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.

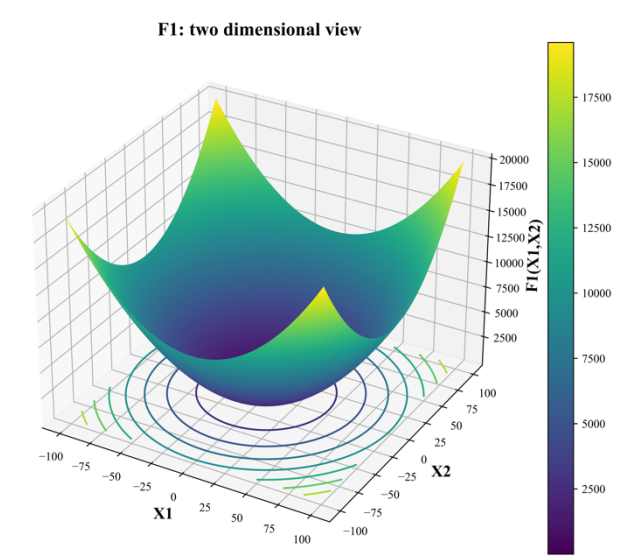
Genetic and Memetic Algorithm

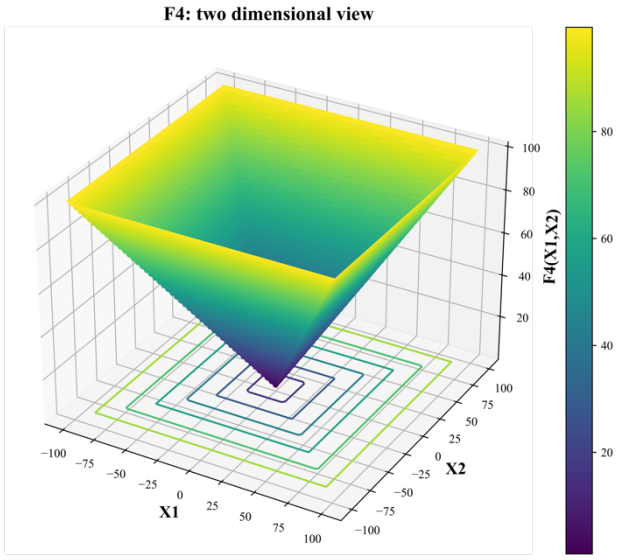
Fitness function

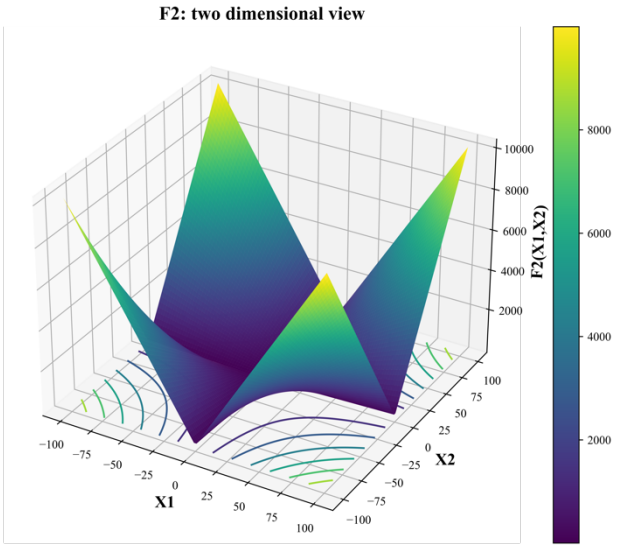
CEC-BC-2017 contains 23 benchmark functions with different characteristics and levels of difficulty, which can be used to assess the performance of new algorithms. Table 1 shows 23 test functions with details. The performance of our algorithms is evaluated based on CEC-BC-2017 test functions for optimization.

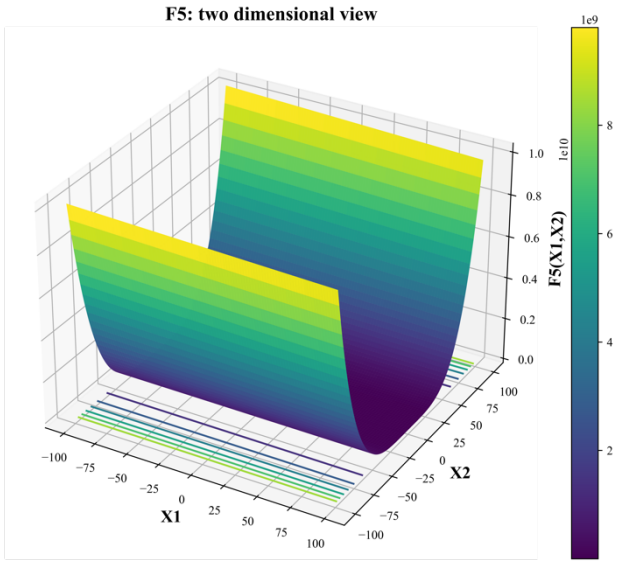
Function	Dim	Range	Optimal
$F1 = \sum_{i=1}^{50} x_i^2$	50	[-100,100]	0
$F2 = \sum_{i=1}^{50} \lvert x_i \rvert + \prod_{i=1}^{50} \lvert x_i \rvert$	50	[-100,100]	0
$F3 = \sum_{i=1}^{50} \left(\sum_{i=1}^{50} = (x_i)^2 \right)$	50	[-100,100]	0
$F4 = max\lvert x_i \rvert$	50	[-100,100]	0
$F5 = \sum_{i=1}^{49} \left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2 \right)$	50	[-30,30]	0
$F6 = \sum_{i=1}^{49} (x_i + 0.5)^2$	50	[-100,100]	0
$F7 = \sum_{i=1}^{50} \left(i x_i^4 \right)$	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
$F9 = \sum_{i=1}^{50} \left(x_i^2 - 10\, cos(2\pi x_i) + 10 \right)$	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
$F12 = \frac{\pi}{50} \Big(10 sin^2(\pi y_1) + \sum_{i=1}^{49} (y_i - 1)^2 (1 + 10 sin^2\ \pi y_{i+1} + (y_{50} - 1)^2) + \sum_{i=1}^{50} u(x_i, 10, 100, 4)$	50	[-50,50]	0
$F13 = 0.1 \Big(sin^2(3\pi x_1) + \sum_{i=1}^{49} (x_i - 1)^2 \left(1 + sin^2\left(3\pi x_i + 1\right)\right) + (x_{50} - 1)^2 \left(1 + sin^2\left(2\pi x_{50}\right)\right) \Big) + \sum_{i=1}^{50} u(x_i, 5, 100, 4)$	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]] $F14 = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} \left(x_i - a_{ij}\right)^6} \right)^{-1}$	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
$F16 = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	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
$F18 = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2)]$	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]] $F19 = -\sum_{i=1}^{4} \left(c_i exp\left(-\sum_{j=1}^{3} a_{ij} \left(x_j - p_{ij}\right)^2 \right) \right)$	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]] $F20 = -\sum_{i=1}^{4} \left(c_i exp\left(-\sum_{j=1}^{6} a_{ij} \left(x_j - p_{ij}\right)^2 \right) \right)$	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
$U(x, a, k, m) = k((x - a)^m)(x > a) + k((-x - a)^m)(x < (-a))$			

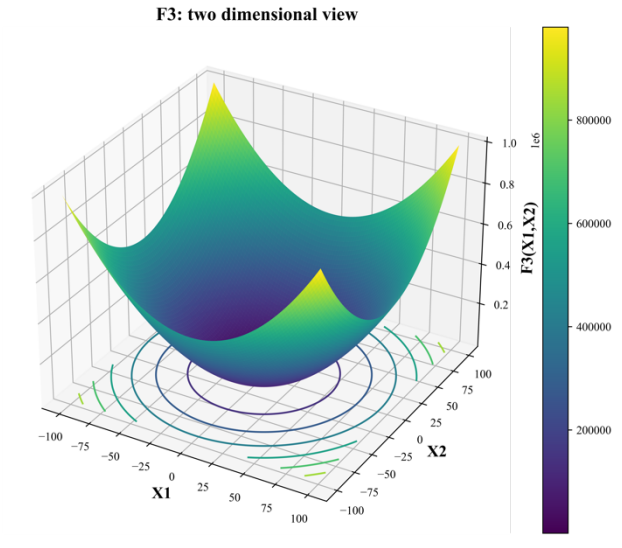
Table 1 CEC-BC-2017 benchmark functions

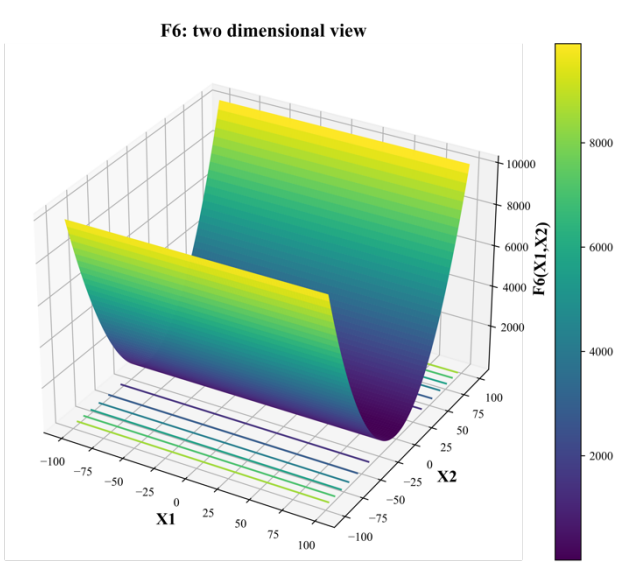

F1: two dimensional view
F1(X1,X2)
X1
X2

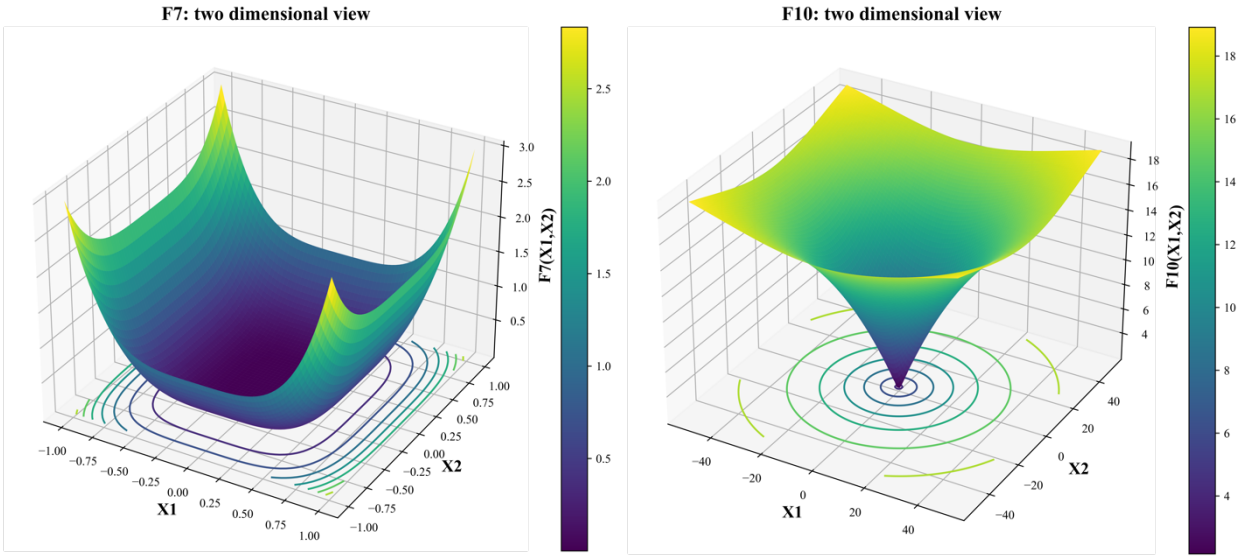

F4: two dimensional view
F4(X1,X2)
X1
X2

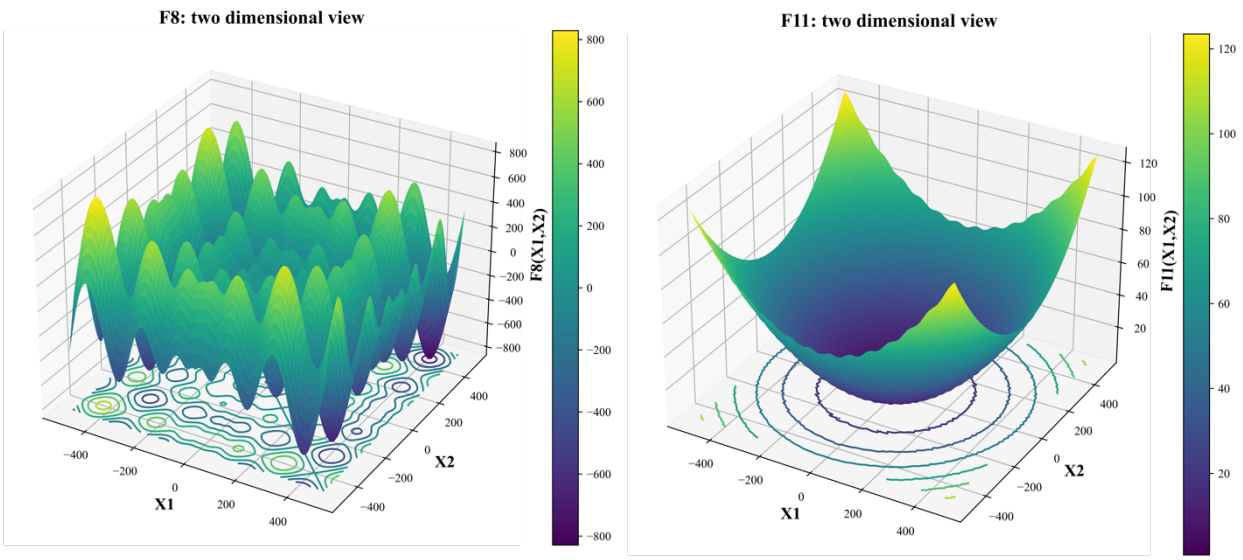

F2: two dimensional view
F2(X1,X2)
X1
X2

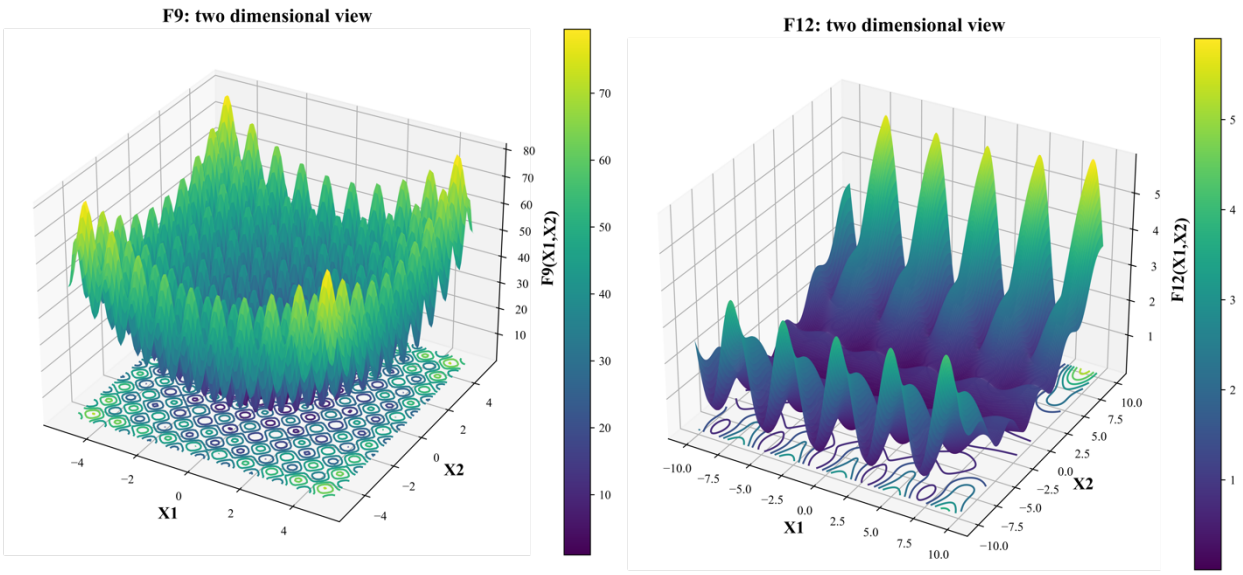

F5: two dimensional view
1e9
1e10
F5(X1,X2)
X1
X2


F3: two dimensional view
1e6
F3(X1,X2)
X1
X2

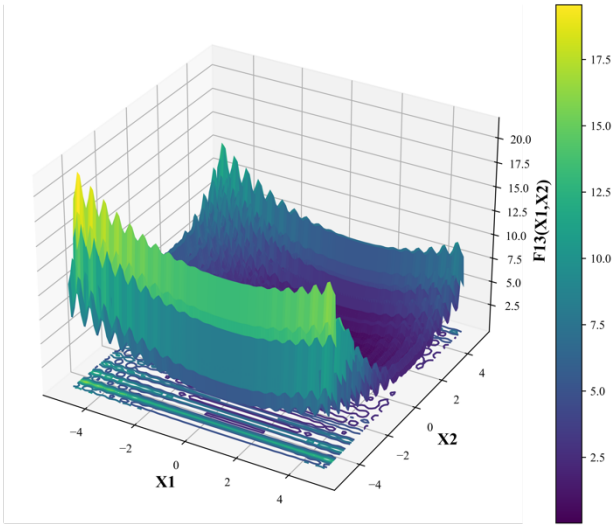

F6: two dimensional view
F6(X1,X2)
X1
X2


F7: two dimensional view
F7(X1,X2)
X1
X2
F10: two dimensional view
F10(X1,X2)
X1
X2

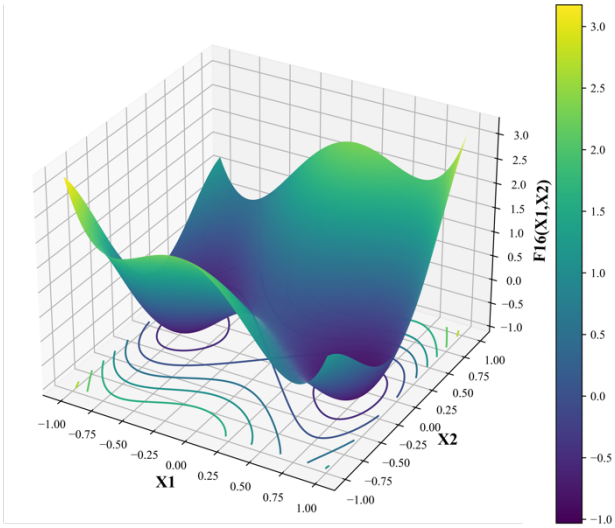

F8: two dimensional view
F8(X1,X2)
X1
X2
F11: two dimensional view
F11(X1,X2)
X1
X2


F9: two dimensional view
F9(X1,X2)
X1
X2
F12: two dimensional view
F12(X1,X2)
X1
X2

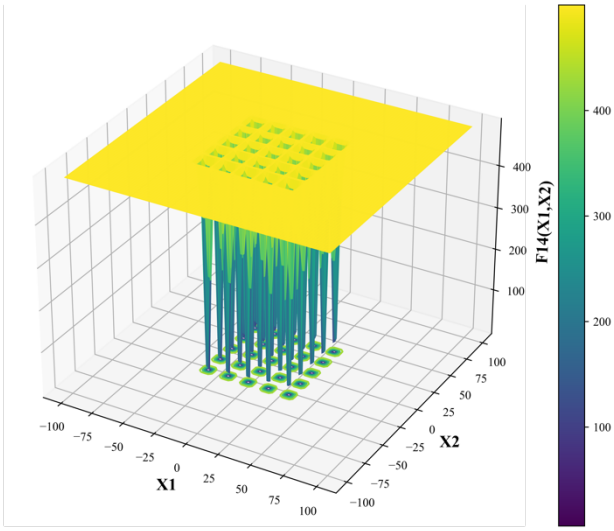
F13: two dimensional view



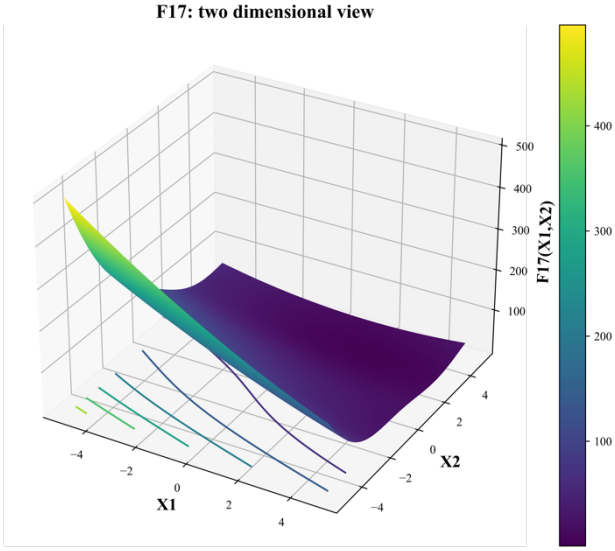
F16: two dimensional view



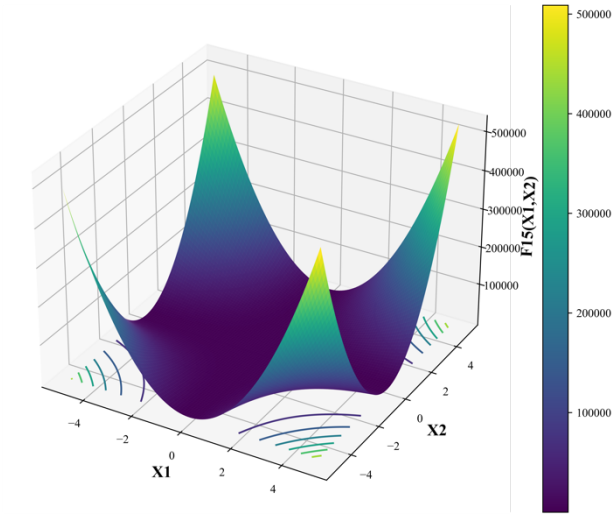
F14: two dimensional view



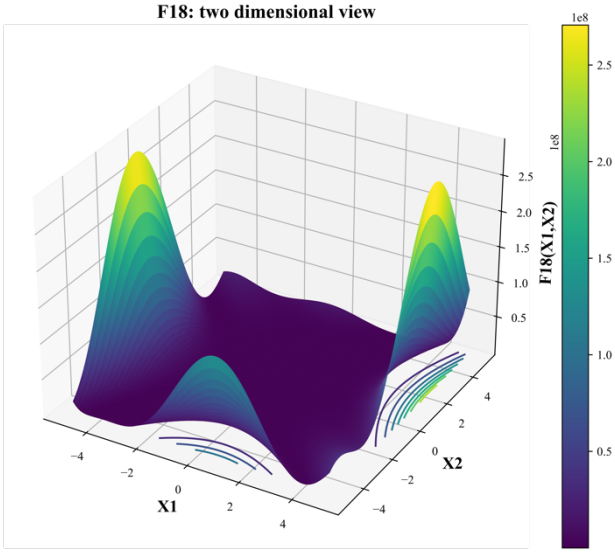
F17: two dimensional view



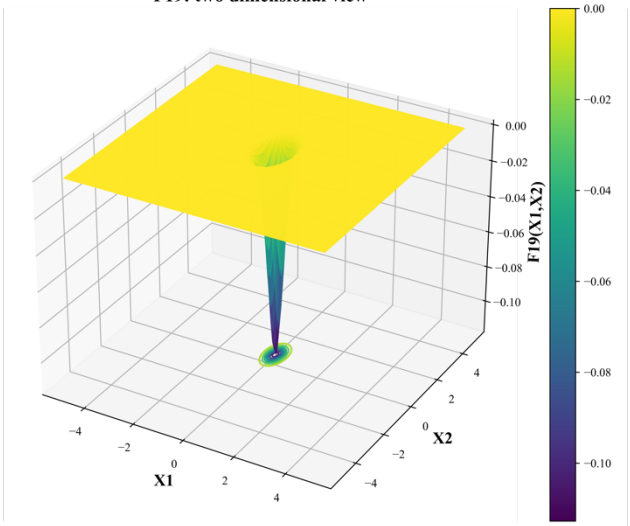
F15: two dimensional view



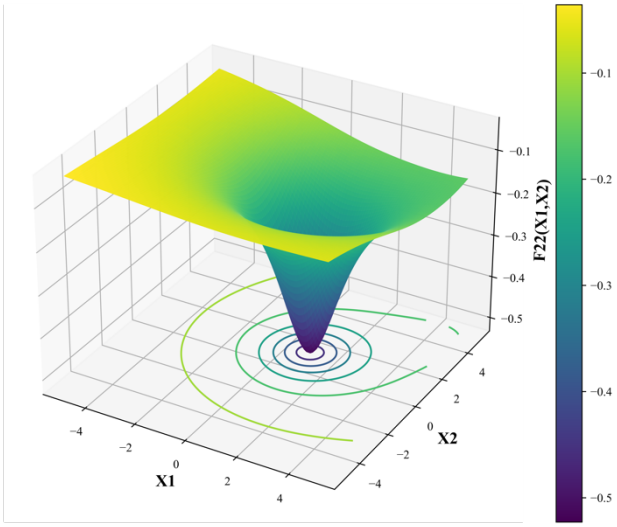
F18: two dimensional view



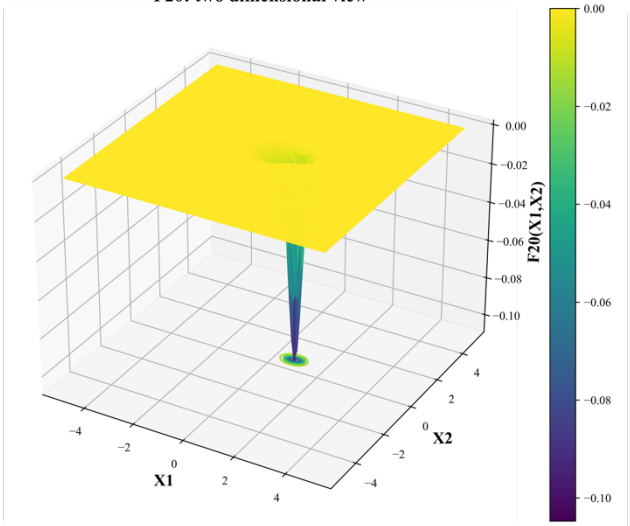
F19: two dimensional view



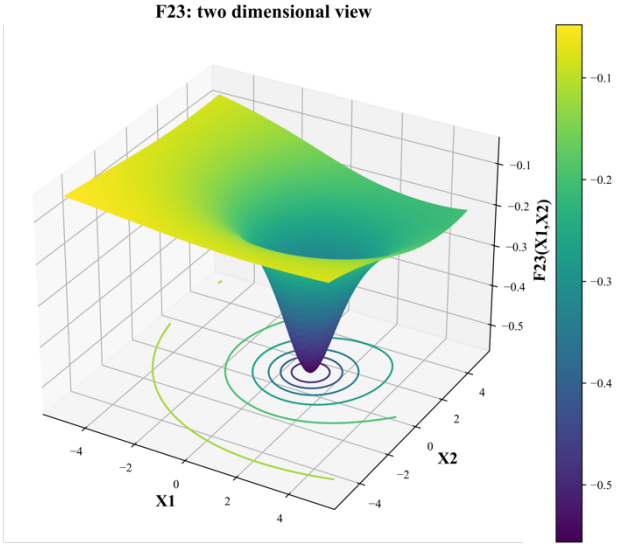
F22: two dimensional view



F20: two dimensional view



F23: two dimensional view



F21: two dimensional view

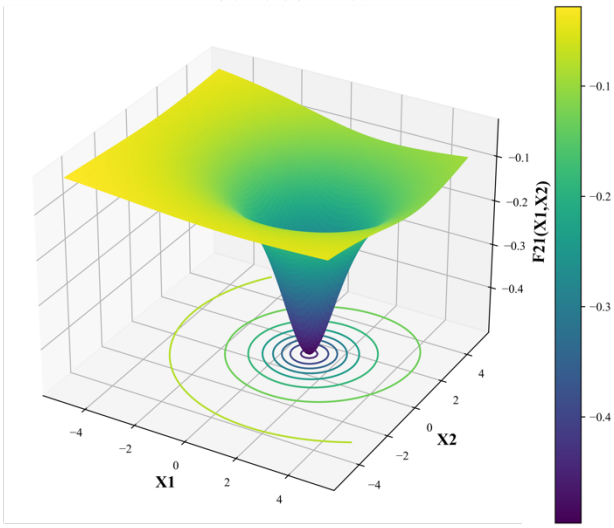


Figure 2 two-dimensional view for CEC-BC-2017

Figure1 has 23 subpictures, and each subpicture is a view of a fitness function in two dimensional 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 data visualization. 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.

Genetic algorithm

Genetic Algorithm (GA) is a population-based stochastic algorithm originated from Darwinian theory of evolution and belonging to the broader category of Evolutionary Algorithms (EA). Each individual's genotype is a solution to a fitness (objective) function [1]. GA consists of three principal bio-inspired operators: 1. Selection, 2. Crossover and 3. Mutation [1-2]. Inspired by that only superior individuals in the population have the chance to produce offspring and pass on their genes, selection operator lays the most important foundation for the GA to be applied to solve optimization problems since that GA always maintains the best solution in each generation and evolves towards better solutions [1,3]. 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 allows two parents to swap their genes with a certain probability, thereby producing a solution between two points [3]. A variety of ways to implement crossover are introduced in the literature such as uniform crossover, half uniform crossover, three parent's crossover, partially matched crossover, cycle crossover, order crossover and position-based crossover [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].

Figure 1(left part) presents the flow 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 factors such as the number of iterations and the complexity of the objective function.

Memetic algorithm

Memetic algorithm (MA) is also part of evolutionary algorithms and MA has been applied in a wide range of real-world optimization problems [21]. Compared to GA uses global search, MA uses local search can be considered as special kind of genetic search in a subspace of GA [22]. MA extends genetic optimization by introducing local search procedures (LSPs) [22-23]. LSPs are 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 [24].

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 the flow of 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.

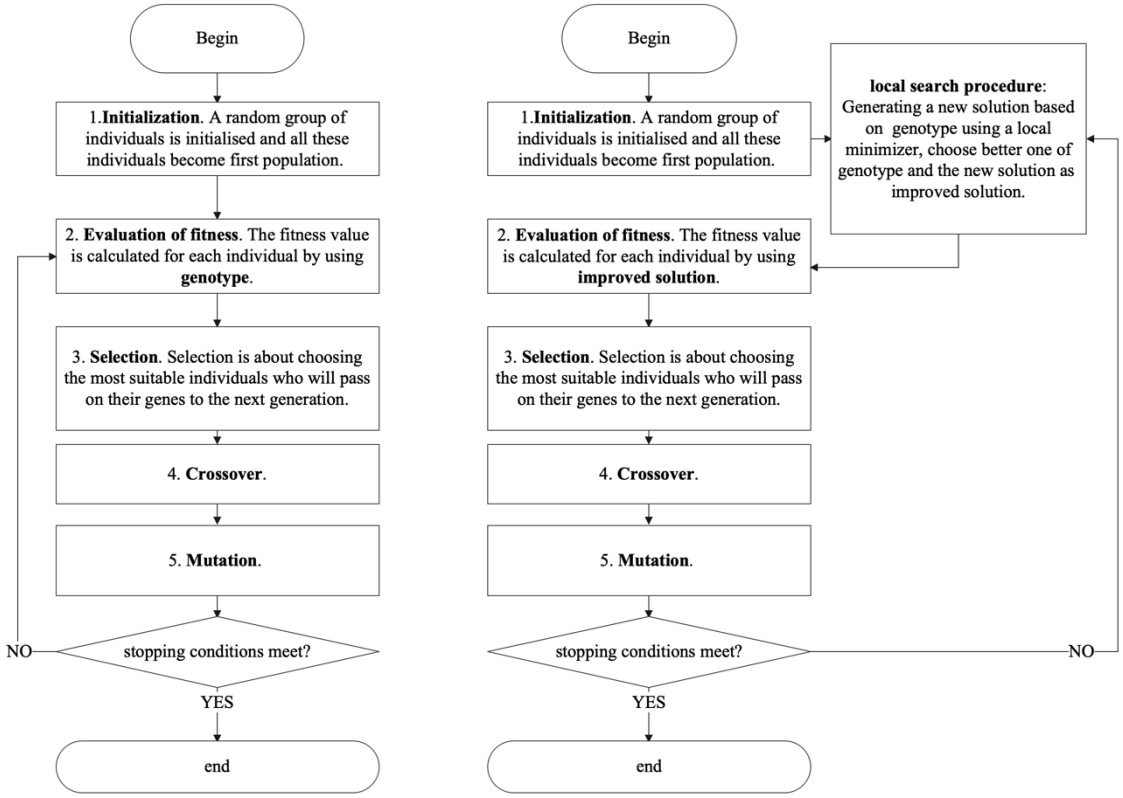


Figure 1 Flowchart of Genetic algorithm(left) and Memetic algorithm(right)

Stopping Criteria for GA and MO are generally organized as follows:

A. Max number of iterations is reached
B. A satisfactory fitness value of objective function has been found
C. The similarity of populations is less than a pre-defined threshold for certain number of iterations

Baldwinian effect and Lamarckian evolution

In order to introduce Baldwinian effect and Lamarckian evolution, it is necessary to begin by distinguishing between genotype and phenotype. Each individual's genotype is just a string of numbers in computation while its biological significance is a complete set of genetic material. A phenotype represents the set of observable characteristics or traits of an individual, and it is an equal-length string of numbers in computation [25-27]. In plain language, a genotype is the hidden DNA in a living organism while a phenotype is the organism's physical body [25].

In biological evolutionary sense, a huge difference exists between genotypes and phenotypes, but in many evolutionary algorithms, they are the identical [25]. Another very important concept is lifetime learning. Lifetime learning will affect the mapping between genotypes and phenotypes. As a general rule, the improvement between genotype and phenotype can be interpreted as the result of a lifelong learning effort [25].

Both the Baldwinian effect 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.

Contribution of this paper

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.

Methods

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 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.

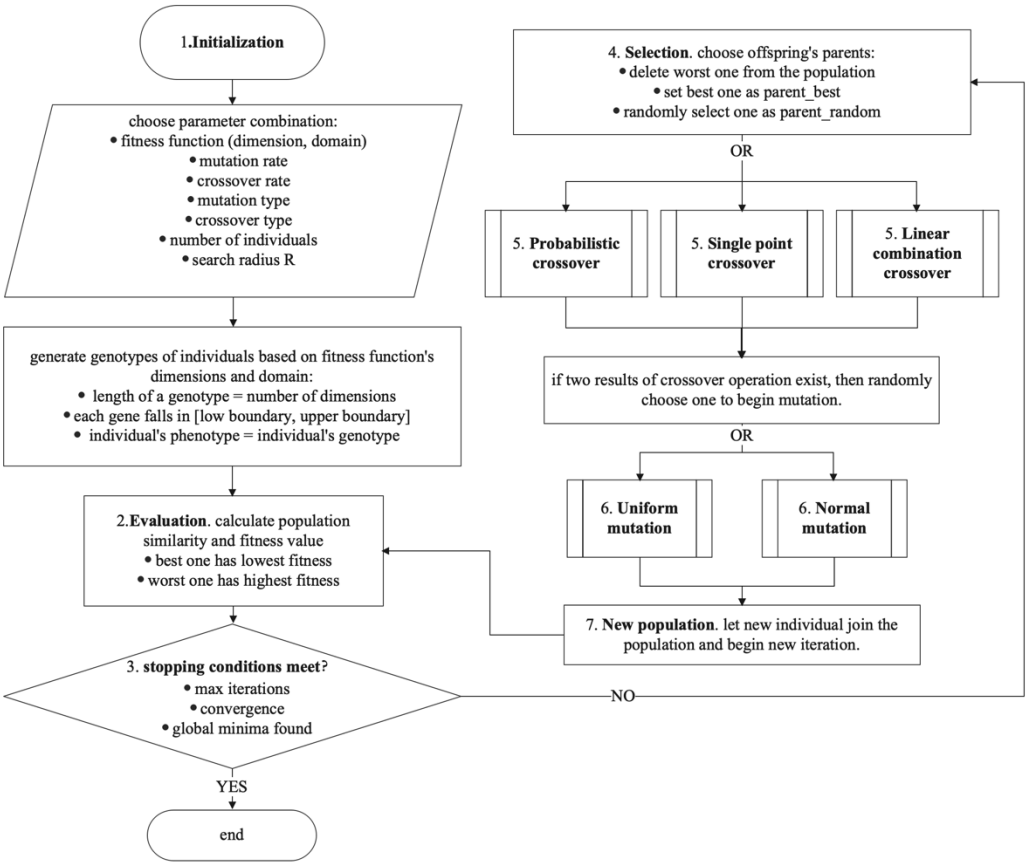


Figure 3 Flowchart of SSGA

SSGA Pseudocode

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 is less than or equal to global minima opt, 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 , each dimension of F is F_i, domain of F is $[F_{low},F_{upper}]$
4. mutation rate γ
5. crossover rate δ
6. mutation type M_t
7. crossover type C_t
8. number of individuals β
9. search radius R
10. global minima opt

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)

end

If mutation type M_t = "Normal mutation":

do Normal mutation (mutation rate γ,search radius R, crossover result)

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 3 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.

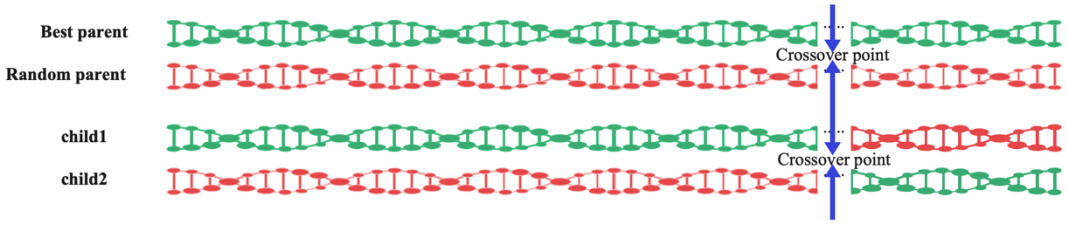


Figure 2 Single point crossover

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

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

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

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 parent R_{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.

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

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

C. Linear combination Crossover

Given crossover rate δ, best parent B_{parent}, random parent R_{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.

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

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, $-a = b = 3 * R * \left(F_{upper} - F_{low}\right)$ 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 = $R * \left(F_{upper} - F_{low}\right)$

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.

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
28.