On the performance of standard nature-inspired algorithms in solving non-stationary optimization problems

Kaiwang Cui[1], and *Evgenii Sopov*[1,2,*]

[1]Siberian Federal University, Institute of Space and Information Technology, Krasnoyarsk, Russia
[2]Reshetnev Siberian State University of Science and Technology, Institute of Informatics and Telecommunications, Krasnoyarsk, Russia

Abstract. Non-stationary optimization problems are a very important class of problems in many practical applications. These problems are characterized by objective functions and constraints that change with time or environmental conditions, so the optimization solution also needs to be dynamically adjusted accordingly. Many algorithms from the field of evolutionary and biology inspired computation are known as an effective approach for dealing with hard optimization problems in changing environments, that is the result of modelling of self-organized systems in nature and evolution in the biology. Natural systems always exist in the changing environments. This article aims to compare the performance of three common nature-inspired techniques, namely genetic algorithm (GA), particle swarm optimization (PSO) algorithm, and differential evolution (DE) in their standard implementation when solving non-stationary optimization problems, so as to provide a reference and rationale for subsequent selection of appropriate algorithms and improvements.

1 Introduction

Many real-world computational and mathematical problems are stated as optimization problems. Both, the objective and constraints are analysed in conditions that can vary over time. In other words, one has to search for the global optimum in the dynamic environment, and the stated optimization task is hard and important. Once a change appears, it can affect the landscape of the objective function, can lead to another problem instance, can reshape the feasible set by changing constraints and so on. In real-world problems, this appears, for example, because of the arrival of additional tasks and requirements, the breakdown of operations, the change of global environmental conditions, and the variance of budgets and resources [1,2]. As the result, the position and value of the global optimal solution to the problem are changed over time and should be tracked.

Many papers define the considered optimization tasks in the dynamic environment as optimization problems that change over time, dynamic problems, or time-dependent problems. The most common name is dynamic optimization problems or DOPs. DOPs are

* Corresponding author: evgenysopov@gmail.com

considered as a special class of dynamic global optimization problems, which are solved online by an optimization algorithm [3].

Evolutionary algorithms and swarm intelligence are effective approaches for dealing with DOPs, because they model some natural self-organized processes from biological evolution, which always evolves and adapts in changing environments [3]. They gradually search for optimal solutions by simulating operations such as natural selection, crossover, and mutation. It has received widespread attention because it is simple, effective and applicable to a variety of problems. However, traditional evolutionary algorithms have some challenges in dealing with non-stationary optimization problems.

In non-steady optimization problems, evolutionary algorithms need to have the characteristics of strong adaptability, fast convergence, and strong ability to jump out of local optimal solutions. Therefore, choosing an appropriate evolutionary algorithm is crucial for solving non-stationary optimization problems. This paper will focus on comparing the performance of three common evolutionary algorithms: genetic algorithm (GA), particle swarm optimization (PSO) algorithm, and differential evolution (DE), which are applied for solving non-stationary global "black-box" optimization problems. We will analyse the performance of these approaches in their standard implementation with no modification for DOPs. The results of the analysis will demonstrate the ability of the standard algorithms to adapt the environmental changes, and will supply us with the baseline for further researches. Finally, it will provide the reference and rationale for selecting appropriate algorithms and improvements for subsequent research.

2 Simulation of the non-stationary environment using the moving peaks benchmark

The Moving Peak Benchmark (MPB) is a dynamic optimization problem that simulates the changing landscape of a fitness function. It is designed for evaluating the performance of optimization algorithms in changing environments and assess their robustness and adaptability [4].

The MPB is characterized by a randomly generated set of peaks in the solution space, each peak is determined by a height and position. As the optimization process proceeds, these peaks move or change height in the solution space, simulating the dynamic changes of the landscape of the objective function. The goal of an optimization algorithm is to find the global optimal solution or approximately optimal solution under the changing peak environment.

The specific content of the MPB problem is as follows:

Problem Setup: The benchmark can be defined in a 2-dimensional or multi-dimensional search space. At the same time, the range and dimensions of the search space can be customized based on the problem requirements.

Peak Configuration: A set of peaks is randomly generated within the search space. Each peak is defined by its height and position. The number and height of the peaks can be adjusted to control the complexity of the problem.

Fitness Function: The fitness function determines the estimation of the quality of a solution. In MPB, the fitness function is composed of the contributions of all the peaks. The objective value of a solution depends on the distance and the height relationship with each peak.

Peak Movement: The key aspect of the Moving Peak Benchmark is the movement of the peaks. The movement can be random, linear, or based on a specific function. The movement rules govern the speed and complexity of the changing peak environment.

Optimization Algorithm: Optimization algorithms are used to search for the gloabal optimum or near-optimal (or so called, the best-found) solutions in the changing peak

environment. Many common optimization techniques, such as GAs, PSO, or simulated annealing (SA), can be applied for solving MPB problems.

Termination Criteria: The termination criteria define when the optimization algorithm stops. It can be based on reaching the maximum number of generations, stabilizing peak positions, or achieving some predefined fitness value. Once the termination criteria are met, the algorithm stops and outputs the best or approximate best solution. Setting the termination criteria is not only to stop the algorithm in time when problems arise, but also takes into account the performance of the equipment. If the algorithm continues to be executed indefinitely, it will cause certain problems for the statistical experiment results.

Currently, researchers have developed a variety of algorithms to solve MPB problems, such as Estimation of Distribution Algorithms (EDA), Evolutionary Strategies, etc [5,6]. However, few people use the most basic standard nature-inspired algorithm for MPB problems.

3 Description of nature-inspired techniques

Evolutionary algorithm (abbreviated as EA) usually addresses to a huge set of nature-inspired algorithms, but not one specific implementation. The main evolutionary loop in EAs is inspired by the evolutionary operations of spices in nature [7]. An EA generally consists of the following standard operations: coding and decoding into a chromosome, population initialization, a parent crossover and random mutation operators, and management of creating new population from parents and offspring.

Compared with formal techniques based of calculus and numerical methods from mathematical programming, approaches of evolutionary computations usually demonstrate the global convergence with the high reliability in many applications. Additionally, EAs can implement different types of self-organization, self-adaptation, and self-learning. EAs can effectively handle hard global “black-box” optimization problems that are difficult or impossible to solve with traditional optimization approaches. EAs can also deal with many NP-hard optimization problems.

Today, many EAs are widely used in numerical and combinatorial optimization, scheduling, resource allocation, and many other fields [8,9,10].

3.1 Genetic algorithm

Genetic Algorithm (or GA) was first proposed in the 1960s [11]. Professor Holland of the University of Michigan in the United States proposed a search algorithm simulates natural processes of selection, recombination, gene altering, and mutation from the biological evolution. The common GA flow chart is presented in Fig.1.

When constructing the genetic algorithm, according to the composition mode of the chromosome, the problem needs to be converted into a method similar to DNA encoding, and all possible solutions of the problem need to be coded. There are various methods for coding/decoding solutions into chromosomes, including the standard binary encoding and Gray-codes, the real number encoding, etc. They need to be determined according to specific problems.

After the coding is completed, the calculation method of the fitness will be determined first, and what kind of individuals belong to high-quality individuals. By simulating the well-known principal of survival of the fittest spices in nature, selecting the fittest individuals, multiplying, hybridizing and mutating among the selected parental spices, and selecting possible higher-quality individuals from them, continuously looping and iterating for producing the population, which becomes better and better until it reaches the desired fitness value.

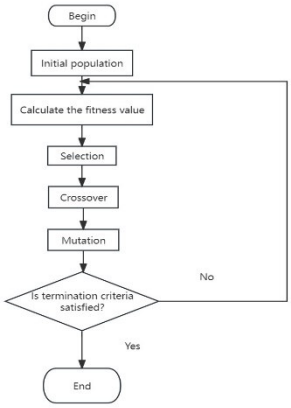

Fig. 1. The common flow chart for GA.

Since individuals of the MPB problem are real numbers, real number coding is selected as the genetic coding of individual chromosomes. Each bit of the chromosome is represented by a real number within a certain range, and then a population with a predefined number of individuals is randomly initialized.

Crossover or genetic recombination operations. Randomly pair excellent individuals in the mating pool, exchange some of their genes, and create offspring solutions that cover new domains in the search space for improving the search process. Since the chromosomes of individuals are encoded with real numbers, real number crossover is used. The crossover operation of the *k*-th chromosome a_k and the *l*-th chromosome a_l at the *j*-th position is:

$$a_{kj} = a_{kj}(1-b) + a_{lj}b \tag{1}$$

$$a_{lj} = a_{lj}(1-b) + a_{kj}b \tag{2}$$

here b is a random number $b \in [0,1]$.

Mutation operations. The individuals in the offspring population that have undergone the crossover operation will change the gene value on one or several gene loci of their own with the mutation probability p_m, and the mutated individuals will be regarded as individuals in the next generation population *x (k+1)*. The mutation operation ensures that the genetic algorithm has local random search capabilities and maintains group diversity. The mutation operation on the *j*-th gene of the *i*-th individual is:

$$a_{ij} = \begin{cases} a_{ij} + (a_{ij} - a_{max}) \cdot f(g), r \geq 0.5 \\ a_{ij} + (a_{min} - a_{ij}) \cdot f(g), r < 0.5 \end{cases} \tag{3}$$

here a_{max} is the upper bound of gene a_{ij}; $f(g)=r_2*(1-g/G_{max})^2$, where g is the current iteration number, r_2 is a random number, and G_{max} is the maximum number of generations; r is a number randomly distributed in the interval [0, 1]; a_{min} is the lower bound of gene a_{ij}.

3.2 Particle swarm optimization algorithm

Particle Swarm Optimization (PSO) is an EA proposed by Dr. Eberhart and Dr. Kennedy in 1995 based on the research on the predatory behavior of bird flocks and fish schools [12]. PSO has the benefits of simplicity and easy implementation, few control parameters, and the good global convergence. It has been widely used in several engineering and scientific fields, such as neural network training and other evolutionary algorithm applications. In recent years, using PSO algorithm and its improved algorithm to solve dynamic optimization problems has gradually become an urgent research issue.

PSO is a population-based algorithm, however, it does not use common evolutionary operators such as selection, recombination, and mutation in the evolutionary process, but treats each individual as a particle with velocity and position vectors in the search space, and completes the search for the optimal solution through the flight of these particles at a certain speed in the search space, which is dynamically adjusted according to the current speed of the particles, the flight experience of the particles themselves, and the overall flight experience of the particles in the population.

The calculation steps of the PSO algorithm are:

Step1: Initialize the population of individuals and create the storage for saving the global optimal and local optimal solutions. Then randomly generate the position and speed of the optimization particles and the position of a certain proportion of monitoring particles;

Step2: Evaluate the population of individuals, obtain the fitness values of all optimization particles, store the current global optimal solution, and calculate the fitness values of monitoring particles;

Step3: Obtain the change degree $\varepsilon(t)$ of the fitness value of the monitored particles before and after the moment. If $\varepsilon(t)>\varepsilon_1$, go to Step 4; if $\varepsilon_2<\varepsilon(t)\leqq\varepsilon 1$, go to Step 5; if $\varepsilon(t)\leqq\varepsilon 2$, then go to Step6;

Step4: Randomly reinitialize the population and optimal solution storage space, generate the position and velocity of the optimized particles, and then go to Step2;

Step5: Generate part of the population according to the Gaussian distribution for the particle positions of the current optimal solution, and randomly generate the optimal particle positions and velocities according to re-initialization, and go to Step2;

Step6: Update the speed and position of each optimization particle;

The speed and position update formulas are as follows:

$$v_{id}(t+1) = w \cdot v_{id}(t) + c_1 r_1\big(p_{id} - x_{id}(t)\big) + c_2 r_2\big(p_{gd} - x_{id}(t)\big) \tag{4}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{5}$$

here $V_{id}(t)$, $X_{id}(t)$ respectively represent the flight velocity and coordinates of particle i at time t; P_{id} represents the individual historical best coordinates of particle i; P_{gd} represents the best coordinates in the group; ω represents the inertia; c_1 and c_2 represent acceleration values; r_1 and r_2 represent numbers randomly distributed in the range [0, 1].

Step7: Determine whether the maximum number of iterations has been reached. If not, go to Step2. If so, the calculation ends.

3.3 Differential evolution algorithm

The differential evolution (DE) algorithm is a global optimization method based on the group of conceptions [13]. It is a stochastic and adaptive search algorithm combining cooperation and competition of individuals in the population. But compared with evolutionary calculation based on using chromosomes, it implements the global search using real numbers, simple mutation operations based on the vector difference and the "one-to-one" selection, which reduces the complexity of common evolutionary operators. At the same time, the specific memory features of DE are very useful for tracking the search situation in the dynamic environment and for adjusting its search strategy. DE demonstrates the global convergence and high reliability. DE does not require the use of comprehensive information about the properties of the objective function and its derivatives. It is suitable for solving many hard optimization problems that are difficult or even impossible to solve using common methods from mathematical programming.

DE is a self-organizing minimization method that requires very little input from the user. Its key idea is different from traditional evolutionary methods, which use predetermined probability distribution functions to determine vector perturbations. The self-organizing program of the differential evolution algorithm uses two randomly selected different vectors

in the population to disturb a vector population. The random perturbation of the population vector can be carried out independently and is therefore parallel. The general steps of DE are:

Initialize the population. The initial population is generated randomly in the predefined domain:

$$x_{ij}^{0} = x_j^L + rand(0,1) * (x_j^U - x_j^L) \tag{6}$$

here $x_{Lj}$and x_{Uj} respectively represent the upper and lower limits of the value range of x_j in the j-th dimension. i=1,2,...,NP; j=1,2,...,D, $x_{0\,i,j}$ represents the j-th dimension of the i-th individual of the 0-th generation, NP represents the number of individuals in the population, and *rand(0,1)* represents a random number uniformly distributed between 0 and 1.

Mutation operations. The differential evolution algorithm realizes individual mutation through differential strategies, which is one of the important signs that the differential evolution algorithm is different from other evolutionary algorithms. For individual x_g^i, a mutant vector is created using the formula below:

$$v_{ij}^{g+1} = x_{r1,j}^{g} + F * \left(x_{r2,j}^{g} - x_{r3,j}^{g}\right) \tag{7}$$

here i=1,2,...,NP; j=1,2,...,D, $r_1,r_2,r_3 \in \{1,2,...,NP\}$, they are randomly generated, and they are not equal to i, so NP must be greater than or equal to four. F is the variation parameter, which is a constant factor between 0 and 2. It controls the gain of the differential variable ($x_{g,r2,j}$-$x_{r3,j}^{g}$). During the evolution process, in order to ensure the feasibility of the solution, it must be judged whether the value of each dimension of the generated mutant individual is within the feasible region. If it is not within the feasible region, then this dimension of the mutant individual needs to be randomly generated within the feasible region.

Crossover operation. In order to make the newly generated individuals have both the inheritance and mutation of the parents, the differential evolution algorithm performs crossover operation between each individual of the population and the mutated individuals generated in the mutation operation. Perform the following operations for individuals $x_{g\,i,j}$ and $v_{g+1\,i,j}$:

$$u_{ij}^{g+1} = \begin{Bmatrix} v_{ij}^{g+1}, if\ rand(0,1) \le CR\ or\ j = j_{rand}, \\ x_{ij}^{g}, otherwise \end{Bmatrix} \tag{8}$$

here CR is the probability of applying crossover and j_{rand} is a random index in $[1,2,...,D]$.

Selection operation. In DE, the greedy way is applied to define the best individual for passing it into new generation. The specific operations are as follows:

$$x_i^{g+1} = \begin{Bmatrix} u_i^{g+1}, if\ f\left(u_i^{g+1}\right) \le f\left(x_i^{g}\right), \\ x_i^{g}, otherwise \end{Bmatrix} \tag{9}$$

All members of the population must be used as target vectors to perform this operation once so that the same number of competitors appears in the next generation. The optimal parameter vectors of each generation are evaluated during the evolution process to record the minimization process. In this way, by using random deviation perturbation to generate new individuals, a very good convergence result can be obtained, which helps to lead the search process to the global optimum.

4 Experimental verification and analysis

This work uses the MPB benchmark for testing, and uses offline error as the performance evaluation criterion. The offline error can be expressed as:

$$\mu = \frac{1}{k}\sum_{i=1}^{k}(h_i - f_i) \tag{10}$$

here h_i represents the height value of the highest peak in the i-th environment; f_i represents the optimal value of fitness that can be obtained by the algorithm in the i-th environment; k represents the total number of environmental changes; μ is the offline error value, which reflects the algorithm's ability to track environmental changes. In theory, $\mu = 0$ is optimal.

Simulate changes in the environment by varying the height, width, and position of multiple peaks. The evaluation function consists of multiple peaks and defined as:

$$F(x,t) = \max_{i=1,2,..,p} \left(\frac{H_i(t)}{1+W_i(t)\cdot\sum_{j=1}^{D}\left(x_j(t)-X_{ij}(t)\right)^2} \right) \tag{11}$$

here $W_i(t)$, $H_i(t)$ represent the width and height of peak i at time t; $X_{ij}(t)$ represents the position of peak i in the j-th dimension at time t; P represents the number of peaks; D represents the position dimension of the peak.

The parameters of the moving peak benchmark problem used are shown in Table 1

Table 1. Moving peak benchmark problem parameters.

Parameter	SCENARIO_1	SCENARIO_2	SCENARIO_3
npeaks	5	10	50
H	[30-70]	[30-70]	[30-70]
W	[0.0001-0.2]	[1.0-12.0]	[1.0-12.0]
move_severity	1.0	1.5	1.0
height_severity	7.0	7.0	1.0
width_severity	0.01	1.0	0.5
period	5000	5000	1000
S	[0-100]	[0-100]	[0-100]
lambda	0.0	0.5	0.5

In Table 1, H is the height range, W is the width range, S is the range of each dimension, and lambda is the distance that a single peak moves as the peak changes.

The population sizes of PSO, GA, and DE algorithms are set to 50, and the environment is changed every 100 iterations, a total of 100 times. Record the data of the best generation in each experiment, repeat each experiment 50 times, and take the average of each type of data as the final result. Running this algorithm produces the following results:

Table 2. Results based on the mean value of offline errors as the evaluation criterion.

SCENARIO	GA	DE	PSO
1	39.0576	0.1026	0.1639
2	26.4113	0.4084	8.6359
3	31.0562	7.6215	13.6597

Table 3. Results based on the mean value of max fitness as the evaluation criterion.

SCENARIO	GA	DE	PSO
1	13.9451	62.3043	49.5373
2	31.4561	51.8344	49.9236
3	36.6198	59.0423	55.9835

Table 4. Results based on the mean value of average fitness as the evaluation criterion.

SCENARIO	GA	DE	PSO
1	8.6335	25.8874	24.5234
2	9.5987	29.2259	17.6466
3	23.6902	40.1487	19.2805

From the overall results, we can see that: the DE algorithm performs better in any scenario, followed by the PSO algorithm, and the GA algorithm performs the worst. From the results in Table 2, it can be found that in scenario 1, the gap between the DE algorithm and the PSO algorithm is not very big. But in scenario two and scenario three, the algorithm gap becomes very large. This is because the number of peaks in Scenario 2 and Scenario 3 increases and the change time interval in Scenario 3 becomes shorter compared to Scenario 1. As a result, the peak function becomes more complex, and the PSO algorithm suffers from local convergence/premature convergence, which ultimately shows the instability of the PSO algorithm. This can also be confirmed from Tables 3 and 4. The difference in fitness values between the DE algorithm and the PSO algorithm is not too big, but the gap is too large in Table 2. This situation does not occur in the DE algorithm. The performance of the GA algorithm can be said to be very poor, and the convergence speed is very slow or even difficult to converge in all experiments.

5 Conclusion

This paper selects the moving peak benchmark as a representative of non-stationary optimization problems as the test function, uses offline error, average fitness and maximum fitness as evaluation criteria, and finally draws conclusions by comparing the performance of three different EAs on the MPB problems: Overall, DE algorithm performs the best and is the most suitable for solving the moving peak benchmark problem. The PSO algorithm is also suitable, but its effect is worse than the DE algorithm. Genetic algorithms are not suitable for solving the moving peak benchmark problem.

Based on the above conclusion, the future will focus on improving PSO and DE algorithms in order to achieve better performance. In addition, consider using a hybrid algorithm to combine the PSO algorithm and the DE algorithm for designing a novel approach to solve the problem.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant No.075-15-2022-1121).

References

1. J. Branke, *Evolutionary approaches to dynamic environments—updated survey*, in: GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, 27–30, (2001)
2. S.X. Yang, Y.S. Ong, Y.C. Jin ey al., *Evolutionary Computation in Dynamic and Uncertain Environments* (Springer-Verlag, Berlin, Heidelberg, 2007)
3. T.T. Nguyen, S.X. Yang, J. Branke, Swarm and evolutionary computation **6**, 1**-**24, (2012)
4. D. Yazdani, J. Branke, M. N. Omidvar, C.H. Li, Generalized Moving Peaks Benchmark, Computer Science, (2021)
5. A.R. Gonçalves, F.J.V. Zuben, *Online learning in estimation of distribution algorithms for dynamic environments*, IEEE Congress on Evolutionary Computation, 62-69, (2011)
6. L. Schönemann, *Evolution strategies in dynamic environments*, Evolutionary Computation in Dynamic and Uncertain Environments, 51-77, (2007)
7. P.A. Vikhar, *Evolutionary algorithms: A critical review and its future prospects*, Information Computing and Communication, 261-265, (2016)
8. T. Bäck, H.P. Schwefel, Evolutionary Computation **1**, 1-23, (1993)

9. D. Davendra, I. Zelinka, R. Senkerik, M. Pluhacek, IBICA **303** (2014)
10. C. Klanke, S. Engell, Computers & Industrial Engineering **174** (2022)
11. J.H. Holland, Genetic algorithm, Scientific American **267**, 66-73 (1960)
12. J. Kennedy, R. Eberhart, *Particle swarm optimization*, Proceedings of ICNN'95-International Conference on Neural Networks, IEEE, 1942-1948, (1995)
13. R.M. Storn, K. Price, Journal of global optimization **11**, 341-359 (1997)
14. Y. Su, G.L. Kong, Modern Electronics Technique **40**, 175-178 (2017)
15. S.X. Yang, H. Richter, *Hyper-learning for population in-cremental learning in dynamic environments*, Proceeding of 2009 IEEE Congress on Evolutionary Computation, 682-689, (2009)