ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

Comparing the Lamarckian and Baldwinian Approaches in Memetic Optimization

University:	Eötvös Loránd University, Hungary
Faculty:	ELTE Faculty of Informatics.
Advisor:	László Gulyás, Associate Professor
Email:	lgulyas@inf.elte.hu
Student Name:	Mei Jiaojiao, Master student
Neptun code:	V6FISC
Email:	v6fisc@inf.elte.hu
Date:	Budapest, 2022-12-04

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 the global search is implemented via the crossover operator and the 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 theories suggest that behaviors of individuals are not only passed on to offspring by crossover and mutation but also through lifetime learning. In Baldwinian approach, learned behaviors affect the mapping of genotypes to phenotypes, which ultimately results in changes to the fitness landscape. In Lamarckian 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 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.

Introduction

Everyone is driven to optimize, to obtain the best education, the most decent employment, the most desirable partner, and so forth. An optimization problem is a choice-making task of choosing the best solution among a limited number of options. Many real-life problems can be generally summarized as difficult optimization problems. Many of them are too complex and subject to many parameters, thus usually they cannot be solved by an analytic expression directly but can only be approximated computationally. Some well-known optimization algorithms in state of art include Line Search Methods (LSMs), Trust-Region Methods (TRMs), Conjugate Gradient Methods (CGMs), Quasi-Newton Methods (QNMs), Derivative-Free Optimization (DFO) and Evolutionary Algorithms (EAs) [1].

A general continuous optimization (minimization) problem can be defined as the following:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, \quad i \in \mathcal{E}, \\ c_i(x) &\geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

where $x \in \mathbb{R}^n$ is an N-dimensional vector ($n \geq 1$), $f: \mathbb{R}^n \to \mathbb{R}$ is an objective function that we want to optimize, $c_i(\cdot), i \in \mathcal{E}$ are called equality constraint functions, $c_i(\cdot), i \in \mathcal{I}$ are called inequality constraint functions [1]. If $f(x^*) \leq f(x)$ for all x, then x^* is a global minimum. If there exists a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$, then x^* is a local minimum [1].

Genetic algorithm and Memetic algorithm

Genetic algorithm

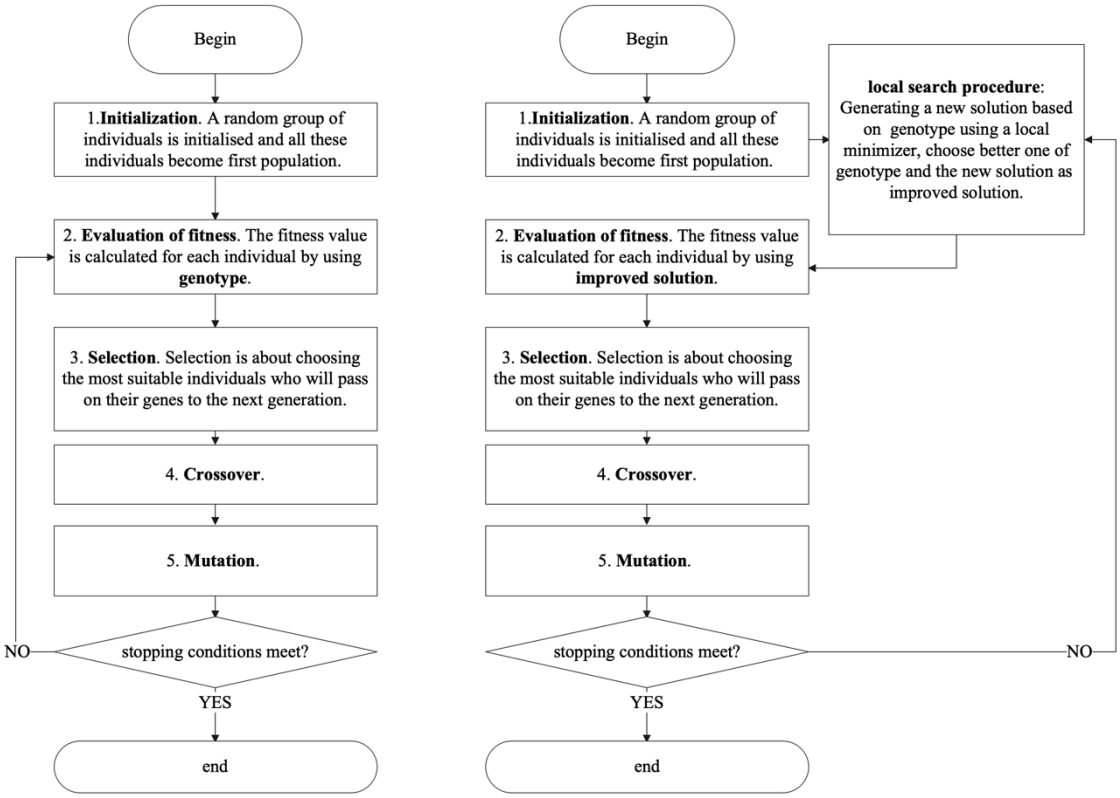
Genetic Algorithm (GA) has become one of the critical methods behind solvers capable of addressing large-scale real-world optimization problems. It has been positively explored in research institutes as well as being actively employed in the industry. GA is a population-based stochastic algorithm that originated from the Darwinian theory of evolution, belonging to the broader category of Evolutionary Algorithms. It consists of three principal bio-inspired operators: selection, crossover, and mutation [2-4]. Inspired by that only superior individuals in the population have the chance to produce offspring and pass on their genes, the 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, and steady state reproduction can be found in [5-9]. The crossover operator allows two parents to swap their genes with a certain probability, thereby producing a solution between two points [2-4]. A variety of ways to implement crossover operators such as uniform crossover, half uniform crossover, three parent's crossover, partially matched crossover, cycle crossover, and order crossover are introduced in the literature [10-15]. Different from the crossover, the 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 [2-4]. Some mutation operators such as Gaussian, shrink, supervised mutation, uniqueness mutation, and 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 his 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

applying the genotype into the fitness function. A Fitness landscape is a way to conceptualize the relative locations between genotypes and a fitness function. A fitness landscape is an (N+1)-dimensional view of a fitness function while a genotype is an N-dimensional point. The height of a fitness landscape represents the 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 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 gradually moves towards better populations, but it is not guaranteed to find the global optimum every time, depending on multiple factors such as the number of iterations and the complexity of the objective function.

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



Memetic algorithm

With the influence of R. Dawkins' notion of 'memes' [23], Pablo Moscato first introduced the memetic algorithm (MA) in 1989 and he proposed that a 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 with a better one if exists [29].

In searching for the optimal value of a fitness function, the crossover operator gives individuals the ability to jump across the landscape of a fitness (objective) function while the mutation operator enables individuals to explore the 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 his 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.

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. We expect that memetic algorithms will create collaborative effects that usefully bridge local and global search.

Baldwinian and Lamarckian evolution

Both the Baldwinian and Lamarckian evolution suggests that behaviors of individuals are not only passed on to offspring by crossover and mutation but also through lifetime learning. In the Baldwinian approach, learned behaviors affect the mapping of genotypes to phenotypes, which ultimately results in changes to the fitness landscape. This corresponds to the idea of memetic evolution. In the Lamarckian 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 [35-39].

Baldwinian and Lamarckian evolution are both memetic algorithms which focus on how an individual's lifetime learning affects the evolutionary direction [30-36]. Individuals can develop a phenotype based on their genotype through lifetime learning, and 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].

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 minimum of the fitness function, which means that a lower fitness value for an individual corresponds to a lower position on the fitness landscape. If the fitness value of the phenotype is lower, 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 in the fitness landscape.

The difference is that phenotype is not inherited to the offspring in Baldwinian approach, but it can be in Lamarckian approach. Both approaches have their strengths and weaknesses, and it is difficult to judge which is better than the other. Individual learning in the Baldwinian approach can increase the diversity of the population and the ability to explore the fitness landscape at a cost [40-41], while the Lamarckian approach potentially accelerates the speed of finding global minima in dynamically evolving environments but may be constrained to valleys on the fitness landscape [42-43]. Most of these papers have focused on genotype to phenotype mapping. That is, how the lifetime learning of individuals affects the evolution of

populations. A comprehensive assessment of the two approaches has not been provided yet. Bereta et al. compared Baldwin effect and Lamarckian evolution but only limited to the Euclidean Steiner tree problem [39].

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, named as Baldwinian algorithm and Lamarckian algorithm, respectively. Before implementing the two algorithms, we create a baseline using the Steady State Genetic Algorithm (SSGA) as global optimization. Then we develop the Baldwinian and Lamarckian 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 Baldwinian and Lamarckian algorithms. The second goal of this paper is to compare the computational performance of Baldwinian and Lamarckian algorithms by using benchmark functions CEC-BC-2017 [44]. This set includes unimodal functions, multimodal functions, hybrid functions and composition functions with high dimensions, etc., which can be used to universally assess proposed Baldwinian and Lamarckian algorithms. This is what makes this study unique.

Methods

Fitness functions

CEC-BC-2017 contains 23 benchmark functions with different characteristics and levels of difficulty, which can be used to evaluate the performance of new algorithms. Table 1 shows 23 test functions with details. The header of Table 1 includes function, dimension, domain, and optimal. The first column of the table defines the function, the second gives the dimensionality of the function, the third defines the domain and finally, the fourth lists the known optimal value of the function on the domain.

Figure 2 shows the first two-dimensional view of the fitness landscape for part of CEC-BC-2017 benchmark functions, considering the layout, a two-dimensional view of the fitness landscape for all functions is not shown. Figure 2 has 12 subpictures, and each subpicture is a fitness landscape view of a fitness function in two dimensions with a title that specifies the 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.

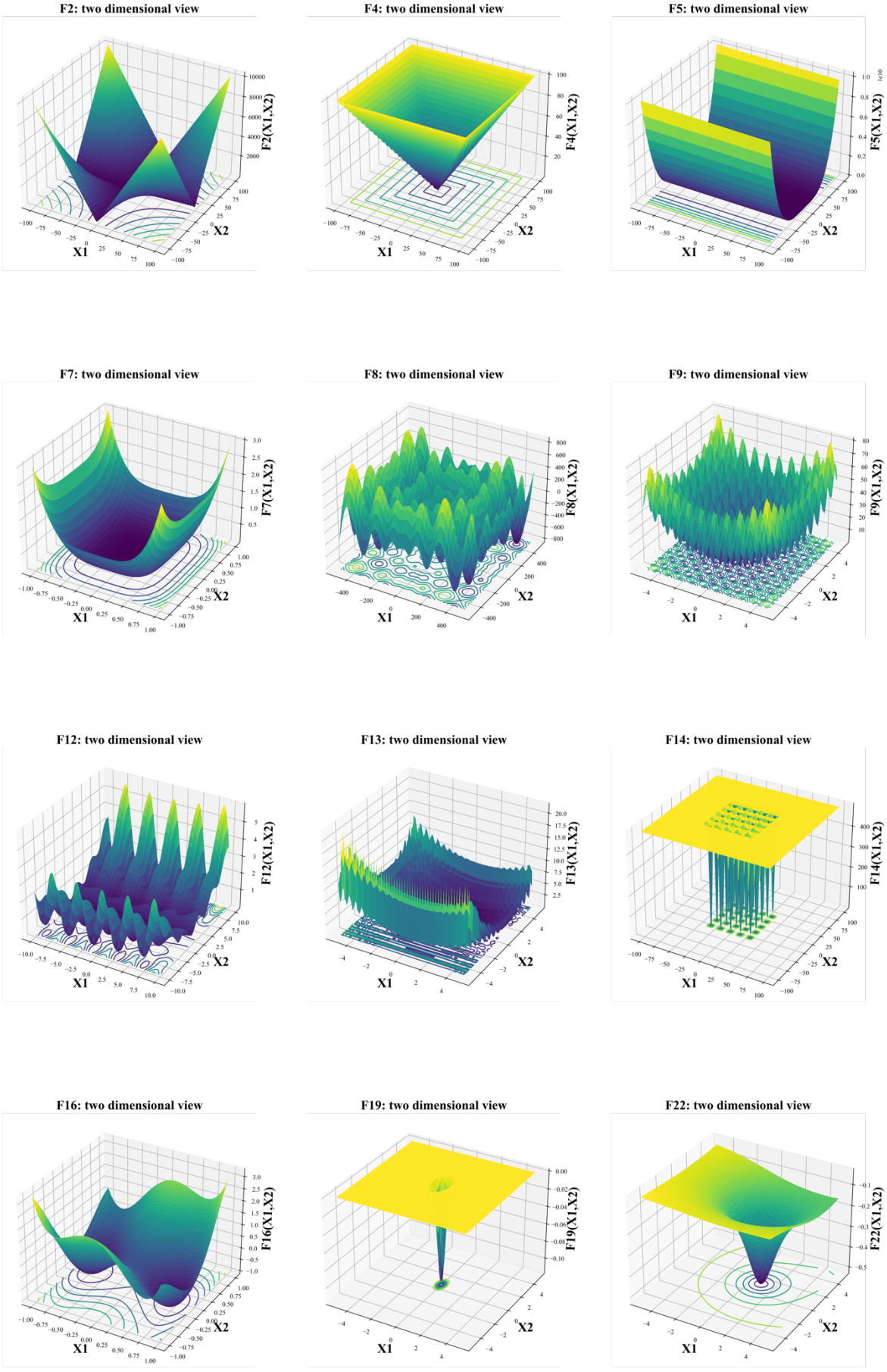
Table 1 CEC-BC-2017 benchmark functions

Function	Dim	Domain	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

$F7 = \sum_{i=1}^{50} (ix_i^4)$	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]	-418.98xd
$F9 = \sum_{i=1}^{50} (x_i^2 - 10\,cos(2\pi x_i) + 10)$	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
$F13 = 0.1\left(sin^2(3\pi x_1) + \sum_{i=1}^{49} (x_i - 1)^2(1 + sin^2(3\pi x_i + 1)) + (x_{50} - 1)^2(1 + sin^2(2\pi x_{50}))\right)$ $+ \sum_{i=1}^{50} u(x_i, 5,100,4)$	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], [-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]] $F14 = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}\right)^{-1}$	2	[-65,65]	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 $F15 = \sum_{i=1}^{11} \left(a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right)^2$	4	[-5, -5]	0.0003
$F16 = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	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]] $F19 = -\sum_{i=1}^{4}\left(c_i exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)\right.$	3	[0,1]	-3.86
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]] $F20 = -\sum_{i=1}^{4}\left(c_i exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)\right.$	6	[0,1]	-3.32
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] $F21 = -\sum_{i=1}^{5} \left((X - a_i)(X - a_i)^T + c_i\right)^{-1}$	4	[0,10]	-10.1532
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] $F22 = -\sum_{i=1}^{7} \left((X - a_i)(X - a_i)^T + c_i\right)^{-1}$	4	[0,10]	-10.4028
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] $F23 = -\sum_{i=1}^{10} \left((X - a_i)(X - a_i)^T + c_i\right)^{-1}$	4	[0,10]	-10.5363

$U(x, a, k, m) = k((x - a)^m)(x > a) + k((-x - a)^m)(x < (-a))$

Figure 2 two-dimensional view for part of test functions



Parameters

Table 2 and Table 3 show the parameters of the two types of algorithms used in this paper, respectively. A detailed description will be given in a later section.

Table 2 Parameters for SSGA

Input	Parameters for SSGA		Output
Parameters	Stopping criteria	• max iterations Max_{iter} • convergence tolerance Tol and N • global minimum opt and threshold θ	best solution X
	Fitness functions	• fitness function F • number of dimensions of F is D • domain of F is $[F_{low},F_{upper}]$	
	Generating offspring	• mutation rate γ • crossover rate δ • mutation type M_t • crossover type C_t	
	Search radius	• search radius R	
	Other parameters	• number of individuals β	

Table 3 Parameters for Baldwinian and Lamarckian algorithms

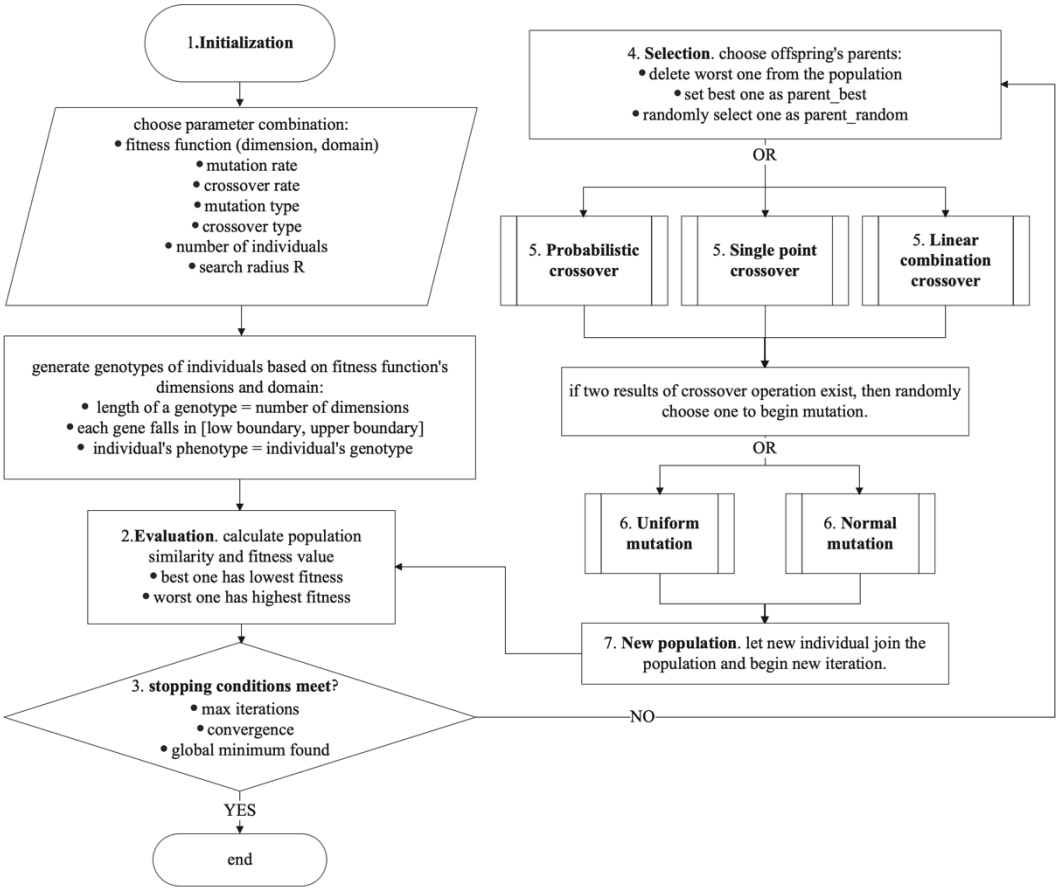
Input	Parameters for Baldwinian and Lamarckian algorithms		Output
Parameters	Stopping criteria	• max iterations Max_{iter} • convergence tolerance Tol and N • global minimum opt and threshold θ	best solution X
	Fitness functions	• fitness function F • number of dimensions of F is D • domain of F is $[F_{low},F_{upper}]$	
	Generating offspring	• mutation rate γ • crossover rate δ • mutation type M_t • crossover type C_t • **mode $Baldwin$ or $Lamarck$**	
	Search radius	• search radius R	
	Local search part	• **local search rate α** • **local search type LS_t** • **number of local search solutions W**	
	Other parameters	• number of individuals β	

Steady State Genetic Algorithm

A 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. The fitness value of an individual is computed based on his phenotype. The genotypes of the initial populations are randomly created with every gene located within the domain of a test function.

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 only one offspring and delete the least adapted individual; (5) crossover operation; (6) mutation operation; (7) insertion of new individuals into the population and begin execution of next iteration.

Figure 3 Flowchart of SSGA



Stopping criteria are 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 a certain number of continuous iterations. In this paper, the global optimum of the function is known in the program while the individuals are not aware of where the global optimum is. The program knows the global optimum and compares it with the fitness value of the best individual to determine whether to terminate the program.

Table 2 shows the parameters of the SSGA algorithm. The maximum number of iterations is Max_{iter}, we terminate the program when Max_{iter} is reached. Convergence tolerance Tol is another stopping condition. If the Euclidean similarity of the whole population is less than Tol in N continuous iterations, then we terminate the program. Alternatively, if the fitness value of the best individual minus global minimum opt is less than threshold θ, we terminate the program. In addition, search radius R is a parameter that controls the range of the mutation.

Pseudocode1 shows pseudo-code of SSGA.

PseudoCode1:SSGA

// **(1) initialization of the population based on the parameter combination**
Randomly generate β feasible genotypes of individuals based on F
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_g^k fitness value for an individual k using its genotype g
end
Sort all the F_g^k in ascending order and change the order of individuals' location in Pop accordingly
The best individual has the smallest fitness value $F_g^{k=1}$ while the worst individual has the largest fitness value $F_g^{k=\beta}$
Save the 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 are met:
output X minimum of the best solution in all iterations
else:
continue next step
// **(4) selection of parents eligible to produce only one offspring and delete the least adapted individual**
Set the 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_g^{new} based on its genotype
Insert F_g^{new} in the fitness list and insert this new individual in Pop without breaking the ascending order
$iter = iter + 1$
Save the 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 this paper.

Single point Crossover

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

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

$$C_{child1}^{i} = \begin{cases} B_{parent}^{i}, & i \leq k \\ R_{parent}^{i}, & i > k \end{cases}$$
$$C_{child2}^{i} = \begin{cases} R_{parent}^{i}, & i \leq k \\ B_{parent}^{i}, & i > k \end{cases}$$

Where $i \in 1,2,3,\ldots N$, i represents one dimension of a vector. This vector may be a genotype or a phenotype. At last, we randomly select child1 or child2 to participate in the mutation operation afterwards.

Probability Crossover

Given crossover rate δ, best parent B_{parent}, random parent R_{parent}, and the number of fitness function's dimensions N, for each gene of a 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.

$$P(C_{child}^{i} = B_{parent}^{i}) = \delta$$
$$P(C_{child}^{i} = R_{parent}^{i}) = 1 - \delta$$

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

Linear combination Crossover

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

$$C_{child}^{i} = B_{parent}^{i} * \delta + R_{parent}^{i} * (1 - \delta) \ \textit{where} \ \ i \in 1,2,3,\ldots N.$$

Two kinds of mutation operators are implemented.

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, μ = mean = 0, η = standard deviation = $R * \left(F_{upper} - F_{low}\right)$. Given the mutation rate γ and the result of crossover. For a gene g in the result of crossover, generate a random probability σ, if $\sigma < \gamma$, then plus a variable x generated by a normal distribution, otherwise remain the same.

$$g = \begin{cases} g + x(\mu, \eta) & \textit{for} \ \sigma < \gamma \\ g & \textit{for} \ \sigma \geq \gamma \end{cases}$$

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 are a and b. More specifically, $\text{abs}|a| = b = 3 * \sigma = 3 * R * (F_{upper} - F_{low})$ 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 a gene g in the result of crossover, generate a random probability σ, if $\sigma < \gamma$, then plus a variable x generated by uniform distribution, otherwise remain the same.

$$g = \begin{cases} g + x(a,b) & for\ \sigma < \gamma \\ g & for\ \sigma \ge \gamma \end{cases}$$

The reason why $\text{abs}|a| = b = 3 * R * (F_{upper} - F_{low})$ in uniform mutation is to ensure that the range of the uniform mutation is the same as the range of the normal mutation. Thus, the values of the variable x will fall within 6 times $R * (F_{upper} - F_{low})$.

Baldwinian algorithm and Lamarckian algorithm

The difference between the Baldwinian and Lamarckian algorithms and SSGA is that they are memetic optimizations, while SSGA is considered as genetic optimization. Compared to the SSGA, a local search procedure is additionally employed in the framework both of the Baldwinian and Lamarckian algorithms. 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 the genotype is located and try to find a better solution, which is defined as a phenotype in this paper. The improvement between genotype and phenotype can be biologically motivated as the result of a lifelong learning effort.

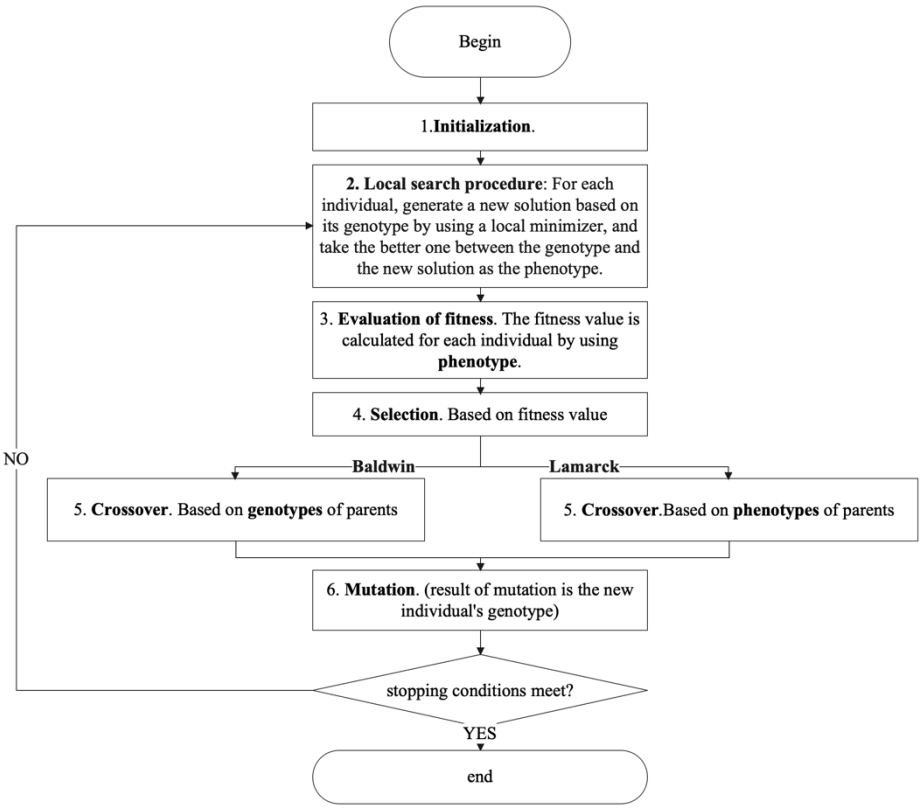
In the Baldwinian approach, the learned results of an individual are not directly passed on to the next generation, but rather the ability to find a better solution, this increases the individual's chance to reproduce. In other words, although it is the phenotype that is involved in fitness evaluation, only the genotype takes part in the computation of crossover and mutation.

In the Lamarckian approach, individuals' learning also influences the mapping of genotype and phenotype, which is similar to the Baldwinian approach. Learning will put one in a more advantageous position. However, 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 4 presents a flow chart of the Baldwinian and Lamarckian algorithms. As one can see, there is an additional local search procedure compared to SSGA. The difference between the Baldwinian and Lamarckian 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.

Table 3 shows parameters for Baldwinian and Lamarckian algorithms.

Figure 4 Flowchart for Baldwinian and Lamarckian algorithms



Local search procedure

Given local search rate α, local search type LS_t , the number of local search solutions W, and the genotype of a new individual, for each gene i in the genotype G of the new individual, generate a random probability σ, the corresponding value in new solution NS is:

$$NS_i = \begin{cases} G_i + x & for\ \sigma\ <\ \alpha \\ G_i & for\ \sigma\ \geq\ \alpha \end{cases}$$

Local search type LS_t will determine how the random variable x is generated. If Local search type LS_t is uniform, then variable x is sampled from a uniform distribution. If Local search type LS_t is normal, then variable x is from a normal distribution. The range for uniform distribution and normal distribution is introduced in mutation operators already. The number of local search solutions W determines how many new solutions are produced for each individual using a local search procedure.

Comparison between Mutation and Local search procedure

In the experiments discussed in this paper, the local search procedure is implemented in the same way as mutation. That is, a random sample of the genotype's neighborhood is created using the uniform or normal distribution, which means that the Lamarckian and Baldwinian algorithm has more information about the neighborhood of the genotype.

The main differences between mutation and this version of the local search are as follows:

1. The two occur in different stages. The mutation operation comes after the crossover operation while the local search procedure follows the mutation operation from the second iteration.

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 only the best one will be selected as the individual's new phenotype, provided that its fitness is better than that of the genotype.
3. The mutation rate and the local search rate can also be different.

Pseudocode2 shows the pseudo-code of Baldwinian and Lamarckian algorithms. Compared to SSGA, Baldwinian and Lamarckian algorithm algorithms have an additional local search procedure. The phenotype is passed on as the input to crossover in the case of the Lamarckian version, but the rest is the same as SSGA, so no repeated explanation will be given here.

PseudoCode2: Baldwinian and Lamarckian algorithms

(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:
                    if local search type LS_t = "uniform":
                        $NSi = Gi + x$
                        Checking domain [F_{low},F_{upper}]
                    end
                    if local search type LS_t = "normal":
                        $NSi = Gi + x$
                        Checking domain [F_{low},F_{upper}]
                    end
                else:
                    $NSi = Gi$
                    Continue to the next dimension
                end
            end
            save the 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 minimum 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 the new individual then begin the next iteration

Results

Best 20 parameter combinations

We find the best 20 parameter combinations by using a grid search. First, we propose different sequential values in iterations, mutation rate, crossover rate, number of individuals, and search radius R, as shown in Table 4.

Table 4 Parts of the sequential values we tried

iterations	[500, 1000,1500, 2000, 3000,5000,6500,10000,20000,1000000]
mutation rate	[0.001/D, 0.02/D, 0.04/D, 0.06/D, 0.08/D, 0.18/D ,0.1/D, 0.2/D ,0.5/D, 1.0/D, 2.0/D]
crossover rate	[0.3,0.5,0.7,0.9,1.0]
number of individuals	[50, 100, 150, 200, 350, 400, 1000]
search radius R	[0.01,0.03,0.05,0.1,0.3,0.5]

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 all parameter combinations for each combination of mutation type and crossover type using SSGA.

Given the number of parameter combinations *M,* σ is the index of a parameter combination, σ belongs to $[1,2,3,\ldots,6M]$. Fi represents a function where i can be$[1,2,3,\ldots,23]$.γ_σ^{Fi} represents the actual solution produced by a specific function and a specific parameter combination. $Rank_\sigma^{Fi}$ represents the ranking for γ_σ^{Fi} among all γ^{Fi} . For example, if γ_1^{F1} produce the smallest solution among all γ^{F1}, then γ_1^{F1} ranked 1st and $Rank_\sigma^{Fi}$=1.

$score_\sigma$ represents the overall performance of a parameter combination of all functions.

$$score_\sigma = \sum_{\sigma=1}^{6M} \sum_{i=1}^{23} Rank_\sigma^{Fi}$$

The smaller the score is, the better.

Since all the parameter combinations are truly numerous, just the contents of Table 4 can produce 10*11*5*7*6=23100 and considering 6 combinations of mutation type and crossover type and 23 functions, then we end up with 23100*6*23=3187800 runs even if we run once for each function and each parameter combination. Therefore, we have chosen some representative functions in our search for good parameters, such as F1, F12, F18, F22 and so on.

We have selected the best 20 parameters found to learn how the Baldwinian and Lamarckian algorithms may improve the SSGA.

Table 5 shows the best parameter combinations for SSGA. The letters in the table 2 header represent, in order, the 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.

The parameters of the local search procedure for Lamarckian and Baldwinian algorithms are proposed in Table 6, other parameters are the same as in Table 5. The letters in the table 6 header represent, in order, the 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 5 best 20 parameter combinations for SSGA

σ	Max_{iter}	γ	β	δ	M_t	C_t	R	Tol	N	θ
1	1000000	2/D	100	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
2	1000000	2/D	100	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
3	1000000	1/D	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
4	1000000	2/D	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
5	1000000	2/D	100	0.7	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
6	1000000	2/D	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
7	1000000	1/D	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
8	1000000	2/D	100	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
9	1000000	1/D	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
10	1000000	1/D	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
11	1000000	2/D	100	0.7	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
12	1000000	1/D	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
13	1000000	2/D	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
14	1000000	0.5/D	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
15	1000000	1/D	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
16	1000000	1/D	100	0.7	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
17	1000000	1/D	200	0.5	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
18	1000000	1/D	200	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
19	1000000	1/D	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001
20	1000000	1/D	100	0.6	Normal	Probabilistic crossover	0.1	0.0001	3000	0.0001

Table 6 parameters for the local search procedure

σ	α	LS_t	W	Mode	σ	α	LS_t	W	Mode
1	0.5	Uniform	1	Baldwin/Lamarck	11	0.5	Uniform	1	Baldwin/Lamarck
2	0.5	Uniform	1	Baldwin/Lamarck	12	0.5	Uniform	1	Baldwin/Lamarck
3	0.5	Uniform	1	Baldwin/Lamarck	13	0.5	Uniform	1	Baldwin/Lamarck
4	0.5	Uniform	1	Baldwin/Lamarck	14	0.5	Uniform	1	Baldwin/Lamarck
5	0.5	Uniform	1	Baldwin/Lamarck	15	0.5	Uniform	1	Baldwin/Lamarck
6	0.5	Uniform	1	Baldwin/Lamarck	16	0.5	Uniform	1	Baldwin/Lamarck
7	0.5	Uniform	1	Baldwin/Lamarck	17	0.5	Uniform	1	Baldwin/Lamarck
8	0.5	Uniform	1	Baldwin/Lamarck	18	0.5	Uniform	1	Baldwin/Lamarck
9	0.5	Uniform	1	Baldwin/Lamarck	19	0.5	Uniform	1	Baldwin/Lamarck
10	0.5	Uniform	1	Baldwin/Lamarck	20	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 *the 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.

Table 7, Table 8, and Table 9 show the percentage values of finding the global minimum of 23 functions for SSGA, Baldwinian and Lamarckian algorithms, respectively. The first row of Table 7-9 is Function and the index of a parameter combination.

Table 7 Percentage values for SSGA

F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F12	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F13	30%	20%	60%	20%	20%	10%	80%	30%	40%	80%	0%	40%	10%	50%	90%	50%	80%	70%	90%	50%
F14	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F15	50%	50%	0%	40%	60%	10%	20%	20%	10%	0%	40%	30%	20%	0%	10%	20%	10%	30%	10%	0%
F16	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F17	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F18	70%	60%	100%	80%	70%	60%	100%	70%	100%	100%	70%	100%	60%	100%	100%	100%	100%	100%	100%	100%
F19	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F20	60%	60%	40%	40%	70%	70%	50%	40%	50%	30%	50%	50%	60%	50%	30%	70%	20%	40%	50%	60%
F21	10%	20%	20%	10%	50%	30%	20%	10%	60%	60%	50%	40%	60%	40%	40%	30%	30%	30%	30%	50%
F22	80%	80%	40%	50%	70%	50%	70%	50%	50%	40%	80%	60%	60%	50%	50%	70%	70%	20%	60%	40%
F23	60%	80%	50%	60%	60%	70%	20%	60%	70%	60%	60%	60%	40%	30%	70%	30%	40%	60%	70%	60%

Table 8 Percentage values for Baldwinian algorithm

F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F12	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	90%	100%	100%	100%
F13	20%	20%	80%	0%	30%	20%	70%	0%	60%	50%	0%	70%	30%	20%	90%	60%	80%	80%	40%	70%
F14	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F15	70%	60%	50%	40%	40%	50%	0%	30%	0%	10%	30%	20%	50%	10%	0%	50%	20%	30%	10%	10%
F16	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F17	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F18	70%	90%	100%	70%	70%	60%	100%	80%	100%	100%	70%	100%	80%	100%	100%	100%	100%	100%	100%	100%
F19	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F20	40%	50%	30%	40%	50%	30%	20%	30%	40%	50%	20%	50%	40%	40%	40%	40%	50%	80%	80%	30%
F21	80%	70%	70%	80%	100%	70%	40%	80%	50%	90%	70%	60%	70%	50%	50%	60%	50%	60%	70%	50%
F22	90%	90%	70%	90%	60%	90%	90%	70%	80%	70%	90%	80%	90%	40%	100%	60%	60%	80%	80%	60%
F23	80%	70%	90%	90%	100%	80%	80%	100%	90%	80%	90%	90%	80%	70%	70%	80%	100%	70%	80%	70%

Table 9 Percentage values for Lamarckian algorithm

F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
F12	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F13	20%	20%	60%	10%	0%	20%	60%	0%	60%	70%	30%	80%	0%	50%	80%	70%	80%	50%	60%	80%
F14	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F15	70%	90%	30%	70%	50%	60%	20%	40%	10%	40%	70%	10%	50%	10%	20%	30%	20%	40%	40%	20%
F16	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F17	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F18	70%	50%	100%	70%	60%	80%	100%	50%	100%	100%	90%	100%	70%	100%	100%	100%	100%	100%	100%	100%
F19	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
F20	40%	50%	40%	50%	30%	50%	20%	50%	40%	50%	30%	30%	80%	40%	50%	30%	30%	30%	40%	50%
F21	60%	80%	70%	40%	80%	30%	100%	50%	60%	70%	80%	60%	60%	60%	60%	60%	80%	50%	70%	60%
F22	80%	100%	70%	90%	90%	100%	60%	80%	70%	80%	70%	80%	80%	60%	60%	100%	70%	80%	90%	60%
F23	80%	100%	90%	90%	90%	90%	60%	90%	90%	100%	90%	100%	90%	90%	100%	100%	80%	90%	60%	90%

Figure 5 shows the bar plots distinguished by each function for percentage values produced by 20 parameter combinations, every three bars together from left to right represent SSGA, Baldwinian and Lamarckian algorithms respectively. Each bar in Figure 5 is generated by 20 data points. The height of a rectangle is the mean probability. The mean probability is the mean of the percentage values produced by 20 parameter combinations. The criterion for finding the minimum successfully is *abs(solution-global minimum) < threshold=0.01*.

Figure 6 shows bar plots with a different threshold of *0.0001*, generated in the same way as Figure 5. The subsequent analysis is based on Figure 6.

Figure 5 Bar plots for percentage values for SSGA, Baldwinian and Lamarckian algorithms(threshold=0.01)

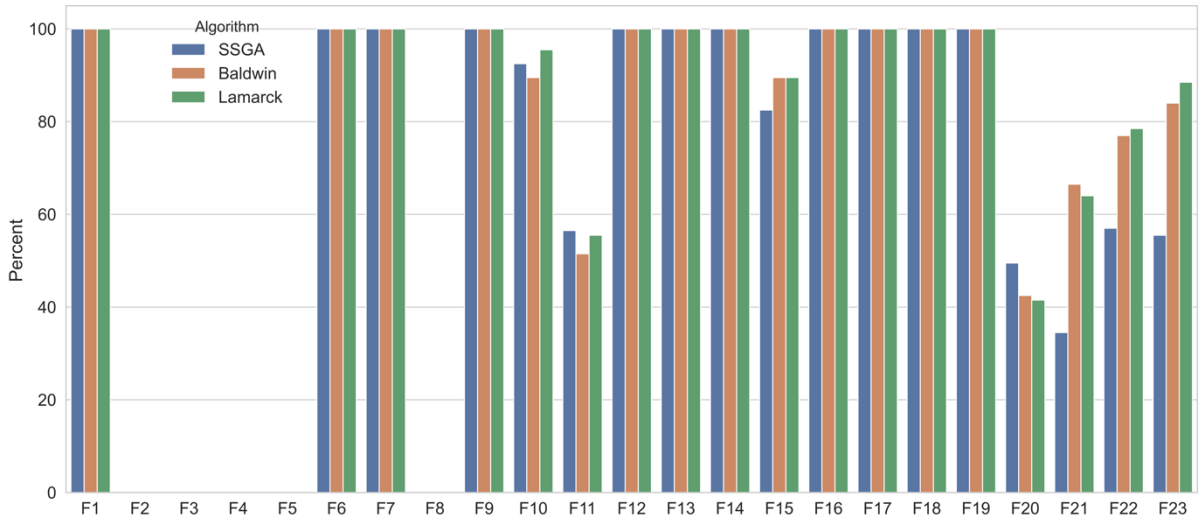


Figure 6 Bar plots for percentage values for SSGA, Baldwinian and Lamarckian algorithms(threshold=0.0001)

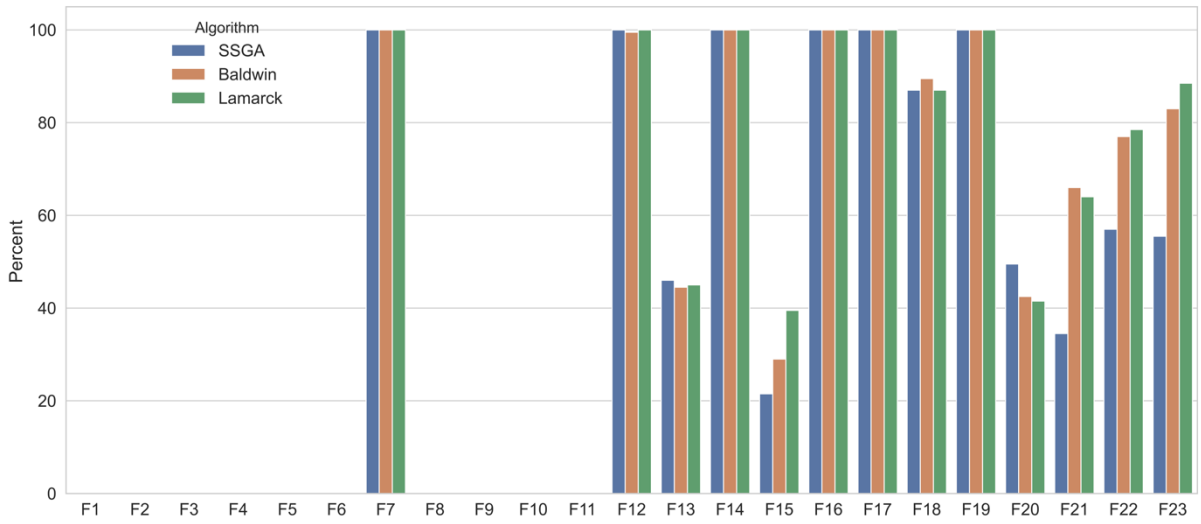


Figure 6 shows that the differences in performance between SSGA, Baldwinian and Lamarckian are in the 7 functions F13, F15, F18, F20, F21, F22, and F23. The following conclusions for Figure 6 can be drawn:

1. The Baldwinian and Lamarckian algorithm performs slightly less well than SSGA on function F13 and F20.
2. The Baldwinian and Lamarckian algorithms are obviously better for the 4 functions F15, F21, F22 and F23.

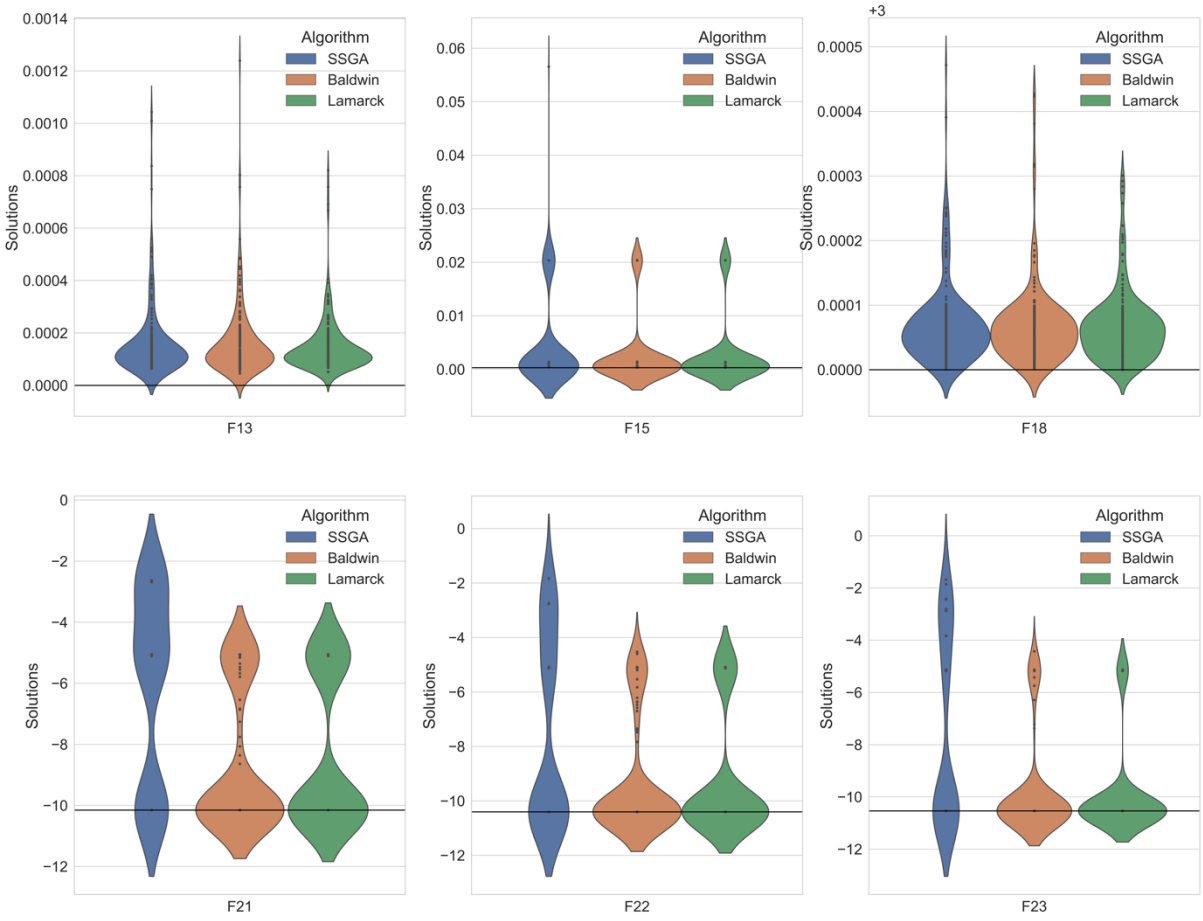
3. The performance of the three algorithms on other functions is comparable.

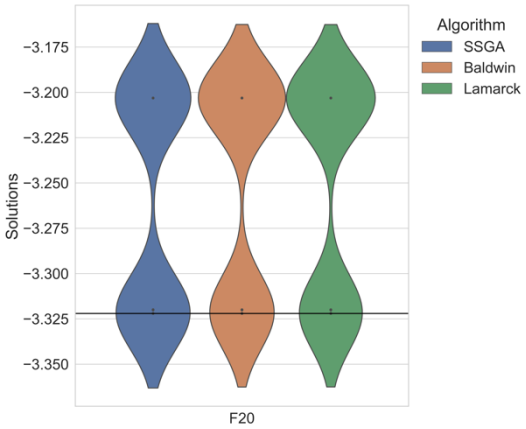
In order to better demonstrate the progress brought by the local search procedure of Baldwinian and Lamarckian algorithms, we prepared violin plots. A violin plot is designed to provide a visual representation of the distribution of numerical data by using a kernel density estimator, which shows the 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, the more data points there are. On the contrary, the thinner areas represent the lower probability and fewer data concentrated here.

Figure 7 shows the violin plots for SSGA, Baldwinian and Lamarckian algorithms on 7 functions F13, F15, F18, F20, F21, F22, and F23. Each subplot has 3 violins together which from left to right represent SSGA, Baldwinian and Lamarckian algorithms respectively.

Each subplot has a horizontal line running through it, which represents the position of the global minimum of a function. The criterion for finding the minimum successfully is *abs(solution-global minimum) < 0.0001*, which is the same as Figure 6. 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. Note that no black dot is under the horizontal line since that global minimum is the smallest solution. The reason why some violin plots occasionally extend below the global minimum is because of the kernel density estimator. In addition, the reason for not seeing so many points in Figure 7 is that many of the solutions are the same.

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





Given that our aim is to find the minimum of a test function, we expect the violin to be short and fat, the shorter and fatter the better. The following conclusions can be drawn from Figure 7:

1. The global minimums for F13, F18, and F20 are 0, 3, and -3.32, respectively. From Figure 6 we see that there is no significant difference in the mean probability of finding the minimum in these 3 functions for the SSGA, Baldwinian and Lamarckian algorithms, but they do differ. Although different, the minimum values of each function they find are of the same order of magnitude, and their distribution of violin plots is basically identical. It is fair to conclude that Baldwinian and Lamarckian have not improved on these 3 functions, but they are not worse either.
2. On the 4 functions F15, F21, F22, and F23, it is very obvious in Figure 7 that the Lamarckian and Baldwinian algorithms find lower solutions than SSGA does, the corresponding violin plots are noticeably fatter and shorter for Baldwinian and Lamarckian algorithms.
3. Baldwinian algorithm performs better than Lamarckian algorithm on F21 while Lamarckian algorithm is better on F13, and F23.
4. Interestingly, the SSGA and Baldwinian violin plots for F21, F22 and F23 show more distinct black dots, which, as mentioned earlier, are the minimum values eventually found by an algorithm with a maximum number of iterations of one million, and these 3 functions again are exactly multimodal functions. If these black dots are local minimums, whether this means that the Lamarckian algorithm has a stronger ability to get rid of local minimums remains to be further investigated.

These algorithms perform almost equally for other functions, such as F1-F6 and F8-F11, and the solutions they achieve are all in the same order of magnitude and also very similar. This will be proven in the convergence analysis coming next.

Convergence analysis

Based on the convergence properties of the functions, we have divided the 23 functions into 3 groups. First group contains 12 functions, which are F1, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, and F13. The second group consists of 7 functions, namely F2, F14, F16, F17, F18, F19, and F20. The last group consists of 4 functions, i.e., F15, F21, F22, and F23. Among all following convergence plots, the horizontal coordinate denotes *iterations/50*, the vertical coordinate denotes the fitness value of the best individual, and the horizontal line specifies the position of the global optimal value.

Figures 8-10 show convergence plots for the first group. From following Figures 8-10, the following conclusion for the first group can be drawn:

1. The convergence rate of Lamarckian algorithm and SSGA is comparable. However, Lamarckian converges a little faster than SSGA on F1, F3, F5, F6 ,F7, F11,F12 and F13. One can see that the position of Lamarck's convergence line is a little lower than that of SSGA.
2. Both SSGA and Lamarckian algorithms converge faster than the Baldwinian algorithm.
3. Crucially, all three algorithms are essentially able to converge to the vicinity of the global minimum of all functions in the first group except for F8, they just fail to achieve an accuracy of *0.0001*.
4. Ultimately, the three algorithms will converge on essentially the same level in terms of the performance of the first group of functions.

Figure 8 Convergence plots for F8, F9 in first group, iterations = 20,000

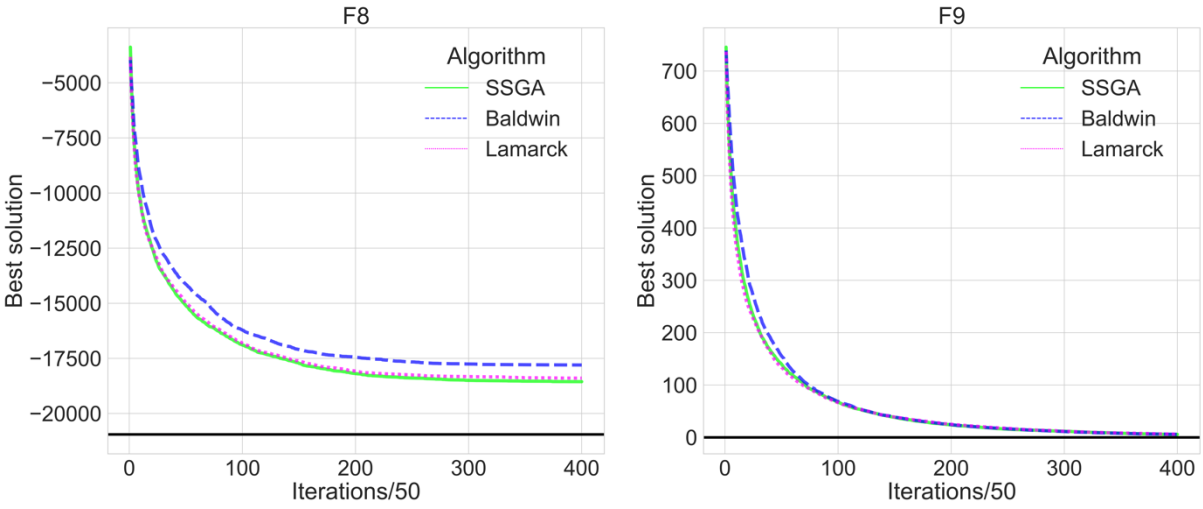


Figure 9 Convergence plots for F4, F10 in first group, iterations = 150,000

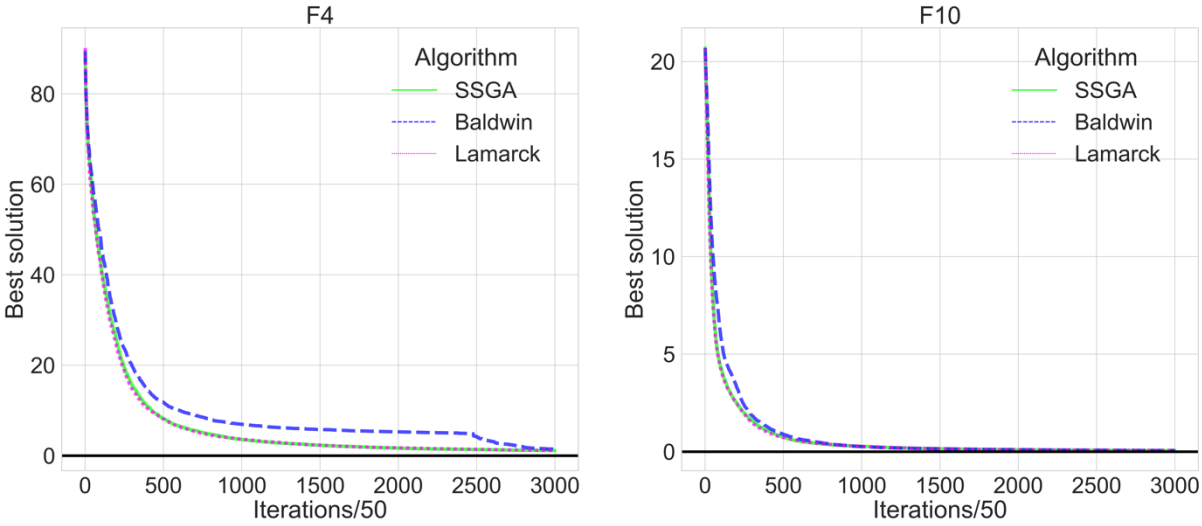


Figure 10 Convergence plots for F1, F3, F5, F6, F7, F11, F12, F13 in first group, iterations = 5,000

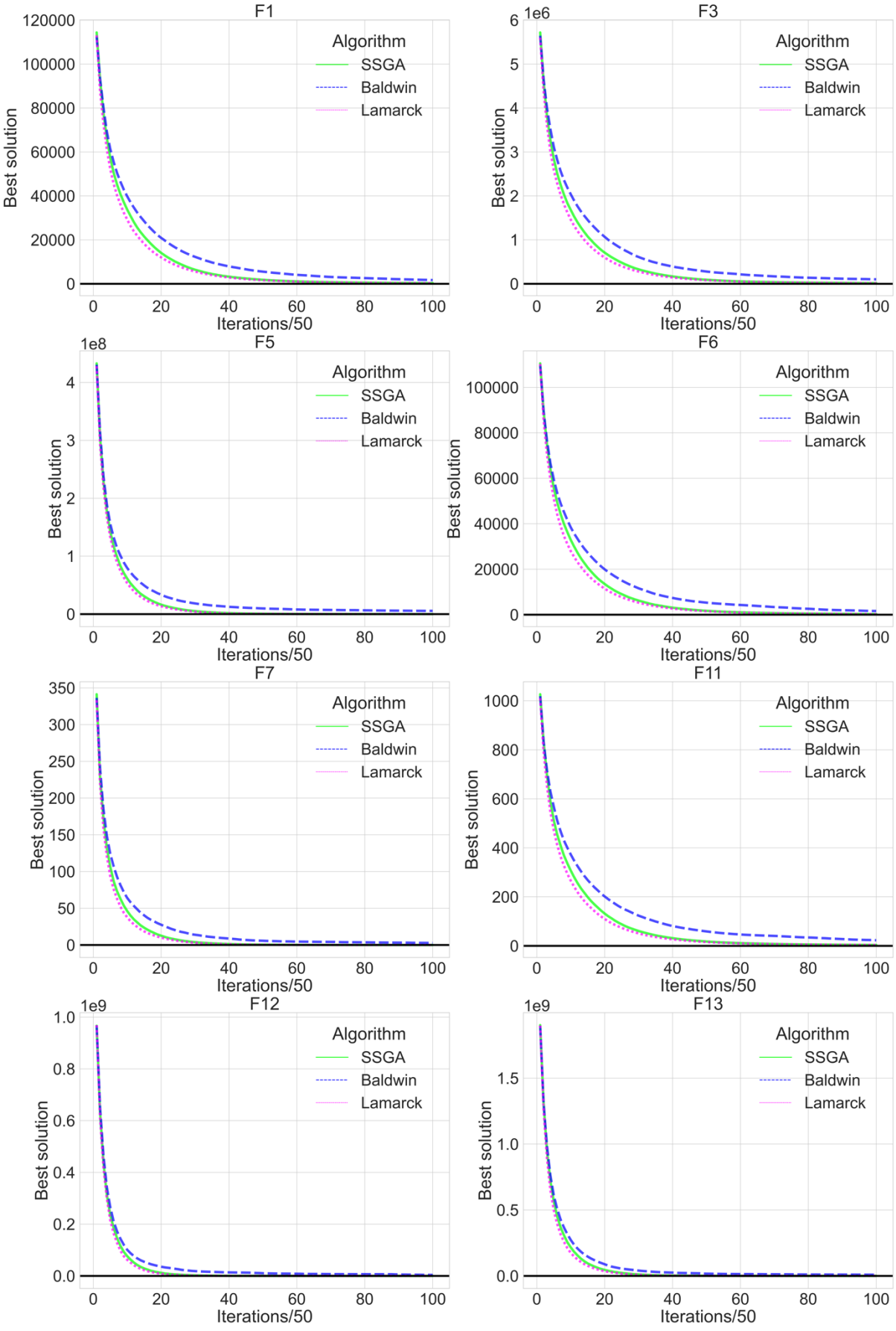


Figure 11 Convergence plots for the second group

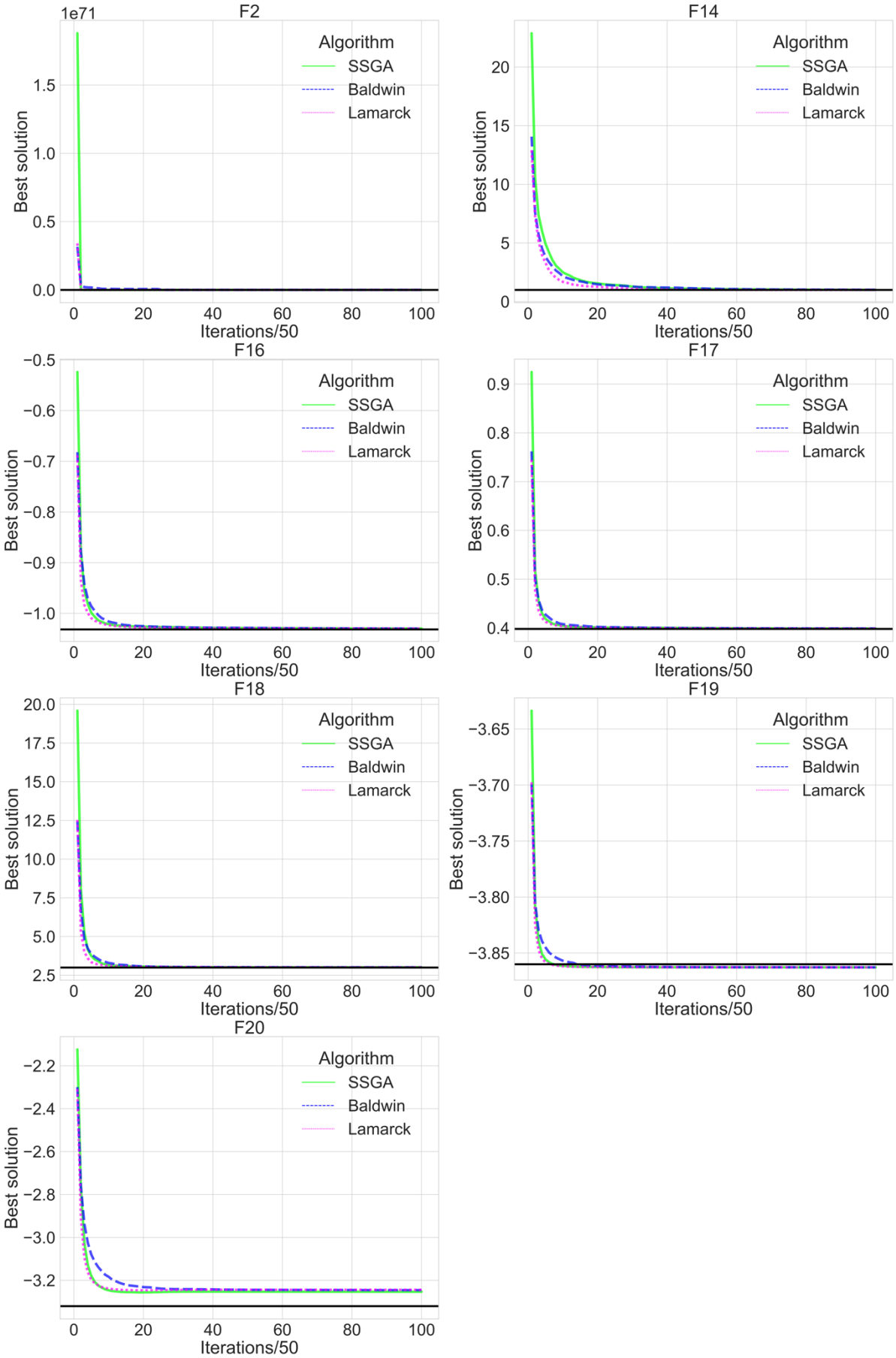


Figure 12 boxen plots for the second group

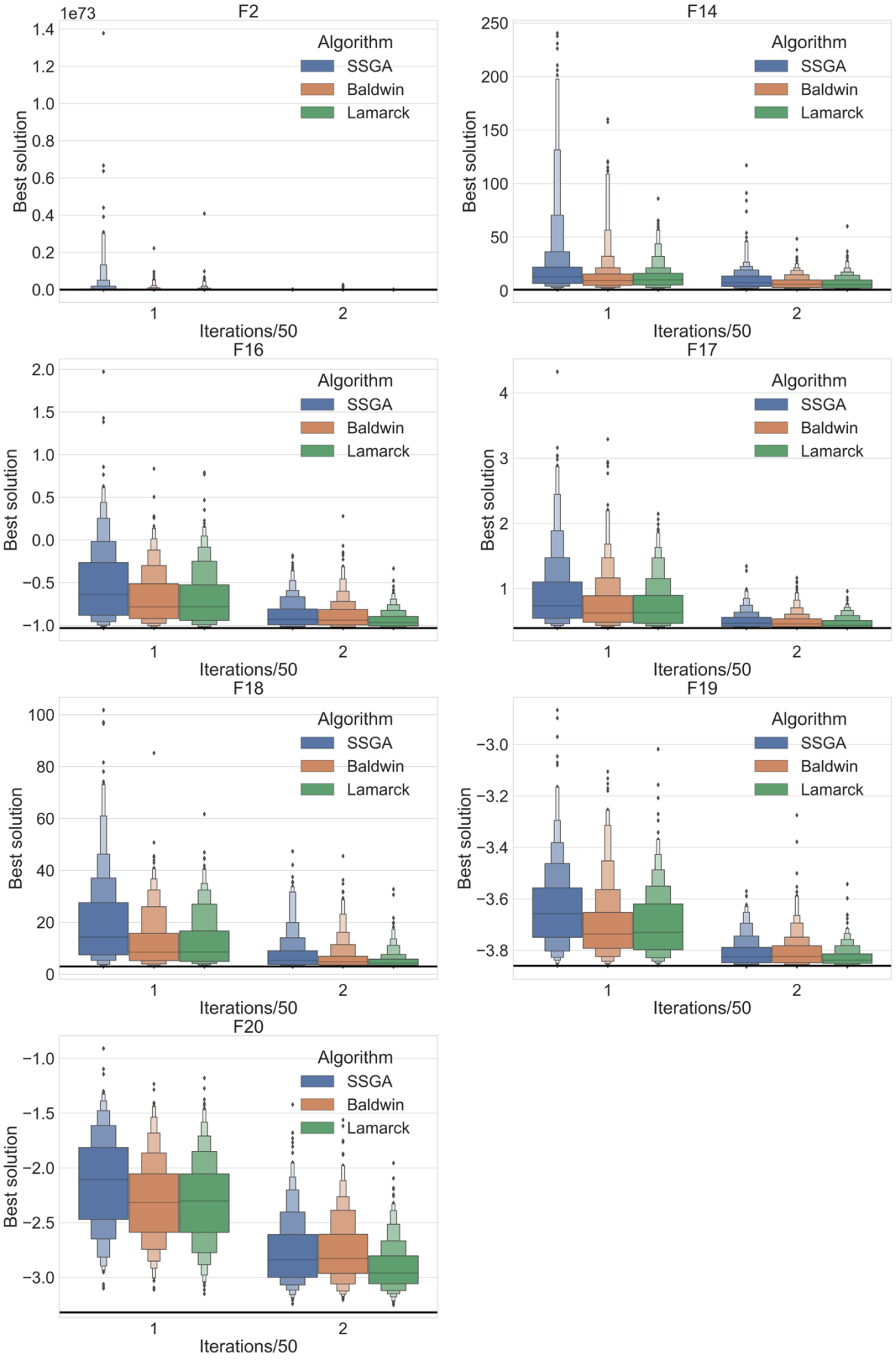


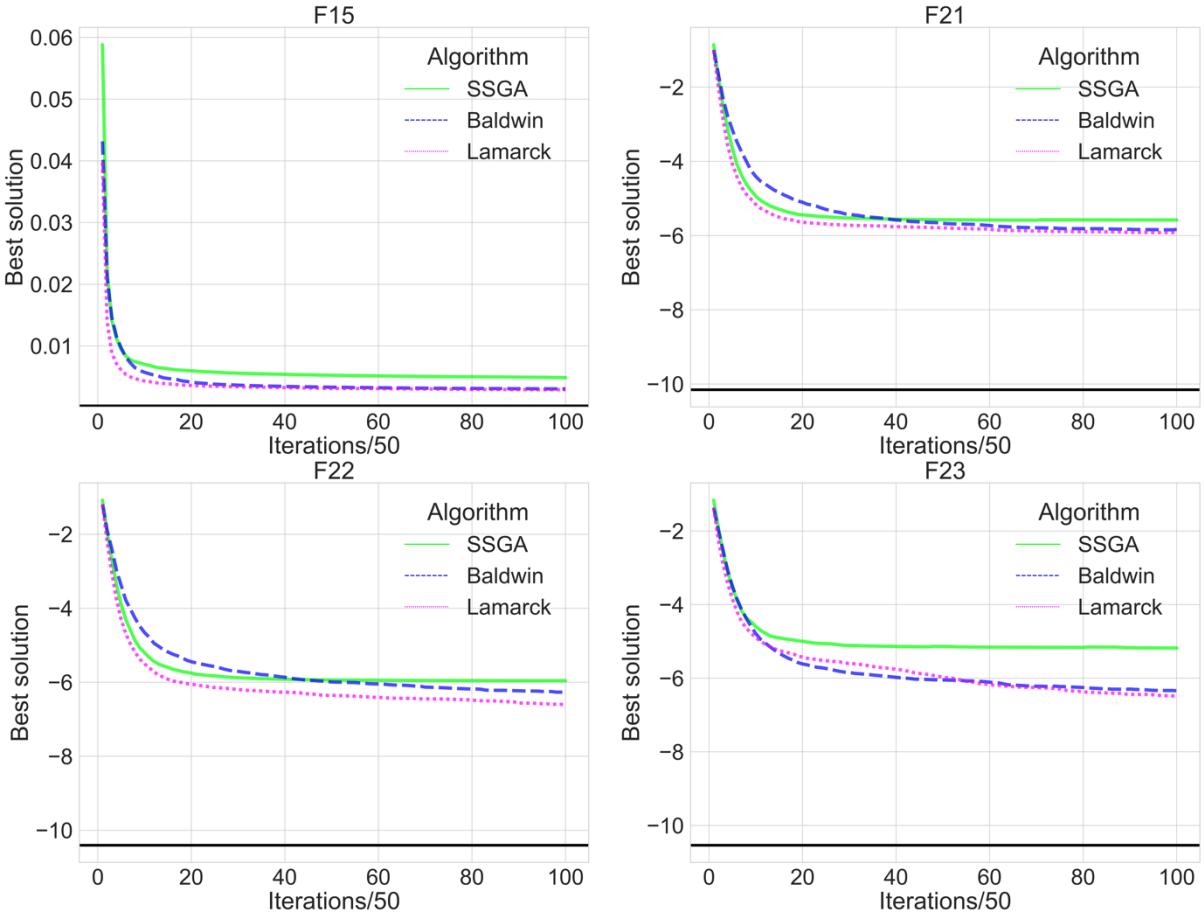
Figure 11 shows the convergence plots for the second function group. Figure 12 shows the boxen plots for the second group functions in terms of iteration =50 (left side of the plots, marked with 1) and iteration=100 (right side of the plots, marked with 2). A boxen plot is similar to a box plot but provides more information about the shape of data distribution, particularly in tails. Each boxen plot is generated based on 20 runs for 20 parameter combinations in Table 5 and Table 6, but the max number of iterations changed. Each subplot has 3 boxes together which from left to right represent SSGA, Baldwinian and Lamarckian algorithms respectively.

From Figures 11-12, the following conclusion for the second group can be drawn:

1. It is easy to see that Baldwinian and Lamarckian algorithms perform better during the initial iterations. It is obvious that the line of SSGA is on top during the initial iterations, which means that the solutions found by the SSGA are relatively far from the global minimum.
2. The conclusions of the first group are also applicable to the second group of functions. Nevertheless, they end up converging at the same level. The quality of the minimum found is almost equal.

The following Figure 13 shows convergence plots for the third group functions (F15, F21, F22, F23), these 4 functions are exactly the ones where the Baldwinian and Lamarckian algorithms show a significant difference compared to SSGA in the previous bar plots and violin plots. We can see that Lamarckian algorithm converged the fastest at the beginning, with SSGA second and Baldwinian third, but Baldwinian quickly overtook SSGA's position to become second. This is a situation that is not seen in other functions.

Figure 13 Convergence plots for F15, F21, F22, F23



Conclusions

The results discussed in the previous section demonstrate the improvements made by the Baldwinian and Lamarckian algorithms over the SSGA, with the dominant differences found in F15, F21, F22, and F23. For the other functions, the Baldwinian and Lamarckian algorithms perform comparably to the SSGA. Overall, Baldwinian and Lamarckian algorithms are better than SSGA. This means that the local search procedures of the Baldwinian and Lamarckian algorithms are useful to a certain level, as we expected for the first goal. It is worth noting that both algorithms need to use the test function twice in each iteration, whereas SSGA uses the test function only once in each iteration.

With regard to the analysis of convergence rates and results, over many functions (12 functions in the first group) the three algorithms eventually converge at an equivalent level, producing solutions of identical quality. On several functions (7 functions in the second group), the Baldwinian and Lamarckian algorithms show apparent superiority in the initial iterations. In the third group of 4 functions, Lamarckian and Baldwinian algorithms are able to converge to a lower position, which means that the solutions they can find will be closer to the global minimum.

Our second goal is to compare the Baldwinian and Lamarckian algorithms. Lamarckian algorithm converges faster than Baldwinian algorithm does. It is hard to claim that the Lamarckian algorithm is definitely better than the Baldwinian algorithm because no significant difference is found between the output of the two algorithms in the long run, and a fast convergence rate may be a disadvantage in some cases.

Future work may try to specify different search radiuses (R) for different functions, as the proposed algorithms are actually capable of finding the global optimal value of many functions, but fail to meet the high accuracy requirements, and this could be overcome if the range of search radius R were reduced. Alternatively, a direction worth investigating is whether local search procedures can get rid of local minima to a certain degree. In addition, further, more complex local search procedures (like gradient descent) will also be investigated, and the studies might also be extended to further functions and more parameter combinations.

References

[1] Nocedal, J. and Wright, S. (2006) Numerical Optimization. 2nd ed. New York, NY: Springer.

[2] Michalewicz, Z. (1992) *Genetic algorithms + data structures = evolution programs*. Berlin, Heidelberg: Springer Berlin Heidelberg.

[3] Michalewicz, Z. and Schoenauer, M. (1996) "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary computation*, 4(1), pp. 1–32. doi: 10.1162/evco.1996.4.1.1.

[4] Mirjalili, S. (2019) "Genetic Algorithm," in *Studies in Computational Intelligence*. Cham: Springer International Publishing, pp. 43–55.

[5] Collins, R. J. and Jefferson, D. R. (1991) "Selection in Massively Parallel Genetic Algorithms," *International Conference on Genetic Algorithms*.

[6] Ishibuchi, H. and Yamamoto, T. (2004) "Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining," *Fuzzy Sets and Systems. An International Journal in Information Science and Engineering*, 141(1), pp. 59–88. doi: 10.1016/s0165-0114(03)00114-3.

[7] Hutter, M. (2002) "Fitness uniform selection to preserve genetic diversity," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. IEEE. doi: 10.1109/cec.2002.1007025.

[8] Grefenstette, J. J. and Baker, J. E. (1989) "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism," *International Conference on Genetic Algorithms*.

[9] Syswerda, G. (1989) "Uniform Crossover in Genetic Algorithms," *International Conference on Genetic Algorithms*. doi: 10.1016/B978-0-08-094832-4.50021-0.

[10] Semenkin, E. and Semenkina, M. (2012) "Self-configuring genetic algorithm with modified uniform crossover operator," in *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 414–421.

[11] Hu, X. B. and Di Paolo, E. (2007) "An efficient Genetic Algorithm with uniform crossover for the multi-objective Airport Gate Assignment Problem," in *2007 IEEE Congress on Evolutionary Computation*. IEEE. doi: 10.1109/cec.2007.4424454.

[12] Tsutsui, S., Yamamura, M. and 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*. Morgan Kaufmann Publishers Inc, pp. 657–664.

[13] Bäck, T. and Fogel, D. B. (2000) *Evolutionary computation 1: Basic algorithms and operators*. Edited by Z. Michalewicz. CRC press.

[14] Oliver, I. M., Smith, D. and Holland, J. R. (1987) "Study of permutation crossover operators on the travelling salesman problem," in Associates, L. E. (ed.) *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*. Cambridge, MA. Hillsdale, NJ. pp. 224–230.

[15] Davis, L. (1985) "Applying adaptive algorithms to epistatic domains," *In IJCAI*, 85, pp. 162–164.

[16] Hinterding, R. (1995) "Gaussian mutation and self-adaption for numeric genetic algorithms," in *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*. IEEE. doi: 10.1109/icec.1995.489178.

[17] Tsutsui, S. and 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 Associates, L. E. (ed.) *Proceedings of the second International Conference on Genetic Algorithms and their Applications*. Cambridge, MA. Hillsdale, NJ. pp. 132–139.

[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*. Elsevier, pp. 53–68. doi: 10.1016/b978-0-08-050684-5.50007-0.

[21] Kallel, L., Naudts, B. and Reeves, C. R. (2001) "Properties of fitness functions and search landscapes," in *Natural Computing Series*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 175–206. doi: 10.1007/978-3-662-04448-3_8.

[22] Van Cleve, J. and Weissman, D. B. (2015) "Measuring ruggedness in fitness landscapes," *Proceedings of the National Academy of Sciences of the United States of America*, 112(24), pp. 7345–7346. doi: 10.1073/pnas.1507916112.

[23] Dawkins, R. (1978) "The Selfish Gene," *Evolution; international journal of organic evolution*, 32(1), pp. 220-221. doi: 10.2307/2407425.

[24] Moscato, P. (1989) *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Pasadena.

[25] Krasnogor, N. (2012) "Memetic Algorithms," in *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 905–935. doi: 10.1007/978-3-540-92910-9_29.

[26] Radcliffe, N. J. and Surry, P. D. (1994) "Formal memetic algorithms," in *Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–16. doi: 10.1007/3-540-58483-8_1.

[27] Garg, P. (2010) "A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm," *arXiv [cs.CR]*. Available at: http://arxiv.org/abs/1004.0574.

[28] Pandey, H. M., Chaudhary, A. and Mehrotra, D. (2014) "A comparative review of approaches to prevent premature convergence in GA," *Applied soft computing*, 24, pp. 1047–1077. doi: 10.1016/j.asoc.2014.08.025.

[29] García-Martínez, C. and Lozano, M. (2007) "Local search based on genetic algorithms," in *Natural Computing Series*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 199–221.

[30] Hinton, G. and Nowlan, S. (1987) "How learning can guide evolution," *Complex Syst*, 1, pp. 495–502.

[31] Bull, L., Holland, O. and Blackmore, S. (2000) "On meme-gene coevolution," *Artif Life*, 6, pp. 227–235. doi: 10.1162/106454600568852.

[32] Houck, C. R. *et al.* (1997) "Empirical investigation of the benefits of partial Lamarckianism," *Evolutionary computation*, 5(1), pp. 31–60. doi: 10.1162/evco.1997.5.1.31.

[33] Mayley, G. (1996) "Landscapes, learning costs, and genetic assimilation," *Evolutionary computation*, 4(3), pp. 213–234. doi: 10.1162/evco.1996.4.3.213.
[34] Turney, P. (1996) "How to shift bias: Lessons from the Baldwin effect," *Evolutionary computation*, 4(3), pp. 271–295. doi: 10.1162/evco.1996.4.3.271.
[35] Whitley, D., Gordon, V. S. and Mathias, K. (1994) "Lamarckian evolution, the Baldwin effect and function optimization," in *Parallel Problem Solving from Nature — PPSN III*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 5–15. doi: 10.1007/3-540-58484-6_245.
[36] Gruau, F. and Whitley, D. (1993) "Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect," *Evolutionary computation*, 1(3), pp. 213–233. doi: 10.1162/evco.1993.1.3.213.
[37] Turney, P. D. (2002) "Myths and legends of the Baldwin effect," *arXiv [cs.LG]*. Available at: http://arxiv.org/abs/cs/0212036.
[38] Houck, C. &., Joines, J. &. and Kay, M. (1996) *Utilizing Lamarckian Evolution and the Baldwin Effect in Hybrid Genetic Algorithms*.
[39] Bereta, M. (2019) "Baldwin effect and Lamarckian evolution in a memetic algorithm for Euclidean Steiner tree problem," *Memetic computing*, 11(1), pp. 35–52. doi: 10.1007/s12293-018-0256-7.
[40] Ishibuchi, H., Kaige, S. and Narukawa, K. (2005) "Comparison between Lamarckian and baldwinian repair on multiobjective 0/1 knapsack problems," in *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 370–385.
[41] Paenke, I., Jin, Y. and Branke, J. (2009) "Balancing population- and individual-level adaptation in changing environments," *Adaptive behavior*, 17(2), pp. 153–174. doi: 10.1177/1059712309103566.
[42] Paenke, I. *et al.* (2007) "On the adaptive disadvantage of lamarckianism in rapidly changing environments," in *Advances in Artificial Life*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 355–364. doi: 10.1007/978-3-540-74913-4_36.
[43] Ong, Y. S. and Keane, A. J. (2004) "Meta-Lamarckian learning in memetic algorithms," *IEEE transactions on evolutionary computation : a publication of the IEEE Neural Networks Council*, 8(2), pp. 99–110. doi: 10.1109/tevc.2003.819944.
[44] Faramarzi, A. *et al.* (2020) "Marine Predators Algorithm: A nature-inspired metaheuristic," *Expert systems with applications*, 152(113377), p. 113377. doi: 10.1016/j.eswa.2020.113377.