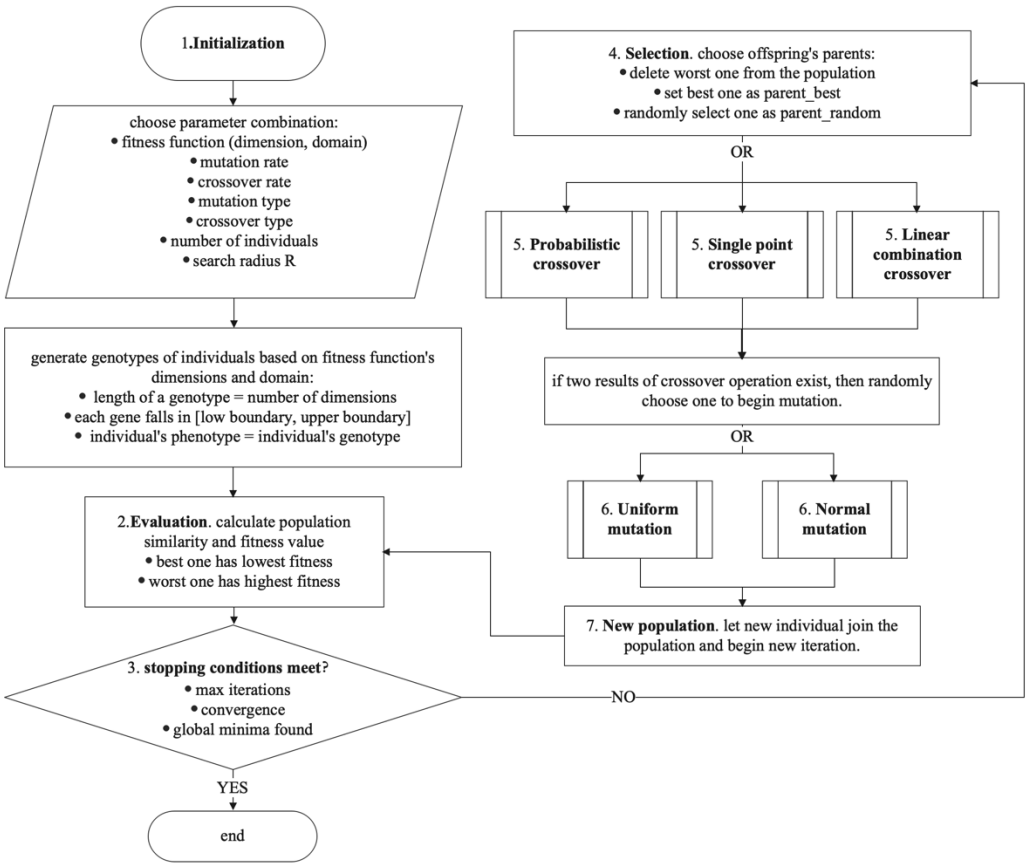
Function	Dim	Range	Optimal
$F1 = \sum_{i=1}^{50} x_i^2$	50	[-100,100]	0
$F2 = \sum_{i=1}^{50} \lvert x_i \rvert + \prod_{i=1}^{50} \lvert x_i \rvert$	50	[-100,100]	0
$F3 = \sum_{i=1}^{50} \left(\sum_{i=1}^{50} = (x_i)^2 \right)$	50	[-100,100]	0
$F4 = max\lvert x_i \rvert$	50	[-100,100]	0
$F5 = \sum_{i=1}^{49} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	50	[-30,30]	0
$F6 = \sum_{i=1}^{49} (x_i + 0.5)^2$	50	[-100,100]	0
$F7 = \sum_{i=1}^{50} (i x_i^4)$	50	[-1.28,1.28]	0
$F8 = \sum_{i=1}^{50} \left(-x_i sin\ \left(\sqrt{\lvert x_i \rvert} \right) \right)$	50	[-500,500]	-418.98 x d
$F9 = \sum_{i=1}^{50} (x_i^2 - 10\, cos(2\pi x_i) + 10)$	50	[-5.12,5.12]	0
$F10 = -20exp\left(-0.2 \sqrt{0.02 \sum_{i=1}^{50} x_i^2} \right) - exp\left(0.02 \sum_{i=1}^{50} cos\ 2\pi x_i \right) + 20 + e$	50	[-32,32]	0
$F11 = \frac{1}{4000} \sum_{i=1}^{50} x_i^2 - \prod_{i=1}^{50} cos\ \frac{x_i}{\sqrt{i}} + 1$	50	[-600,600]	0
$F12 = \frac{\pi}{50} \left(10sin^2(\pi y_1) + \sum_{i=1}^{49} (y_i - 1)^2 (1 + 10sin^2\ \pi y_{i+1} + (y_{50} - 1)^2) + \sum_{i=1}^{50} u(x_i, 10,100,4) \right)$	50	[-50,50]	0
$F13 = 0.1 \left(sin^2(3\pi x_1) + \sum_{i=1}^{49} (x_i - 1)^2 \left(1 + sin^2\ (3\pi x_i + 1)\right) + (x_{50} - 1)^2 \left(1 + sin^2\ (2\pi x_{50})\right) \right) + \sum_{i=1}^{50} u(x_i, 5,100,4)$	50	[-50,50]	0
a = [[-32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32],	2	[-65,65]	1

[-32, -32, -32, -32, -32, -16, -16, -16, -16, -16, 0, 0, 0, 0, 0, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32]]			
$F14 = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} \left(x_i - a_{ij}\right)^6}\right)^{-1}$			
a = [.1957, .1947, .1735, .16, .0844, .0627, .0456, .0342, .0323, .0235, .0246]; b = [.25, .5, 1, 2, 4, 6, 8, 10, 12, 14, 16]; b=1. /b	4	[-5, -5]	0.0003
$F15 = \sum_{i=1}^{11} \left(a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right)^2$			
$F16 = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	[-5, -5]	-1.0316
$F17 = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)cos\,(x_1) + 10$	2	[-5, -5]	0.398
$F18 = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times$ $[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	[-2, -2]	3
a = [[3, 10, 30], [.1, 10, 35], [3, 10, 30], [.1, 10, 35]]; c = [1, 1.2, 3, 3.2] p = [[.3689, .117, .2673], [.4699, .4387, .747], [.1091, .8732, .5547], [.03815, .5743, .8828]]	3	[0,1]	-3.86
$F19 = -\sum_{i=1}^{4}\left(c_i exp\left(-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right)\right)$			
a = [[10, 3, 17, 3.5, 1.7, 8], [.05, 10, 17, .1, 8, 14], [3, 3.5, 1.7, 10, 17, 8], [17, 8, .05, 10, .1, 14]]; c = [1, 1.2, 3, 3.2] p = [[.1312, .1696, .5569, .0124, .8283, .5886], [.2329, .4135, .8307, .3736, .1004, .9991], [.2348, .1415, .3522, .2883, .3047, .6650], [.4047, .8828, .8732, .5743, .1091, .0381]]	6	[0,1]	-3.32
$F20 = -\sum_{i=1}^{4}\left(c_i exp\left(-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right)\right)$			
a = [[4, 4, 4, 4], [1, 1, 1, 1], [8, 8, 8, 8], [6, 6, 6, 6], [3, 7, 3, 7]]; c = [.1, .2, .2, .4, .4]	4	[0,10]	-10.1532
$F21 = -\sum_{i=1}^{5}\left((X - a_i)(X - a_i)^T + c_i\right)^{-1}$			
a = [[4, 4, 4, 4], [1, 1, 1, 1], [8, 8, 8, 8], [6, 6, 6, 6], [3, 7, 3, 7], [2, 9, 2, 9], [5, 5, 3, 3]]; c = [.1, .2, .2, .4, .4, .6, .3]	4	[0,10]	-10.4028
$F22 = -\sum_{i=1}^{7}\left((X - a_i)(X - a_i)^T + c_i\right)^{-1}$			
a = [[4, 4, 4, 4], [1, 1, 1, 1], [8, 8, 8, 8], [6, 6, 6, 6], [3, 7, 3, 7], [2, 9, 2, 9], [5, 5, 3, 3], [8, 1, 8, 1], [6, 2, 6, 2], [7, 3.6, 7, 3.6]] c = [.1, .2, .2, .4, .4, .6, .3, .7, .5, .5]	4	[0,10]	-10.5363
$F23 = -\sum_{i=1}^{10}\left((X - a_i)(X - a_i)^T + c_i\right)^{-1}$			
$U(x, a, k, m) = k((x - a)^m)(x > a) + k((-x - a)^m)(x < (-a))$			

Figure 3 illustrates the flowchart of the SSGA algorithm. SSGA primarily consists of the following steps: (1) initialization of the population based on the parameter combination; (2) evaluation of the fitness value and similarity for the population; (3) checking stopping conditions;(4) selection of parents eligible to produce offspring and extinction of ineligible individuals; (5) crossover operation; (6) mutation operation; (7) insertion of new individuals into the population and begin execution of next iteration.

Stopping Criteria is generally organized as: (1) max number of iterations is reached; (2) a satisfactory fitness value of objective function has been found; (3) the similarity of populations is less than a pre-defined threshold for certain number of continuous iterations.

Figure 3 Flowchart of SSGA



The following are the parameters of the SSGA algorithm. The maximum number of iterations is Max_{iter}, terminate the program when Max_{iter} is reached. Convergence tolerance Tol is another stopping condition. If the Euclidean similarity of whole population is less than Tol in N continuous iterations, then terminate the program. Alternatively, if the fitness value of the best individual minus global minima opt is less than threshold θ, terminate the program. In addition, search radius R is a parameter that controls the range of the mutation.

Parameters for SSGA

Input: parameter combination

1. max iterations Max_{iter}
2. convergence tolerance Tol and N
3. fitness function F , number of dimensions of F is D , domain of F is $[F_{low},F_{upper}]$
4. mutation rate γ , crossover rate δ
5. mutation type M_t, crossover type C_t
6. number of individuals β
7. search radius R

8. global minima opt and threshold θ

Output: best solution X

The following **Pseudocode1** shows pseudo-code of SSGA.

PseudoCode1

// **(1) initialization of the population based on the parameter combination**

Randomly generate β feasible genotypes g of individuals based on F

Create phenotypes p of individuals where $p_i = g_i$

Save all the individuals in the population Pop

Set iteration $iter = 1$

// **(2) evaluation**

// **(2.1) evaluation of the fitness value**

For $k = 1$ to β do:

Calculate F_p^k fitness value for an individual k using its phenotype p

end

Sort all the F_p^k in an ascending order and change order of individuals' location in Pop accordingly

Best individual has smallest fitness value $F_p^{k=1}$ while worst individual has the largest fitness value $F_p^{k=\beta}$

Save best solution of this generation $iter_{best}$ in iteration list $Iter\ list$

// **(2.2) evaluation of similarity for the population**

For $k = 2$ to β do:

calculate Euclidean similarity S^k between best individual and individual k

end

Sum all the S^k and save it as $iter_{similarity}$ in similarity list $Similarity\ list$

// **(3) checking stopping conditions**

If stopping conditions meet:

output X minimum of best solution in all iterations

else:

continue next step

// **(4) selection of parents eligible to produce offspring and extinction of ineligible individuals**

Set best individual as best parent B_{parent}

Delete the worst individual from Pop

Randomly choose another parent as R_{parent} from Pop

// **(5) crossover operation**

If crossover type C_t = "Probabilistic Crossover":

do Probabilistic Crossover (crossover rate δ, B_{parent}, R_{parent})

end

If crossover type C_t = "Single point Crossover":

do Single point Crossover (crossover rate δ, B_{parent}, R_{parent})

end

If crossover type C_t = "Linear combination Crossover":

do Linear combination Crossover (crossover rate δ, B_{parent}, R_{parent})

end

// **(6) mutation operation**

If mutation type M_t = "Uniform mutation":

do Uniform mutation (mutation rate γ,search radius R, crossover result, domain of F is [-F_{low},-F_{upper}])

end

If mutation type M_t = "Normal mutation":

do Normal mutation (mutation rate γ,search radius R, crossover result, domain of F is [-F_{low},-F_{upper}])

end

// **(7) insertion of new individuals into the population**

Calculate the fitness value for this new individual F_p^{new} based on its phenotype

Insert F_p^{new} in fitness list and insert this new individual in Pop without breaking the ascending order

$iter = iter + 1$

Save best solution of this generation $iter_{best}$ in iteration list $Iter\ list$

Repeat (2.2) (3) (4) (5) (6) (7)

Crossover and Mutation operators of SSGA

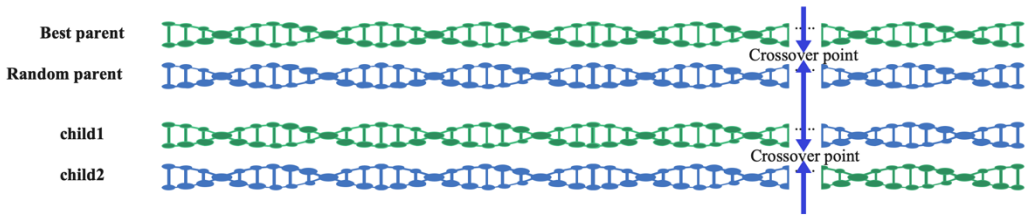
Three kinds of crossover operators are implemented in SSGA in this paper.

A. Single point Crossover

Given crossover rate δ, first generate a random probability σ, if $\sigma < \delta$, then begin the procedure of crossover operation, otherwise randomly choose one of the two parents as the result of crossover.

Figure 4 illustrates the single point crossover process, with the best parent in green and the random parent in red. The process of single point crossover is to select a random crossover point and then swap the best parent and the random parent for all the genes following that point.

Figure 4 single point crossover



Given crossover rate δ, best parent B_{parent}, random parentR_{parent}, and number of fitness function's dimensions N, generate a positive integer k ($k < N$):

$$C^i_{child1} = \begin{cases} B^i_{parent}, & i \le k \\ R^i_{parent}, & i > k \end{cases}$$
$$C^i_{child2} = \begin{cases} R^i_{parent}, & i \le k \\ B^i_{parent}, & i > k \end{cases}$$

Where $i \in 1,2,3,\ldots N$, i represents one location of a phenotype. Randomly select child1 or child2 to participate in the mutation operation afterwards.

B. Probability Crossover

Given crossover rate δ, best parent B_{parent}, random parentR_{parent}, and number of fitness function's dimensions N, for each gene of child, the probability of getting the best parent's gene is the same as crossover rate δ while the probability of inheriting a gene from a random parent is 1-δ. P represents probability.

$$P(C^i_{child} = B^i_{parent}) = \delta$$
$$P(C^i_{child} = R^i_{parent}) = 1 - \delta$$

Where $i \in 1,2,3,\ldots N$, i represents one location of a phenotype.

C. Linear combination Crossover

Given crossover rate δ, best parent B_{parent}, random parentR_{parent}, and number of fitness function's dimensions N, the child always inherits the characteristics of two parents. i represents one location of a phenotype.

$$C^i_{child} = B^i_{parent} * \delta + R^i_{parent} * (1 - \delta) \ \ where \ \ i \in 1,2,3,\ldots N.$$

Two kinds of mutation operators are implemented as following:

A. Uniform mutation

The probability density function of the continuous uniform distribution is:

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

The lower boundary and upper boundary for variable x is a and b. More specifically, $-a = b = 3 * R * (F_{upper} - F_{low})$ where F_{upper} and F_{low} represent the upper and lower bounds of the fitness function respectively.

Given mutation rate γ and result of crossover, for each value in the result of crossover, generate a random probability σ, if $\sigma < \delta$, then plus a variable x generated by uniform distribution, otherwise remain the same.

B. Normal mutation

The probability density function of the continuous normal distribution is:

$$f(x) = \frac{1}{\sigma\sqrt{2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

x = value of the variable

$\mu =$ the mean = 0

$\sigma =$ the standard deviation = $R * (F_{upper} - F_{low})$

Given mutation rate γ and result of crossover, for each value in the result of crossover, generate a random probability σ, if $\sigma < \delta$, then plus a variable x generated by normal distribution, otherwise remain the same.

Baldwin algorithm and Lamarck algorithm

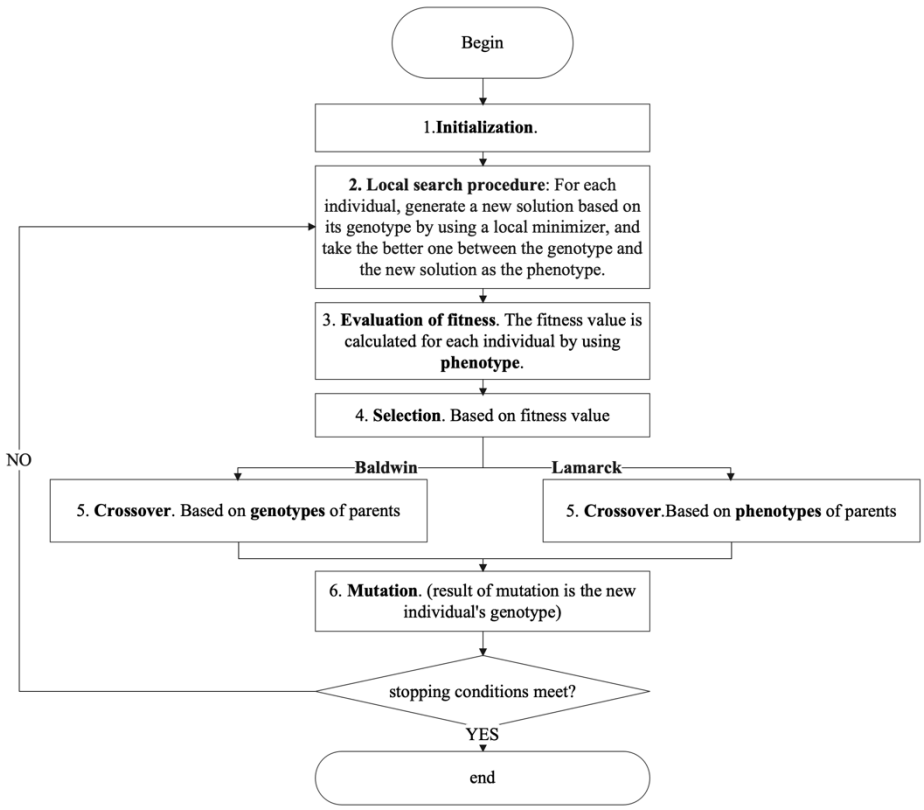
The difference between the Baldwin and Lamarck algorithm and SSGA is that they are combinations of memetic optimization and genetic optimization while SSGA is considered as global optimization. Compared to the SSGA, a local search procedure is additionally employed in the framework of the Baldwin and Lamarck algorithm. As mentioned before, each individual's genotype is a solution to an objective function. The local search procedure is designed to explore the neighborhood environment where genotype is located and try to find a better solution, which is defined as phenotype in this paper. The improvement between genotype and phenotype can be interpreted as the result of a lifelong learning effort.

In Baldwin approach, the learned results of an individual are not directly passed on to the next generation, but rather the ability to find a better solution in their own vicinity. In other words, although it is phenotypes that are involved in fitness evaluation, what is involved in the computation of crossover and mutation are genotypes.

In Lamarck approach, individuals' learning also influence the mapping of genotype and phenotype, which is consistent with the Baldwin approach. From an individual's perspective, learning will put one in a more advantageous position on the fitness landscape. More importantly, the results of individual learning can be passed on directly to future generations as a legacy. In other words, the phenotype is not limited to fitness evaluation, but is also engaged in subsequent crossover and mutation operations.

Figure 5 presents a flow chart of the Baldwin and Lamarck algorithm. As one can see, there is an additional local search procedure compared to SSGA. The difference between the Baldwin and Lamarck algorithms is whether the parents' genotypes or phenotypes are used to produce offspring. They are the same as SSGA except for the additional local search procedure and the ingredients of crossover operation.

Figure 5 Flowchart for Baldwin and Lamarck



The following are the parameters of the Baldwin and Lamarck algorithm.

Parameters for Baldwin and Lamarck algorithm

Input: parameter combination

1. max iterations Max_{iter}
2. convergence tolerance Tol and N
3. fitness function F ,number of dimensions of F is D, domain of F is [-F_{low},-F_{upper}]
4. mutation rate γ , crossover rate δ
5. mutation type M_t , crossover type C_t
6. number of individuals β
7. search radius R
8. global minima opt and threshold θ
9. local search rate α
10. local search type LS_t
11. number of local search solutions W
12. mode $Baldwin$ or $Lamarck$

Output: best solution X

The following **Pseudocode2** shows pseudo-code of Baldwin and Lamarck algorithm. Compared to SSGA, Baldwin and Lamarck algorithm algorithms have an additional local search procedure and a slight difference in the crossover operation, but the rest is the same as SSGA, so no repeat explanation will be given here.

PseudoCode2

(1) initialization of the population (number of individuals β, number of dimensions of F, domain of F is [F_{low},-F_{upper}])

Save all the individuals in the population Pop

(2) local search procedure (local search rate α, local search type LS_t, number of local search solutions W, population Pop, fitness function F, domain of F is [-F_{low},-F_{upper}], search radius R)

For $k = 1$ to β do:

Get genotype g of individual k in population Pop

For $w = 1$ to W do:

For i $= 1$ to D do:

Generate a random probability σ

if σ > local search rate α:
gi remains the same
Continue to next dimension
else:
if local search type LS_t = "uniform":
$gi = gi + x$ (x is generated by uniform distribution [-3*R*(F_{upper}-F_{low}]), 3*R*(F_{upper}-F_{low}]))
$gi = min(gi, F_{upper})$
$gi = max(gi, F_{low})$
end
if local search type LS_t = "normal":
$gi = gi + x$ (x is generated by normal distribution [MEAN = 0, STD =R*(F_{upper}-F_{low}]))
$gi = min(gi, F_{upper})$
$gi = \max(gi, F_{low})$
end
end
end
save one new solution to W list
end
calculate fitness values using fitness function F for genotype and W new solutions
choose the best one as the phenotype of individual k
end
(3) evaluation of the fitness value and similarity for the population **(**population Pop, fitness function F)
Based on individuals' phenotypes.
(4) checking stopping conditions (max iterations Max_{iter}, tolerance Tol and N, global minima opt and threshold θ)
(5) selection
(6) crossover operation (crossover rate δ, crossover type C_t, mode $Baldwin$ or $Lamarck$)
If mode = "Baldwin":
Choose parents' genotypes to do crossover operation
end
If mode = "Lamarck":
Choose parents' phenotypes to do crossover operation
end
(7) mutation operation (crossover results, mutation rate γ, mutation type M_t, search radius R, domain $[-F_{low}, -F_{upper}]$)
(8) go to step (2) and generate a phenotype for new individual then begin next iteration

---

Local search procedure

Roughly speaking, the local search procedure is implemented in the same way as mutation, i.e., a random number generated by uniform distribution or normal distribution is added to a dimension with a certain probability.
The main differences between the two are as follows:

1. The two occur in different stages. The mutation operation comes after the crossover operation and local search procedure follows the mutation operation.
2. The two produce different numbers of new solutions. Mutation operation will produce only one new solution, which will serve as the genotype of the new individual, whether or not it is better. However, the local search procedure can generate many new solutions, from which the best one is selected and compared with the genotype, and the better one of the best one of new solutions and the genotype is eventually determined as the phenotype of the new individual.
3. The mutation rate and the local search rate can also be different.

Results

Best 20 parameter combinations

We find the best 20 parameter combinations by using grid search. First, we propose 200 parameter combinations with different values in iterations, mutation rate, crossover rate, number of individuals, and search radius R. Considering that we have 3 crossover types and 2 mutation types, thus we have 6 combinations of mutation type and crossover type. Then we run 200 parameter combinations for each combination of mutation type and crossover type using SSGA. So, in total we end up with 1200 parameter combinations.

Suppose that σ is the index of a parameter combination, σ belongs to $[1,2,3,\ldots,1200]$.. Fi represents a function where i can be $[1,3,6,12,18,22]$. γ_σ^{Fi} represents the final outcome produced by a specific function and a specific parameter combination. $Rank_\sigma^{Fi}$ represents the ranking for γ_σ^{Fi}. For example, if γ_1^{F1} produce the smallest solution among all γ_σ^{F1}, then γ_1^{F1} ranked 1st and $Rank_\sigma^{Fi}$=1.

$score_\sigma$ represents the performance of a parameter combination for 6 functions.

$$score_\sigma = \sum_{\sigma=1}^{1200} (Rank_\sigma^{F1} + Rank_\sigma^{F3} + Rank_\sigma^{F6} + Rank_\sigma^{F12} + Rank_\sigma^{F18} + Rank_\sigma^{F22})$$

Considering the time cost, we only used 6 functions. The smaller the score is, the better. Table 2 shows the best parameter combinations for SSGA. The letters in the table 2 header represent, in order, index of a parameter combination σ, max iterations Max_{iter}, mutation rate γ, number of individuals β, crossover rate δ, mutation type M_t, crossover type C_t, search radius R, convergence tolerance Tol in continuous N iterations, the threshold θ for finding global optimal value.

Table 2 best 20 parameter combinations for SSGA

σ	Max_{iter}	γ	β	δ	M_t	C_t	R	Tol	N	θ
590	1000000	2/dim	100	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
593	1000000	2/dim	100	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
579	1000000	1/dim	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
588	1000000	2/dim	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
589	1000000	2/dim	100	0.7	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
591	1000000	2/dim	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
576	1000000	1/dim	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
587	1000000	2/dim	100	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
569	1000000	1/dim	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
558	1000000	1/dim	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
592	1000000	2/dim	100	0.7	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
572	1000000	1/dim	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
594	1000000	2/dim	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
542	1000000	0.5/dim	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
570	1000000	1/dim	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
562	1000000	1/dim	100	0.7	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
575	1000000	1/dim	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
573	1000000	1/dim	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
552	1000000	1/dim	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
555	1000000	1/dim	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001

The parameter about local search procedure for Lamarck and Baldwin are proposed in table 3, other parameters are the same as table 2. The letters in the table 3 header represent, in order, index of a parameter combination σ, local search rate α, local search type LS_t, number of local search solutions W, *mode* $Baldwin\ or\ Lamarck$.

Table 3 parameters for local search procedure

σ	α	LS_t	W	Mode	σ	α	LS_t	W	Mode
590	0.5	Uniform	1	Baldwin/Lamarck	592	0.5	Uniform	1	Baldwin/Lamarck
593	0.5	Uniform	1	Baldwin/Lamarck	572	0.5	Uniform	1	Baldwin/Lamarck

579	0.5	Uniform	1	Baldwin/Lamarck	594	0.5	Uniform	1	Baldwin/Lamarck
588	0.5	Uniform	1	Baldwin/Lamarck	542	0.5	Uniform	1	Baldwin/Lamarck
589	0.5	Uniform	1	Baldwin/Lamarck	570	0.5	Uniform	1	Baldwin/Lamarck
591	0.5	Uniform	1	Baldwin/Lamarck	562	0.5	Uniform	1	Baldwin/Lamarck
576	0.5	Uniform	1	Baldwin/Lamarck	575	0.5	Uniform	1	Baldwin/Lamarck
587	0.5	Uniform	1	Baldwin/Lamarck	573	0.5	Uniform	1	Baldwin/Lamarck
569	0.5	Uniform	1	Baldwin/Lamarck	552	0.5	Uniform	1	Baldwin/Lamarck
558	0.5	Uniform	1	Baldwin/Lamarck	555	0.5	Uniform	1	Baldwin/Lamarck

Results of Experiments

Percentage of finding the global optimal value

We ran 10 times for a parameter combination and a function. The percentage value for a parameter combination and a function is calculated as (number of times finding the global optimal value of the function /10). We have 20 parameter combinations, so we have 20 percentage values for each function.

Figure 6 has three sub-pictures from top to bottom showing the bar plots distinguished by each function for percentage values produced by 20 parameter combinations for the SSGA, Baldwin and Lamarck algorithms respectively. Each bar in Figure 6 is generated by 20 data points. The number on the blue rectangle is the height of the blue rectangle, also known as mean probability. The mean probability is the mean of the percentage values produced by 20 parameter combinations. The black vertical lines that dive into the blue rectangle is called error bars. The length of the error bars in Figure 6 represents that confidence interval = 95%.

Confidence Interval Error Bars

Error bars usually consist of four categories. First one is range which is amount of spread between the extremes of the data (i.e., the maximum value minus the minimum value). Second is standard deviation (SD), which is calculated as the square root of the variance. SD can be used to measure the stability of whole data points. Third one is standard error (SE), which is calculated as (SD/number of data points). The fourth one is confidence interval, which is calculated as $\mu \pm Z\frac{s}{\sqrt{n}}$, wherein μ is mean of all data points, Z is chosen Z-value, Z is number of data points and s SD. With a 95% confidence interval, the chance to be wrong is 5%. [28] If the 95% confidence intervals of two different groups do not overlap, then they are considered statistically significantly different from each other. Conversely, no significant difference between the two groups [29-31]. But the absence of significant differences does not mean that no differences exist [32].

Analysis

First, we can have a look at SSGA subpicture in Figure 6. The mean probabilities for SSGA finding global optimal values of F1, F2, F3, F4, F6, F7, F9, F10, F11, F12, F13, F14, F16, F17 and F18 is 100%. However, SSGA fails to find the minimum values of F5, F8 and F19 at all. For F15, F20, F21, F22 and F23, the mean probability of SSGA finding the global minimum of these five functions is 13%, 47.5%, 31.5%, 48.5%, 51.5%.

Second, in the middle subpicture showing Baldwin algorithm's results, one can see that the mean probabilities regarding F1, F2, F3, F4, F6, F7, F9, F10, F11, F12, F13, F14, F16, F17, F5, F8 and F19 are the same as SSGA. For F15, F20, F21, F22 and F23, the mean probability of Baldwin finding the global minimum for these five functions is 15.5%, 40.5%, 53.5%, 77.5%, 84.5%.

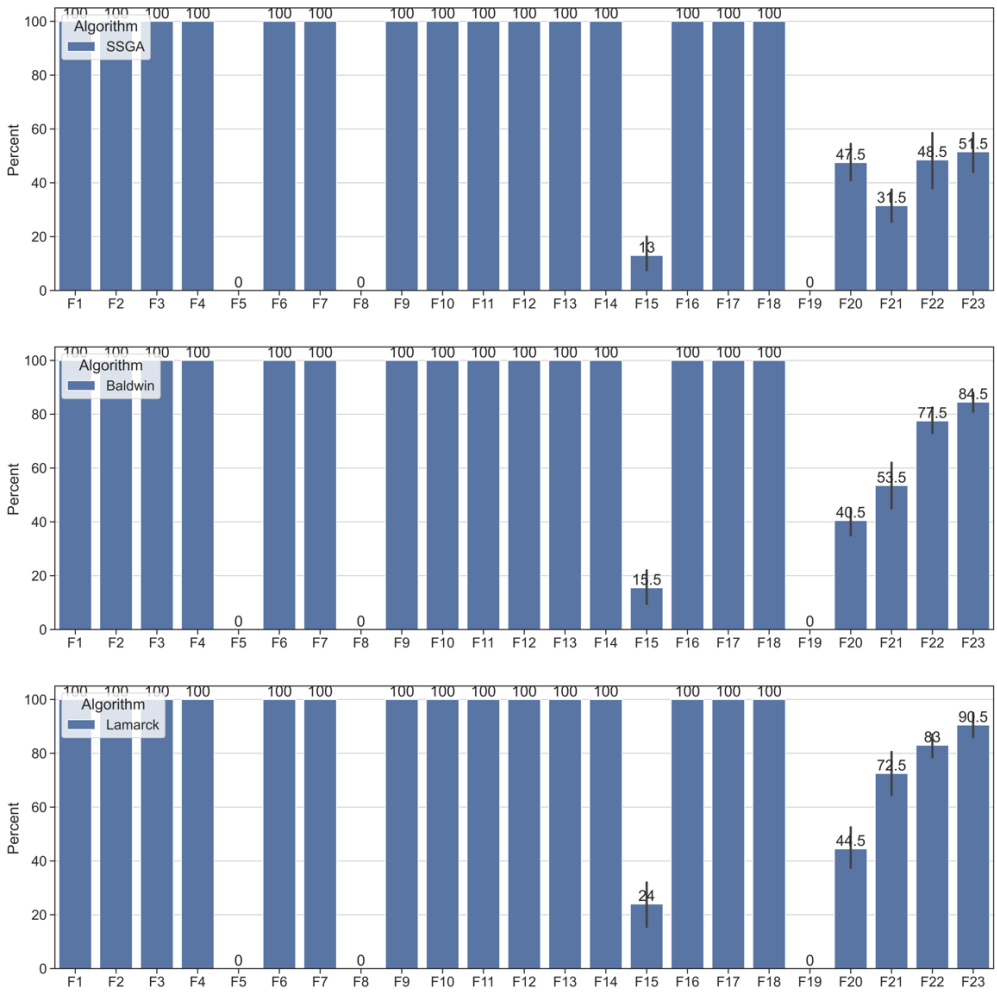
It is essential to note here that obviously the error bars of Baldwin and the error bars of SSGA do not overlap for F21, F22 and F23, which means that their performance in these three functions is significantly different. Or in other words, compared to SSGA, Baldwin is significantly better on F21, F22 and F23.

The error bars of Baldwin and the error bars of SSGA do overlap on F15 and F20, the performances for Baldwin and SSGA are not significant different, but they remain distinct. The mean probability of Baldwin finding the global optimal values of F15 is 2.5% more than in SSGA while the mean probability of Baldwin finding the global optimal values of F20 is 7% less than in SSGA. Considering that we only did 10 runs for each combination of parameters, these probabilities are only approximations of the true probability. It is highly possible that Baldwin and SSGA are comparable in terms of performance of these two functions.

Third, in the bottom subpicture showing Lamarck algorithm's results, one can see that the mean probabilities regarding F1, F2, F3, F4, F6, F7, F9, F10, F11, F12, F13, F14, F16, F17, F5, F8 and F19 are the also same as SSGA. For F15, F20, F21, F22 and F23, the mean probabilities of Lamarck finding the global minimum are 23%, 44.5%, 72.5%, 83%, 90.5%, respectively. Compared to mean percentage values of SSGA with 13%, 47.5%, 31.5%, 48.5%, 51.5% on these five functions, Lamarck's algorithm has nearly doubled the percentage except for F20. The difference between the Lamarck algorithm and SSGA is only 3% in the performance of the F20, which is perfectly acceptable.

Based on the above analysis of Figure 6, one can conclude that Baldwin and Lamarck algorithms are remarkably better than SSGA in finding global minimum of multimodal functions with fixed dimensions like F21, F22 and F23. In terms of the performance of the other functions, the proportions between the three of them are almost equal.

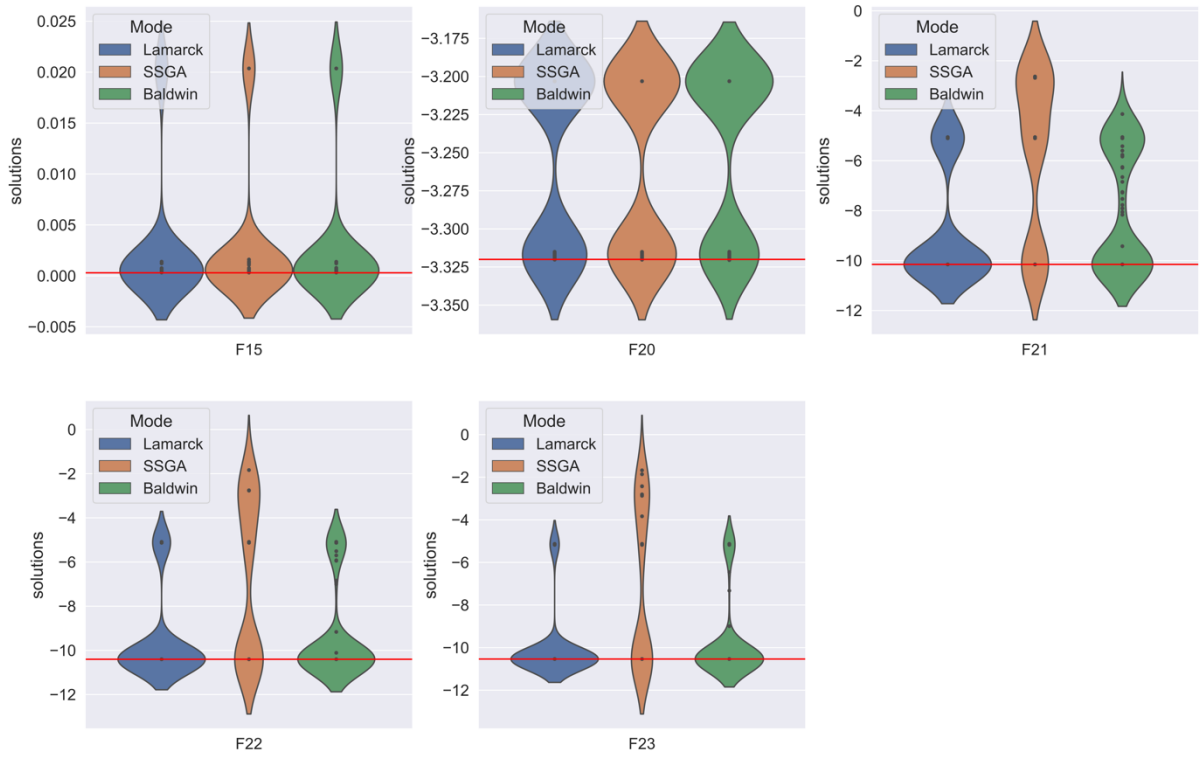
Figure 6 Bar plots for percentage values for SSGA, Baldwin and Lamarck algorithms



Improvements between SSGA and Baldwin, Lamarck algorithms

Violin plots

Figure 7 violin plots for comparing solutions of SSGA, Baldwin, and Lamarck algorithms



A violin plot is designed to provide a visual representation of the distribution of numerical data by using a kernel density estimator, which shows the high peaks in the data. The wider areas of the violin plot represent the higher probability, where many data points are gathered. The wider the area is, the more data points there are. On the contrary, the thinner areas represent the lower probability and less data concentrated here.

Figure 7 shows the violin plots for SSGA, Baldwin and Lamarck algorithms on F15, F20, F21, F22 and F23, respectively. Since all three perform identically on other functions, we will primarily focus on the five functions. Each subplot has a red horizontal line running through it, which represents the position of the global minimum of a function. We have 20 parameter combinations. Each parameter combination was run 10 times for each function. This means that each violin is generated from 200 solutions. The black dots represent the actual solutions. The reason for not seeing so many points on Figure 7 is that many of the solutions are the same.

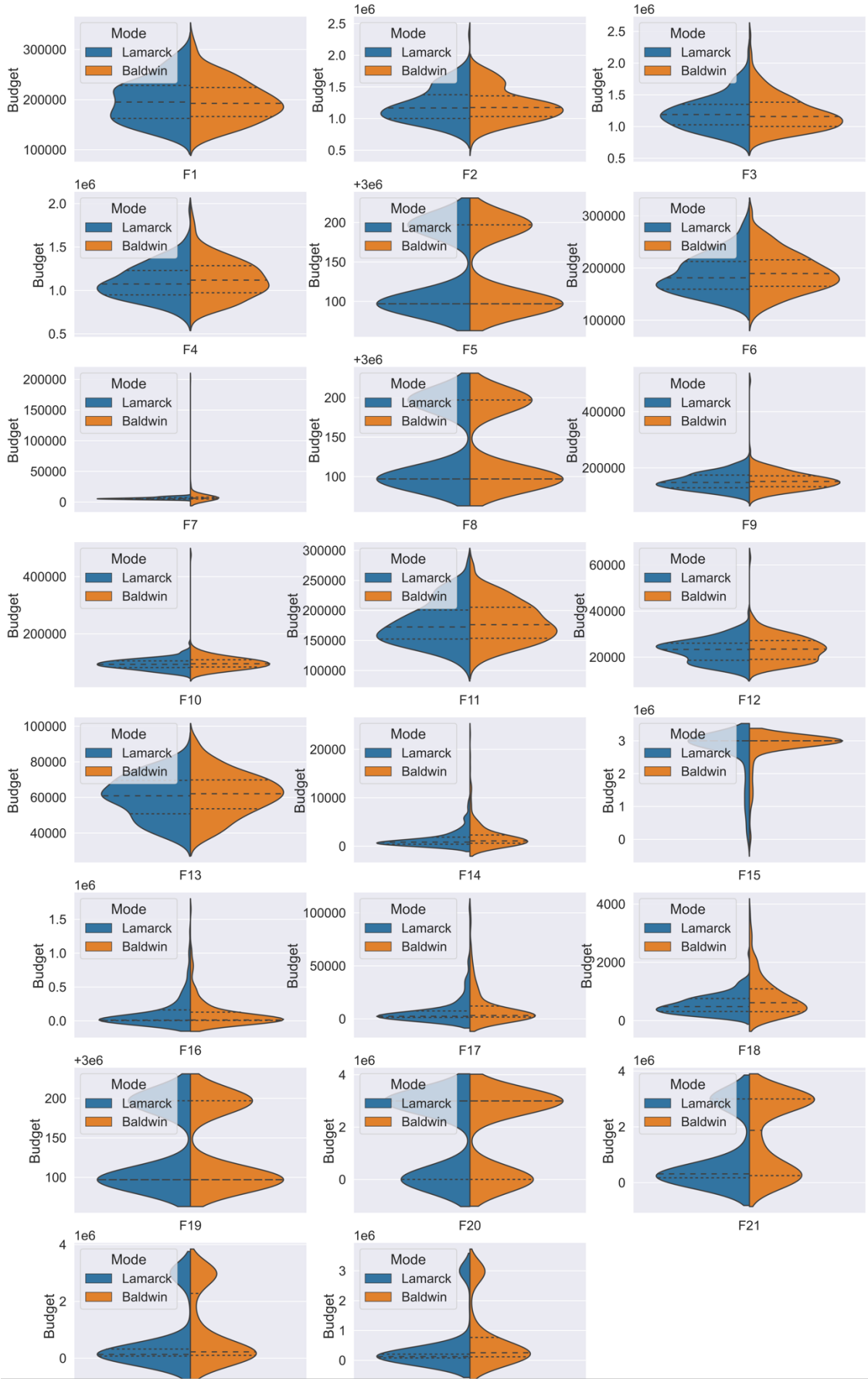
Analysis

For F15 and F20, it can be seen that the SSGA, Baldwin and Lamarck algorithms perform almost in equal measure. For F21, F22 and F23, the data points for both the violins of Lamarck and Baldwin algorithms are highly concentrated, and the solutions found by these two algorithms are smaller than those found by SSGA.

Comparison between Baldwin and Lamarck algorithm

Figure 8 illustrates the violin plots for Baldwin and Lamarck algorithms for each function, using budget data. Budget in this case means the number of times a fitness function is called throughout the process of generating a solution for a function using a parameter combination. It can be seen that on F21, F22 and F23, the budget used by Lamarck's algorithm is significantly lower than that required by Baldwin's algorithm, and virtually no gap is found in the other functions.

Figure 8 violin plots for comparing budgets for Baldwin and Lamarck algorithms



Conclusion

Based on the Steady-state genetic algorithm (SSGA), this paper extends genetic search with memetic optimization. We offer two valuable versions of memetic optimization: one motivated by the Baldwinian effect, while the other by the Lamarckian theory of evolution, named after Baldwin algorithm and Lamarck algorithm, respectively. Compared to SSGA, Baldwin and Lamarck showed a significant advantage in finding the global minima of multimodal functions with fixed dimensions such as F21, F22 and F23. But the differences in the performance of the other functions are not obvious. Compared to Baldwin, Lamarck's algorithm can find the minimum value of a function using a lower budget. The performance of both Baldwin and Lamarck algorithms is evaluated and validated on benchmark functions CEC-BC-2017.

Reference

1. Wikipedia Contributors (2019). *Genetic algorithm*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Genetic_algorithm.
2. Katoch, S., Chauhan, S.S. and Kumar, V. (2020). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*. doi:10.1007/s11042-020-10139-6.
3. Mirjalili, S. (2019). Genetic Algorithm. In: Evolutionary Algorithms and Neural Networks. Studies in Computational Intelligence, vol 780. Springer, Cham. https://doi.org/10.1007/978-3-319-93025-1_4
4. Collins, R. J., & Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms (pp. 249–256). University of California (Los Angeles), Computer Science Department.
5. Ishibuchi, H., & Yamamoto, T. (2004). Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining. *Fuzzy Sets and Systems*, *141*(1), 59–88.
6. Hutter, M. (2002). Fitness uniform selection to preserve genetic diversity. In Proceedings of the 2002 Congress on Evolutionary Computation, CEC'02 (Vol. 1, pp. 783–788). IEEE.
7. Grefenstette, J. J. (1989). How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the 3rd International Joint Conference on Genetic Algorithms (ICGA89)*.
8. Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9). Morgan Kaufmann Publishers.
9. Semenkin, E., &Semenkina, M. (2012). Self-configuring genetic algorithm with modified uniform crossover operator. In International Conference in Swarm Intelligence (pp. 414-421). Heidelberg: Springer.
10. Hu, X. B., & Di Paolo, E. (2007). An efficient genetic algorithm with uniform crossover for the multi-objective airport gate assignment problem. In IEEE Congress on Evolutionary Computation, 2007 (CEC 2007) (pp. 55-62). IEEE.
11. Tsutsui, S., Yamamura, M., & Higuchi, T. (1999). Multi-parent recombination with simplex crossover in real coded genetic algorithms. In Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1 (pp. 657-664). Morgan Kaufmann Publishers Inc.
12. Bck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (2000). Evolutionary computation 1: Basic algorithms and operators (Vol. 1). CRC press.
13. Oliver, I. M., Smith, D., & Holland, J. R. (1987). Study of permutation crossover operators on the travelling salesman problem. In Proceedings of the Second International Conference on Genetic Algorithms and their Applications, July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA. Hillsdale, NJ: L. Erlhaum Associates.
14. Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In IJCAI (Vol. 85, pp. 162-164).
15. Whitley, D., Timothy, S., & Daniel, S. Schedule optimization using genetic algorithms. In D. Lawrence (Ed.) 351-357.
16. Hinterding, R. (1995). Gaussian mutation and self-adaption for numeric genetic algorithms. In *IEEE International Conference on Evolutionary Computation* (Vol. 1, p. 384). IEEE.
17. Tsutsui, S., & Fujimoto, Y. (1993). Forking genetic algorithm with blocking and shrinking modes (fGA). In *ICGA* (pp. 206–215).

18. Oosthuizen, G. D. (1987). Supergran: A connectionist approach to learning, integrating genetic algorithms and graph induction. In *Proceedings of the second International Conference on Genetic Algorithms and their Applications*, July 28–31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA. Hillsdale, NJ: L. Erlhaum Associates.
19. Mauldin, M. L. (1984). Maintaining diversity in genetic search. In *AAAI* (pp. 247–250).
20. Ankenbrandt, C. A. (1991). An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. In *Foundations of genetic algorithms* (Vol. 1, pp. 53–68). Elsevier.
21. Wikipedia Contributors (2019). Memetic algorithm. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Memetic_algorithm.
22. Radcliffe, Nicholas J., and Patrick D. Surry. "Formal Memetic Algorithms." Evolutionary Computing, 1994, pp. 1–16, 10.1007/3-540-58483-8_1. Accessed 22 Nov. 2022.
23. Dutta, Priyom & Mahanand, B.S.. (2022). Affordable energy-intensive routing using metaheuristics. 10.1016/B978-0-323-85117-6.00013-3.
24. García-Martínez, Carlos, and Manuel Lozano. "Local search based on genetic algorithms." Advances in metaheuristics for hard optimization. Springer, Berlin, Heidelberg, 2007. 199-221.
25. Wikimedia Foundation. (2022, November 21). *Genotype*. Wikipedia. Retrieved November 22, 2022, from https://en.wikipedia.org/wiki/Genotype
26. *Phenotype* (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Phenotype (Accessed: November 22, 2022).
27. https://doi.org/10.48550/arXiv.cs/0212036