A NEW HYBRID ALGORITHM MPCM FOR SINGLE OBJECTIVE OPTIMIZATION PROBLEMS

NGUYEN TRONG TIEN
Khoa Công nghệ Thông tin, Trường Đại học Công nghiệp thành Phố Hồ Chí Minh,
nguyentrongtien@iuh.edu.vn

Abstract. One of the biggest challenges for researchers is finding optimal solutions or nearly optimal solutions for single-objective problems.
In this article, authors have proposed new algorithm called MPCM for resolving single-objective problems. This algorithm is combined of four algorithms: Mean-Search, PSOUpdate, CRO operator and new operator call Min-Max. The authors use some parameters to balance between the local search and global search. The results demonstrate that, with the participation of Min-Max Operator, MPCM gives the good results on 23 benchmark functions. The results of MPCM will compare with three famous algorithms such as Particle Swarm Optimization (PSO), Real Code Chemical Reaction Optimization (RCCRO) and Mean PSO-CRO (MPC) for demonstration the efficiency.
Keywords. Optimization, single-object problems, algorithm.

1 INTRODUCTION

Recently, the optimal problem has been widely applied in all aspects of human life. So that, many researchers from universities around the world have focused on this field. In the real situation, these problems have been transformed into two basic types of mathematical problems: single-objective and multi-objective. Within the scope of this paper, the authors stressed only on solving a single-objective prob lem. There were a lot of new optimization algorithms such as CRO [1], PSO [2], MPC [3], ACROA [4], DA [6], Spider Monkey [9], Harmony Search [12], Simulated Annealing [19]. From 2011, the CRO was utilized as a medium to solve many problems from different fields even single-objective or multi-objective [3, 8, 18, 20, 21, 22, 23, 25, 26, 29, 30, 31, 32, 33, 34]. In CRO, there is a good search operation that was confirmed as a vital factor [1, 7]. The fast convergence of the algorithm has also been demonstrated through these papers. PSO [2] algorithm has been proven as very good and fast converges on many papers [7, 27, 34] including single-objective or multi-objective.
In recent years, there are a lot of research about PSO [10], Swarm Intelligence [14, 15, 16, 17] and metaheuristics algorithm [11, 12, 13]. Hybridization with PSO to create new algorithms has become popular in this field [24]. The combination of PSO and CRO has been emerged in single or multi-objective of MPC[3], HP_CRO[7], HP_CRO for multi-object[34]. In MPC there exist also an operation called Mean Search (MSO). This operation has also tested as quite effective when searching in spaces where the CRO and PSO are unreachable. The combination of three operators above seems to be perfect. However, as the NFL[5] theory stated, in optimal algorithms, none of algorithm is the best, which means that there isn't an algorithm can solve all the optimal problems.
In order to design a well structured optimization algorithm for solving problems, the algorithm should not only good at exploration and good at exploitation but also good to maintaining diversity. If an algorithm is good at exploration searching then it may be poor at exploitation searching and vice versa. In order to achieve good performances on problem optimizations, the two abilities should be well balanced.
In this paper, the authors proposed a new operation called Min Max Operator (MMO), in combination with operations already existed in CRO [1], PSO [2] and MPC [3] algorithms to solve some single-objective problem. Wherein, MMO, CRO and MSO play the role of exploiting operators, PSO play the role as the exploratory operator. In particular, the combination and the balance between the operations created the effectiveness of this algorithm in solving problems defined in the next part of this paper.
Currently, we can formulate many practical problems as single-objective global optimization problems, which is the key to setting up state variables or model parameters for finding the optimum solution of an objective or cost function. We must determine a parameter vector $\overrightarrow{x^*}$ for solving the cost function $f(\vec{x})(f: \Omega \subseteq \Re^D \to \Re)$ where Ω is a non-empty, large, bounded set that represents the domain of the

variable space. The cost function usually considers D decision variables $\vec{x} = [x_1, x_2, \dots, x_D]$. This means that if $f(\vec{x}) > f(x^*), \forall \vec{x} \in \Omega$ when$max\{f(\vec{x})\} = -min\{-f(\vec{x})\}$, the result of minimization does not degrade the general characteristics.

This paper proposed a new approach to harmoniously combine the exploiting operators and exploratory operators. The main contributions of this paper are summarized as follows:

- A new mathematical operation (MMO) has been created that collects the advantages of the solutions to create a better solution.
- Creating a new algorithm, a new approach that can be applied to other algorithms in improving search capability.
- A new algorithm (MPCM) has been created with a new approach that other algorithms can use to improve search capabilities.
- An algorithm for navigating the search to avoid the local optimization towards the global optimization that improves speed and result of convergence.
- The algorithm (MPCM) is tested on twenty three well-known standard functions. The results showed that the proposed algorithm is highly effective.

The rest of this paper is organized as follows. Section 2 reviews studies that are related to PSO, CRO and MSO. Section 3, analysis and design new Min-Max Operator. Section 4, the design of the main MPCM algorithm. Experimental results on the test functions are provided in Section 5. Finally, Section 6 concludes the work and discusses opportunities for future work.

2 RELATED WORKS

2.1 PSO algorithm

The PSO [2] conducts searches using a population of particles which correspond to molecules in CRO, a population of particles is initially randomly generated. The standard particle swarm optimizer maintains a swarm of particle that represent the potential solutions to problem at hand. Suppose that the search space is D-dimensional, and the position of *j*th particle of the swarm can be represented by a D-dimensional vector, $x_j = (x_{j1}, x_{j2}, ..., x_{jD})$. The velocity (position change per generation) of the particle x_j can be represented by another D-dimensional vector $v_j = (v_{j1}, v_{j2}, ..., v_{tD})$. The best position previously visited by the *j*th particle is denoted as $p_j = (p_{j1}, p_{j2}, ..., p_{jD})$. In essence, the trajectory of each particle is updated according to its own flying experience as well as to that of the best particle in the swarm. The basic PSO [...] algorithm can be described as:

$$v_{j,d}^{k+1} = wv_{j,d}^{k} + c_1 \times r_1 \times \left(p_{j,d}^{k+1} - x_{j,d}^{k}\right) + c_2 \times r_2 \times \left(p_{g,d}^{k+1} - x_{j,d}^{k}\right) \quad (0.1)$$

$$x_{j,d}^{k+1} = x_{j,d}^{k} + v_{j,d}^{k+1} \quad (0.2)$$

Where d $\in$[1, D], $x_{j,d}^{k}$ is the *d*th dimension position of particle *j*th in cycle *k*; $v_{j,d}^{k}$ is *d*th dimension velocity of the particle *j* in *k*th cycle; $p_{i,d}^{k}$ is the *d*th dimension of individual best (*pbest*) of the particle *j* in *k*th cycle; $p_{g,d}^{k}$ is the *d*th dimension of global best (*gbest*) in cycle *k*; c_1 is the cognitive weight and c_2 is a social weight; w is the inertia weight; r_1 and r_2 are two random values similar distributed in the range of [0, 1]. In this paper, update process of PSO is used to explore another part of solution space when the local search caries out many times but cannot get better solution. It can not only avoid premature convergence but also escape from the local minimum.

Algorithm 1: PSOUpdate operator is presented as follows

```
1:  Input: particle jth (or Pop[j])
2:  Vel_value[j] = w × Vel_value[j] + c1 × r1 × (Pbests_value[j] -
Pop_value[j])
      + c2× r2× (Archive_value[gbest] - Pop_value[j])
3:  Pop_value[j] = Pop_value[j] + Vel_value[j]
4:  Constraint handling
5:  Set Numhit = 0
6:  Output: Update the new value for particle jth.
```

At line 2 of the algorithm, expression used to calculate the velocity of each element; $Pbests_{value}[j]$ is the best position that the molecule has received. The index gbest is random in [1, *n*], where *n* is the Archive size. $Archive_{value}[gbest]$ is a value derived from an external population (Archive). $Pop_{value}[j]$ is the current value

of molecule *j*th in population. The line 3 of the algorithm is used to calculate the new position of the *j*th molecule after obtaining its velocity. At line 5, it means that, after using the operator PSOUpdate, it must search by other local search operators. This work helps the algorithm avoid premature convergence. In the rest of this paper, the particle and molecule can be used interchangeability.

2.2 On-wall operator in CRO algorithm

An on-wall ineffective collision [1] occurs when a molecule hits the wall and then bounces back. Some molecular attributes change in this collision, and thus, the molecular structure varies accordingly. As the collision is not so vigorous, the resultant molecular structure should not be too different from the original one. Suppose the current molecular structure is ω. The molecule intends to obtain a new structure $\varepsilon' = Neighbor(\varepsilon)$ in its neighborhood on the PES in this collision. The change is allowed only if

$$PE_\varepsilon + KE_\varepsilon \geq PE_{\varepsilon'} \quad (0.3)$$

We get $KE_{\varepsilon'} = (PE_\varepsilon + KE_\varepsilon - PE_{\varepsilon'}) \times q$ where $q \in [KELossRate, 1]$, and $(1 - q)$ represents the fraction of *KE* lost to the environment when it hits the wall. *KELossRate* is a system parameter which limits the maximum percentage of *KE* lost at a time. The lost energy is stored in the central energy buffer. The stored energy can be used to support decomposition. If (0.4) does not hold, the change is prohibited and the molecule retains its original ε, *PE* and *KE*. The pseudocode of the on-wall ineffective operator is as follows:

Algorithm 2: On-Wall Operator

1. **Input:** A molecule *M* with its profile and the central energy buffer *buffer*.
2. Obtain $\varepsilon' = Neighbor(\varepsilon)$
3. Calculate $PE_{\varepsilon'}$
4. **If** $PE_\varepsilon + KE_\varepsilon \geq PE_{\varepsilon'}$ **then**
5. Generate *q* randomly ∈ [*KELossRate*, 1]
5. $KE_{\varepsilon'} = (KE_\varepsilon = PE_\varepsilon + KE_\varepsilon - PE_{\varepsilon'}) \times q$
6. Update $buffer = buffer + (PE_\varepsilon + KE_\varepsilon - PE_{\varepsilon'}) \times (1 - q)$
7. Update *M* : $\varepsilon = \varepsilon'$, $PE_\varepsilon = PE_{\varepsilon'}$ and $KE_\varepsilon = KE_{\varepsilon'}$
8. **end if**
9. **Output** *M* and *buffer*

In the new algorithm, the authors utilized On-Wall operator to exploit neighbor elements (find the best solution around the initial elements)

2.3 Min-Search Operator in MPC algorithm

The On-Wall operator searchs in the regions of solution space that is near the original solution, while the PSOUpdate operator is used to searches in remote regions. MSO searches [3] in a region that is unexplored by the On-Wall and PSOUpdate operators in the solution space. The MSO algorithm is described as follows:

Algorithm 3: MSO Algorithm

1. **Input:** *x* is a solution, the dimension of the problem is *D*
2. α := random [0, 1]
3. **for** *t* := 1 → *D*
4. Generate random number $b \in [0,1]$
5. **if** ($b > \alpha$)
6. $x'(t) = x(t) + N(0,\sigma^2) * x_{best}(t)$
7. Inspect and handle boundary constraint
8. **End if**
9. **End for**
10. **Output:** solution *x'*

The parameter α is used to determine whether an element in solution x will be altered or not. $N(0,\sigma^2)$ is Gaussian distribution, σ is called Stepsize. x_{best} is the best solution that this molecule has achieved for the time being. The *t*th element will be changed by line 6 when $b > \alpha$. That means, the value of b corresponds to α will determine the choice of elements to be changed by MSO. Moreover, the dependence on x_{best}

helps guide the search direction towards an efficient trajectory. Hence, this process gives us a more efficient operator.

3 ANALYSES AND DESIGN NEW MIN-MAX OPERATOR

The steps executed for finding the particle result elements from the first two elements *particle1* and *particle2* having *n* dimensions as followed:

Step 1: Compute and compare $f_{fitness}(particle1)$ and $f_{fitness}(particle2)$

Step 2: If $f_{fitness}(particle1) > f_{fitness}(particle2)$ in the case of *Min* problem or $f_{fitness}(particle1) < f_{fitness}(particle2)$ in the case of *Max* problem then:

Step 3: The *particle result* is *particle2* and is replaced *k* ($k < n$) elements of the *particle result* by *k* elements in *particle1* as followed:

Step 4: Randomly select *k* elements in *particle1* to replace in the corresponding position in *particle result* so that $f_{fitness}(particle1[t]) < f_{fitness}(particle\ result[t])$ {with *t* runs from 1 to *k*}.

Example: In Figure 1 is a particular problem of the *f1* problem in the table [...] with the dimension $n = 8$, $f1_{min}(x) = \sum_{i=0}^{n} x_i^2$, we have $particle2(x_2)$ with sum of squares $f1_{\text{fitness}}(x_2) = 49.4781$ smaller than $particle1(x_1)$ with the sum of squares $f1_{\text{fitness}}(x_1) = 207.177$. So the algorithm will retain the *particle2* then randomly select some elements (3 elements) in *particle1* (for example, in 3rd, 4th and 5th position), because $f1_{fitness}(x_1[3, 4, 5]) = 1.0621 < f1_{\text{fitness}}(x_2[3, 4, 5]) = 24.5681$ *so* the process of replacing these three elements into *particle result* at the corresponding positions. When $f1_{min}$ *(particle result)* = 25.9721 obtained less than $f1_{min}$ *(particle1)* = 207.177 and $f1_{min}$ *(particle2* = 49.4781.

In Figure 2, also with the problem *f1* but by randomly selecting 3 elements at 3rd, 5th and 7th positions in *particle1*, we have $f1_{\text{fitness}}(x_1[3, 5, 7]) = 0.954 < f1_{\text{fitness}}(x_2[3, 5, 7]) = 31.16$, so the process of replacing the 3 elements of *particle1* into *particle result* at the corresponding positions (3, 5, 7). The result obtained is that the *particle result* having $f1_{min}$ *(particle result)* = 19.2721 is less than $f1_{min}$ *(particle1)* = 207.177 and $f1_{min}$ *(particle2* = 49.4781.

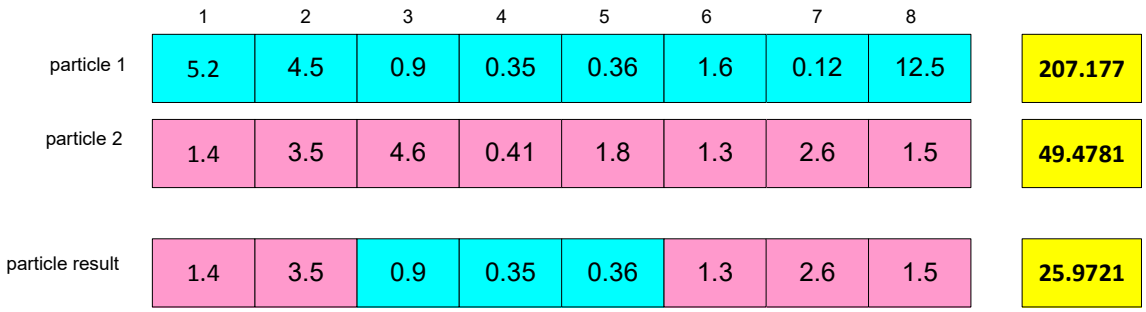

Figure 1: Description of selecting 3 successive element

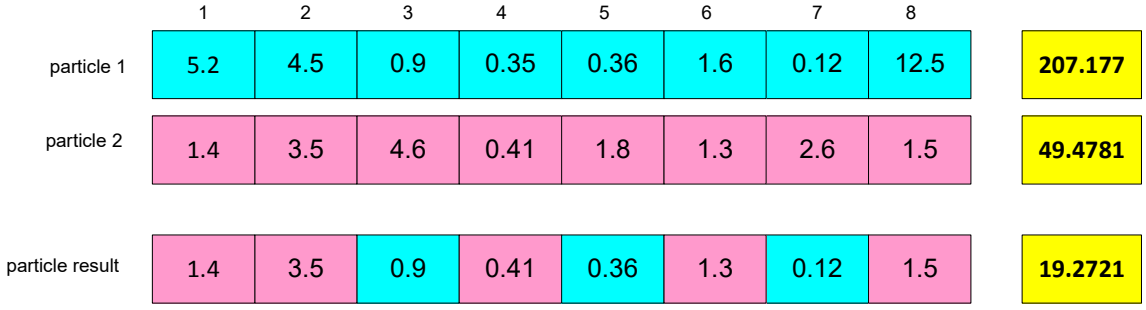

Figure 2: Description of selecting 3 random elements.

The Max-Min algorithm is detailed in Algorithm 4.

Algorithm 4: Max-Min Operator Algorithm

1. **Input:** The Solution *particle1, particle2*, the dimension of the problem is *D.*
2. S1 ←$f_{fitness}$*(particle1)* ; S2 ←$f_{fitness}$*(particle2)*;
3. **if** S1 > S2 **then**
4. *Particle3* ←*particle2* (**For** i =1 **to** *D* **do** *Particle3[i]* ←*particle2[i])*
5. Generate int k (0 < *k* < *D*); { *k* elements need replacing }
6. *i* := 1;
7. *int A[k]*; {Creating an array of k elements for storing the position will change in *Particle3*}
8. **while (** *i* < *k* **)do**
9. Generate int t (*0* < *t* < *D* and *A[t]* ≠ *A[ahead]*)
10. $S_{par1} = S_{par1} + f_{fitness}$*(particle1[t]);*
11. $S_{par3} = S_{par3} + f_{fitness}$*(particle3[t]);*
12. *A[i]:=t;*
13. *i* ++;
14. **end while**
15. **if** *(* $S_{par1} < S_{par3}$ *)* **then**
16. **for** *i* = 1 **to** *k* **do**
17. *Particle3[A[i]]* ←*particle1[A[i]]*
18. **end for**
19. **end if**
20. **Output:** solution *Particle3;*

4 THE MAIN MPCM ALGORITHM

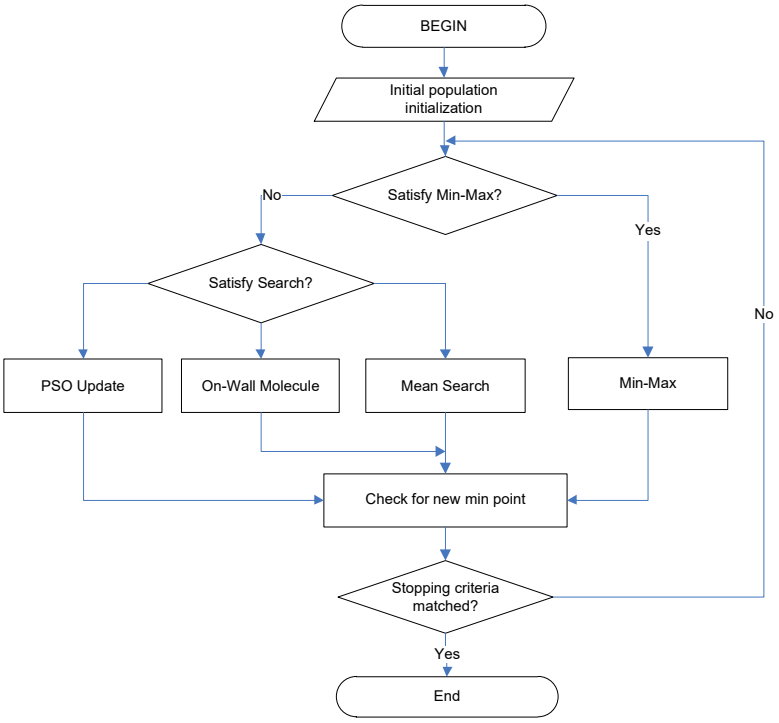

Figure 3: Flowchart of the MPCM algorithm

The procedure of the proposed MPCM algorithm can be summarized as follows: Algorithm consists of 3 stages:

Stage 1 (initialization stage): Including initialization values for the input parameters of the algorithm. Where *Popsize* is the initial molecule set size, *StepSize* is the parameter that determines the modification of the random molecule value in the On-Wall operator; α, γ, ∂ are parameters that control the selection of one of the PSOUpdate, On-Wall, Mean-Search or Min-Max operators to change the molecule to a new molecule. Initially, these parameters are set to 0, meaning that the molecule has not been changed by PSOUpdate, On-Wall, or Mean-Search.

If these parameters are equal 1 that means they have been modified by three operators PSOUpdate, On-Wall, Mean-Search (the left side of the ABC diagram) and then the Min-Max operator will be executed (the right side of the ABC diagram). That means the molecule must be transformed by three operations PSOUpdate, On-Wall, Mean-Search before performing the Min_Max operation; r is the parameter used to store the elements needed to exchange in the Max-Min operator, n is the whole number selected which depends on each problem.

Stage 2 (Reiteration stage): The input of this stage is the selection of a random molecule (M_w) from population set *Pop*. A molecule has three attached control parameters to decide which one it will be manipulated by.

In each iteration, any molecule is transformed into a new molecule through a single operator among four operators (PSOUpdate, On-Wall, Mean-Search, and Min-Max). The selection of which operator to perform depends on the parameters α, γ and ∂ in the molecule. Any molecule has to be transformed through the four operations to find a better solution. An element after being transformed by the three operators PSOUpdate, On-Wall, and Mean-Search (left side of ABC diagram) will be transformed by the Min-Max operator (the right side of Figure ABC). Because of the input of the Min-Max operator is two molecules so before the execution of the Min-Max operation, a random element from M_w in population set Pop is chosen by the algorithm which will create a molecule that has better fitness than the first two solutions.

Stage 3: (ending stage): The algorithm will end if any stop criteria are satisfied and will create the best solution found and its objective function value ($f_{min}(solution)$).

MPCM algorithm has been simulated through algorithm flowchart in Figure 3

Algorithm 5: MPCM Algorithm

```
1.  Input: Problem function f, constraints for f, and dimension D of the problem.
2.  \\Initialization
3.  Assign parameter values to PopSize, StepSize.
4.  ∂, α, γ parameters ← 0;
5.  Assign value n to r.
6.  Let Pop be the set of particle 1, 2,…, PopSize
7.  for each of molecule do
8.      Assign Random(solution) to the particle (particle position) w;
9.      Compute the fitness by f(w);
10. end for
11. \\Iterations
12. while (the stopping criteria not met) do
13.     Select a particle Mw from the Pop randomly;
14. if (γ == 0 or ∂ == 0 or α == 0)
15.     if (α == 0)
16.             PSOUpdate(Mw);
17.             α ←1
18.     else if (γ == 0)
19.             Mean-Search(Mw);
20.             γ ← 1
21.         else
22.             On-Wall(Mw);
23.             ∂ ← 1
```

24. **end if**
25. **else**
26. Select a particle $M_{w'}$ from the population set (*Pop*) randomly;
27. Select *r* element from the *Pop* randomly $M_{w'}$;
28. Max-Min(M_w, $M_{w'}$);
29. $\partial, \alpha, \gamma \leftarrow 0$;
30. **end if**
31. Check for any new minimum solution;
32. **end while**
33. //The final stage
34. **Output**: the best solution found and its objective function value

5 SIMULATION RESULTS

5.1 Experimental setting

The algorithm is coded in Visual C# 2010, and all simulations are performed on the same personal computer with Intel (R) Core (TM) i5-62000U CPU @2.30GHz 2.40GHz and 12 GB of RAM in Windows 10 environment. There can be no search algorithm that outperforms all others on all problems [28].

Table 1: Setting parameters for the representative functions in each category.

Order	Parameter	Category I	Category II	Category III
1	*PopSize*	10	20	100
2	*StepSize*	0.2	1.0	0.5
3	*Buffer*	0	100000	0
4	*InitialKE*	1000	10000000	1000
5	*MoleColl*	0.2	0.2	0.2
6	*KELossRate*	0.1	0.1	0.1

5.1.1 Parameters and Benchmarks

The number of control parameters was reduced, thus, it makes the implementation simple. The parameters from the source code of RCCRO were used directly. All the parameters used in this chapter are presented in Table 1.

In this chapter, our proposed MPCM was tested to solve the test functions used in the paper[1]. The test functions are classified into three categories. The dimensions of the functions in Category I and Category II are both 30. These functions are called high-dimensional functions. The test function name and their dimension size, feasible solution space *S*, and global minimum are also included in it.

(1) High-dimensional Unimodal Functions

This group consists of functions f_1 - f_7 and they are high-dimensional. There is only one global minimum in each of the functions. They are relatively easy to solve when compared with those in the next group.

(2) High-dimensional Multimodal Functions

This group is composed of functions f_8 - f_{13}. They are high-dimensional and contain many local minima. They are considered as the most difficult problems in the benchmarks set.

(3) Low-dimensional Multimodal Functions

This group includes functions f_{14} - f_{23}. They have lower dimensions and fewer local minima than the previous group.

5.1.2 Experiment comparisons

5.1.2.1 Comparisons with some modern algorithms

As can be seen in the paper [1], RCCRO4 is the best version of RCCRO. However, it shows worse results than MPC [3], which shows the best results in the versions of the hybrid algorithm. The PSO was proposed to optimize numerical functions, which has effective search ability [2]. The results of MPCM were compared with those of RCCRO4, MPC and PSO in this section. For each function, 50 runs were done, and the averaged computed value (Mean) and standard deviation (StdDev) were recorded. The four algorithms were ranked over the functions, and the average ranks for every category were obtained. The outcome was tabulated in Table 2 to Table 3 .

Table 2: Optimization computing results for f_1 to f_7.

	FEs		**MPCM**	**PSO**	**MPC**	**RCCRO4**
f_1	150000	Mean	1.12E-300	3.69E-37	6.96E-250	7.14E-07
		StdDev	3.02E-200	2.46E-36	1.99E-157	2.14E-07
		Rank	**1**	**3**	**2**	**4**
f_2	150000	Mean	4.18E-75	7.14E-24	2.15E-60	2.06E-03
		StdDev	2.01E-75	2.81E-23	4.04E-60	3.52E-04
		Rank	**1**	**3**	**2**	**4**
f_3	250000	Mean	4.43E-250	1.55E-03	5.71E-199	2.63E-07
		StdDev	6.69E-00	5.91E-3	7.04E+00	5.93E-08
		Rank	**1**	**4**	**2**	**3**
f_4	150000	Mean	5.73E-21	4.43E-01	9.88E-12	9.88E-03
		StdDev	1.83E-20	2.56E-01	2.62E-12	5.58E-04
		Rank	**1**	**4**	**2**	**3**
f_5	150000	Mean	2.75E+01	2.22E+01	8.01E+01	6.31E+01
		StdDev	1.60 E-01	3.50 E+01	3.49 E+01	5.59 E+01
		Rank	**2**	**1**	**4**	**3**
f_6	150000	Mean	0	0	0	0
		StdDev	0	0	0	0
		Rank	**1**	**1**	**1**	**1**
f_7	150000	Mean	2.08E-03	8.18E-03	3.68E-03	8.61E-03
		StdDev	9.96E-04	2.87E-03	1.10E-04	3.39E-03
		Rank	**1**	**3**	**2**	**4**
Average rank			**1.14**	**2.71**	**2.14**	**3.14**
Overall rank			**1**	**3**	**2**	**4**

From the average ranking shown in Table 2 to Table 3, MPCM shows the best result. Therefore, MPCM can be used to solve the benchmark problems. Note that no general algorithm can work best on all the functions. As can be concluded, nearly each algorithm can outperform the others on specific functions: MPCM performs best on $f_1, f_2, f_3, f_4, f_7, f_8, f_9, f_{10}, f_{16}, f_{19}, f_{21}, f_{22}$ and f_{23}. PSO works best on $f_5, f_{11}, f_{15}, f_{16}$, *and* f_{17}. MPC performs best on f_{12}, f_{13} and f_{14}.

Table 3: Optimization computing results for f_8 to f_{13}

	Fes		**MPCM**	**PSO**	**MPC**	**RCCRO4**
f_8	150000	Mean	-1.24E+04	-1.25E+03	-1.00E+04	-1.15E+04
		StdDev	2.47E+02	1.61E+02	6.74E+02	2.92E+01
		Rank	**1**	**4**	**3**	**2**
f_9	250000	Mean	0	6.20E+01	1.78E-00	1.81E-03
		StdDev	0	1.31E+00	5.50E-00	5.64E-04
		Rank	**1**	**4**	**3**	**2**
f_{10}	150000	Mean	4.44E-16	6.88E-03	2.25E-15	2.93E-03
		StdDev	0	2.33E-02	3.41E-00	3.83E-04
		Rank	**1**	**4**	**2**	**3**
f_{11}	150000	Mean	1	1.04E-01	1.00E+00	1.00E+00
		StdDev	0	1.04E-01	1.62E-11	4.37E-07
		Rank	**2**	**1**	**2**	**2**
f_{12}	150000	Mean	9.80E-04	1.08E-02	3.45E-18	3.45E-01
		StdDev	1.75E-04	2.09E-02	3.60E-17	2.11E-01
		Rank	**2**	**3**	**1**	**4**
f_{13}	150000	Mean	1.12E-02	1.04E-02	2.70E-13	1.80E-05
		StdDev	2.76E-03	1.59E-01	1.19E-13	2.30E-05

		Rank	**4**	**3**	**1**	**2**
Average rank			**1.83**	**3.16**	**2**	**2.5**
Overall rank			**1**	**4**	**2**	**3**

Table 2 gives the results for high-dimensional unimodal functions. According to the overall rank in Table 2, MPCM outperforms the rest of the algorithms. The standard deviations of MPCM are always less than those of PSO, RCCRO4 and MPC. However, MPCM gives poorer results in solving f_{13}. In f_6 MPCM can obtain the global minima 0 and their standard deviations are 0. It shows that MPCM is robust in solving these test functions. For f_4 and f_7, our algorithm gives the best results although it cannot obtain the global minima, and the standard deviation is the least. However, MPCM gives poorer performance than PSO and HP-CRO4 in solving f_5.

Table 2 shows that MPCM gets the highest overall rank. MPC ranks second then followed by PSO, and RCCRO4 ranks the lowest. In general, MPCM is efficient in solving high-dimensional unimodal functions.

Table 4: Optimization computing results for f_{14} to f_{23}

	FEs		**MPCM**	**PSO**	**MPC**	**RCCRO4**
f_{14}	7500	Mean	9.98E-01	9.98E-00	2.81E-01	3.56E+00
		StdDev	8.71E-10	6.46E-01	3.04E-11	1.81E+00
		Rank	**2**	**4**	**1**	**3**
f_{15}	250000	Mean	3.80E-04	2.05E-04	5.35E-04	6.82E-04
		StdDev	6.12E-05	5.62E-04	1.07E-04	8.56E-05
		Rank	**2**	**1**	**3**	**4**
f_{16}	1250	Mean	-1.01E+00	-1.01E+00	-0.95E+00	-0.966E+00
		StdDev	1.89E-02	2.19E-02	6.84E-03	6.45E-01
		Rank	**1**	**1**	**4**	**3**
f_{17}	5000	Mean	3.99E-01	3.98E-01	4.00E-01	3.99E-01
		StdDev	9.79E-04	3.96E-02	1.20E-02	1.39E-02
		Rank	**2**	**1**	**4**	**2**
f_{18}	10000	Mean	3.00E+00	3.00E+00	3.05E-01	3.03E+00
		StdDev	1.85E-04	6.14E-03	3.89E-05	4.20E-02
		Rank	**2**	**2**	**1**	**4**
f_{19}	4000	Mean	-3.86E+00	-3.86E+00	-3.86E+00	-3.82E+00
		StdDev	2.80E-03	3.73E-03	9.35E-03	2.97E-04
		Rank	**1**	**1**	**1**	**4**
f_{20}	7500	Mean	-3.26E+00	-3.25E+00	-3.32E+00	-2.41E+00
		StdDev	3.70E-02	2.41E-02	2.74E-01	4.08E-03
		Rank	**2**	**3**	**1**	**4**
f_{21}	10000	Mean	-9.57E+00	-9.50E+00	-8.53E+00	-1.23E+00
		StdDev	7.37E-01	3.85E-00	2.64E-01	7.52E+00
		Rank	**1**	**2**	**3**	**4**
f_{22}	10000	Mean	-9.96E+00	-9.75E+00	-9.49E+00	-1.26E+00
		StdDev	4.83E-01	4.77E-01	2.07E-01	5.95E+00
		Rank	**1**	**2**	**3**	**4**
f_{23}	10000	Mean	-1.00E+01	-9.91E+00	-8.53E+00	-2.08E+00
		StdDev	6.05E-01	5.31E-01	3.22E-01	1.04E+00
		Rank	**1**	**2**	**3**	**4**
Average rank			**1.5**	**1.9**	**2.4**	**3.6**
Overall rank			**1**	**2**	**3**	**4**

Table 3 gives the results for high-dimensional multimodal functions. MPCM outperforms PSO, MPC and RCCRO4. The performance of MPCM is the best in solving all the functions, except for f_{11}, f_{12} and f_{13}. For f_9, MPCM can obtain the global minimum and its standard deviation is 0. MPCM also gives the best performance when solving f_8, f_9, and f_{10}.

Table 3 supports the conclusion that MPCM obtains the highest overall rank, followed by MPC, RCCRO4 and PSO. Thus, MPCM is efficient in solving high-dimensional multimodal functions.

Table 4 gives the results for low-dimensional multimodal functions. MPCM also outperforms the other algorithms. It gives the best performance when solving f_{16}, f_{19}, f_{21}, f_{22} and f_{23}. MPC can get the best rank when solving f_{14}, f_{19} and f_{20}. For f_{17}, PSO ranks first.

From Table 4, it can be concluded that MPCM ranks first, PSO ranks second, MPC ranks third, and followed by RCCRO4. In other words, MPCM is also efficient in solving low-dimension multimodal functions.

5.2 Experimental results

Figure 4 shows the results of 50 independent runs of 4 problems f_1, f_2, f_3 and f_4 for the four algorithms PSO, MPC, RCCRO and New (MPCM). In Figure 4 (a) shows:

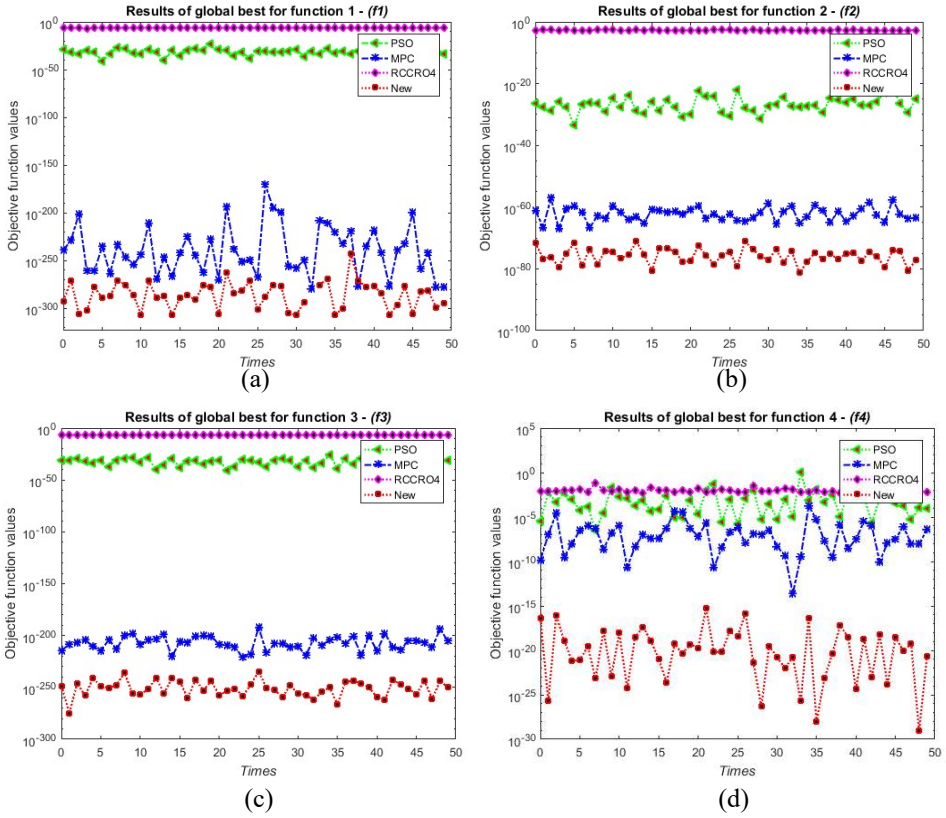

Figure 4: Global-best results of PSO, MPC, RCCRO4 and MPCM for f_1(a), f_2(b), f_3(c) and f_4(d) of 50 runtimes

About standard deviation: the results of RCCRO are more stable than the other 3 algorithms, the result difference between runs is not large, we can conclude that this is RCCRO algorithm for the highest stability in 4 algorithms shown in the figure. The second stable result belongs to PSO, followed by MPCM. The standard deviation of the MPC algorithm is the largest.

About Global-best results: In **Error! Reference source not found.**, for all the selected functions, the RCCRO algorithm gave the worst results, followed by PSO, MPC, and MPCM for the best results. Although the MPCM 50 runs were superior to the other three algorithms, the MPC algorithm that had a few runs gave better results than the MPCM. Even so, the better number of times still belongs to MPCM. In addition, according to the data in the table Table 2 to Table 4, MPCC is superior to the PSO, RCCRO

and MPC algorithms. Based on these important results, we conclude that the MPCM algorithm is superior to the other algorithm.

In addition, according to the data in the Table 2 to Table 4, MPCC is superior to the PSO, RCCRO and MPC algorithms. Based on these important results, we conclude that the MPCM algorithm is superior to the other algorithm.

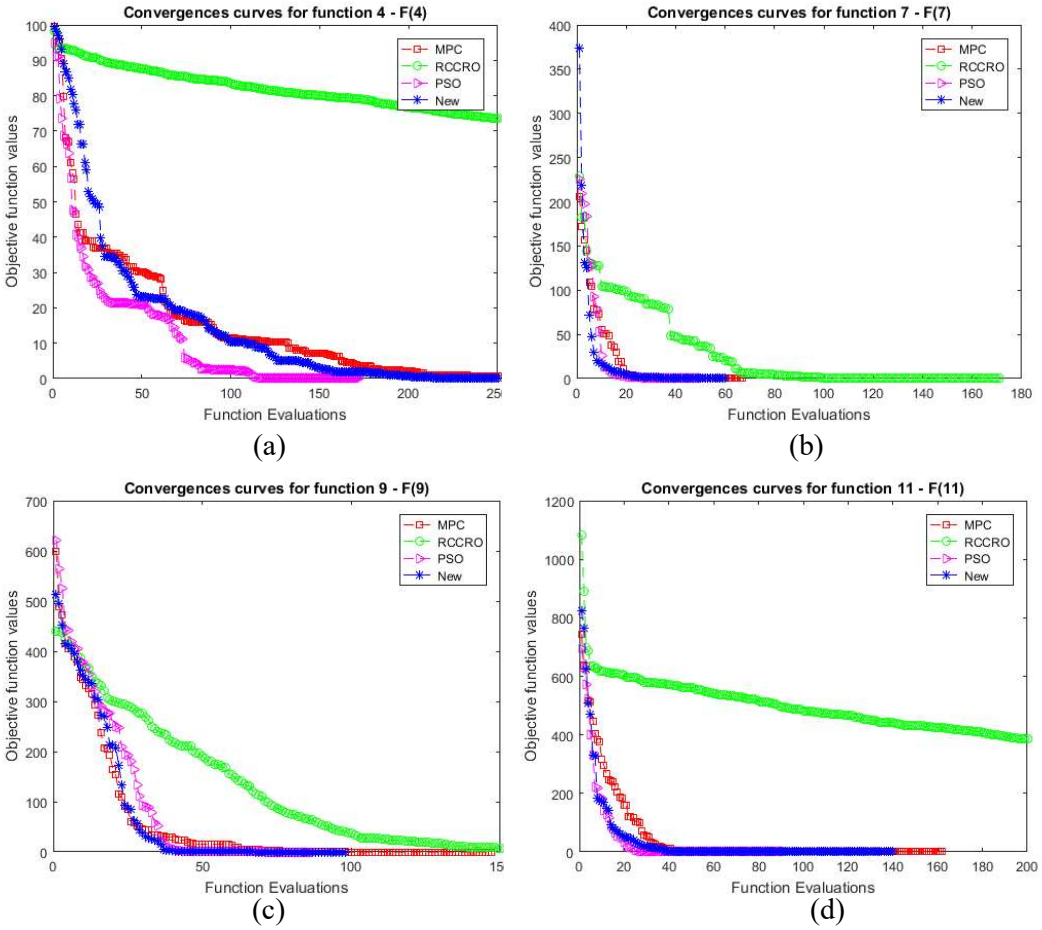

Figure 5: Convergence curves of MPC, RCCRO, PSO and MPCM for f_4(a), f_7(b), f_9(c) and f_{10}(d)

Figure 5 illustrates the convergence of algorithms MPC, RCCRO, PSO and MPCM for 4 randomly selected problems in functional groups: f_4, f_7 belong to group I, f_9 and f_{11} belong to group II.

In general, Figure 5 (a), 5 (b), 5 (c) and 5 (d), the convergence of RCCRO is slowest. It is also easy to see that, in most problems, PSO is the fastest convergence of near-optimal value among 4 comparison algorithms.

In Figure 5 (a), the values of solutions of RCCRO change step by step, however, the change is very small, it means that RCCRO converges slowly, the remaining 3 algorithms converge. condenser very fast. PSO has quite large changes in the root value through the iterations, so PSO converges to the nearest point near 0, then MPCM algorithm and finally MPC algorithm. Similar to Figure 5 (a), in Figure 5 (b), PSO has the fastest convergence rate to the nearest point, then to MPCM, MPC and finally RCCRO.

Looking at Figure 5, it is easy to see that the convergence speed of the illustration in Figure 5 (a) shows that the gradual convergence of the algorithms is very different, for RCCRO, most of all the algorithms in the figure. For the remaining 3 algorithms, the change over step by step is very large, converging very quickly. The fastest is the PSO, then the MPCM and MPC. Figure 5(b) shows that, for this f_7 problem, the algorithms converge very quickly, the fastest is still PSO, then MPCM, MPC and RCCRO.

6 CONCLUDING REMARKS AND FUTURE WORK

In this paper we proposed a novel algorithm to solve global optimization problems. This algorithm is based on the balancing of global and local search strategy. MPCM is conceptually simple and relatively easy to implement. MPCM can tackle a wide range of different continuous optimization problems and has the potential to be employed to solve real-world problems.

In order to evaluate the performance of MCPM, we adopted a set of 23 benchmark functions which cover a large variety of different optimization problem types. We compared MPCM with the stateof-the-art optimization algorithms, namely, PSO, RCCRO and MPC. These algorithms have been employed to solve a large set of different benchmark optimization functions and real-world problems, and demonstrated outstanding performance.

The results show that the performance of MPCM is outstanding compared with the above listed algorithms in all three different groups of functions. This conclusion was supported by both the simulation results and the statistics of the simulation data.

REFERENCES

[1] Lam, A. Y., Li, V. O., & Yu, J. J. (2012). Real-coded chemical reaction optimization. *IEEE Transactions on Evolutionary Computation, 16*(3), 339–353.

[2] Kennedy, J., & Eberhart, R. (1995, November/December). Particle swarm optimization. *Proceedings of the IEEE international conference on neural networks*. Perth, WA, Australia.

[3] Le Anh Duc, Kenli Li, Tien Trong Nguyen, Vu Minh Yen & Tung Khac Truong (2018) A new effective operator for the hybrid algorithm for solving global optimisation problems, International Journal of Systems Science, 49:5, 1088-1102

[4] Alatas, B. (2011). ACROA: Artifcial chemical reaction optimization algorithm for global optimization. *Expert Systems with Applications, 38*(10), 13170–13180.

[5] Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications, 115*(3), 549–570.

[6] Mirjalili, S. (2016). Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications, 27* (04), 1–21.

[7] Nguyen, T. T., Li, Z., Zhang, S., & Truong, T. K. (2014). A hybrid algorithm based on particle swarm and chemical reaction optimization. *Expert Systems with Applications, 41*(5), 2134–2143.

[8] Ao, H., Cheng, J., Zheng, J., & Truong, T. K. (2015). Roller bearing fault diagnosis method based on chemical reaction optimization and support vector machine. *Journal of Computing in Civil Engineering, 29*(5), 04014077.

[9] Bansal, J. C., Sharma, H., Jadon, S. S., & Clerc, M. (2014). Spider monkey optimization algorithm for numerical optimization. *Memetic Computing, 6*(1), 31–47. Dorigo, M. (1992). *Optimization learning and natural algorithms* (Ph. D. thesis). Italy: Politecnico di Milano University.

[10] Eberhart, R. C., & Shi, Y. (2000, July). *Comparing inertia weights and constriction factors in particle swarm optimization*. *Proceedings of the 2000 congress on evolutionary computation, IEEE, La Jolla, CA, USA*.

[11] García-Martínez, C., Rodriguez, F. J., & Lozano, M. (2012). Arbitrary function optimisation with metaheuristics. *Soft Computing, 16*(12), 2115–2133.

[12] Geem, Z. W., Kim, J.H., & Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation, 76*(2), 60–68.

[13] Golberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Boston, MA, USA: Addison Wesley Longman Publishing Co. Inc. Ho, Y.-C., & Pepyne, D. L. (2002).

[14] James, J., & Li, V. O. (2015). A social spider algorithm for global optimization. *Applied Soft Computing, 30*, 614–627.

[15] Kang, F., & Li, J. (2016). Artifcial bee colony algorithm optimized support vector regression for system reliability analysis of slopes. *Journal of Computing in Civil Engineering, 30*(3), 04015040.

[16] Kang, F., Li, J., & Ma, Z. (2011). Rosenbrock artifcial bee colony algorithm for accurate global optimization of numerical functions. *Information Sciences, 181*(16), 3508–3531.

[17] Kang, F., Xu, Q., & Li, J. (2016). Slope reliability analysis using surrogate models via new support vector machines with swarm intelligence. *Applied Mathematical Modelling, 40*(11), 6105–6120.

[18] Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization*, (Report No. TR06). Kaysery: Erciyes University.

[19] Kirkpatrick, S., & Vecchi, M. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

[20] Lam, A. Y., & Li, V. O. (2010a). Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation, 14*(3), 381–399.

[21] Lam, A. Y., & Li, V. O. (2010b, December). *Chemical reaction optimization for cognitive radio spectrum allocation*. *Proceedings of the 2010 IEEE global telecommunications conference (GLOBECOM 2010)*, Miami, FL, USA.

[22] Lam, A. Y., & Li, V. O. (2012). Chemical reaction optimization: A tutorial. *Memetic Computing, 4*(1), 3–17.

[23] Lam, A. Y., Xu, J., & Li, V. O. (2010, July). Chemical reaction optimization for population transition in peer-to-peer live streaming. *Proceedings of the 2010 IEEE congress on evolutionary computation (CEC)*, Barcelona, Spain.

[24] Li, Z., Wang, W., Yan, Y., & Li, Z. (2015). PS–ABC: A hybrid algorithm based on particle swarm and artifcial bee colony for high-dimensional optimization problems. *Expert Systems with Applications, 42*(22), 8881–8895.

[25] Pan, B., Lam, A. Y., & Li, V. O. (2011, December). *Network coding optimization based on chemical reaction optimization*. *Proceedings of the 2011 IEEE global telecommunications conference (GLOBECOM 2011)*, Kathmandu, Nepal. Storn, R., & Price, K. (1997). Diff erential evolution – a simple and efcient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.

[26] Truong, T. K., Li, K., & Xu, Y. (2013). Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem. *Applied Soft Computing, 13*(4), 1774–1780.

[27] Van Den Bergh, F., & Engelbrecht, A. P. (2006). A study of particle swarm optimization particle trajectories. *Information Sciences, 176*(8), 937–971.

[28] Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

[29] Xu, J., Lam, A. Y., & Li, V. O. (2010, May). *Chemical reaction optimization for the grid scheduling pr oblem*. *Proceedings of the 2010 IEEE international conference on communications (ICC), Cape Town, South Africa*.

[30] Xu, J., Lam, A. Y., & Li, V. O. (2011). Chemical reaction optimization for task scheduling in grid computing. *IEEE Transactions on Parallel and Distributed Systems, 22*(10), 1624–1631.

[31] Xu, Y., Li, K., He, L., Zhang, L., & Li, K. (2015). A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. *IEEE Transactions on Parallel and Distribution Systems, 26*(12), 3208–3222.

[32] Yu, J. J., Lam, A. Y., & Li, V. O. (2011, June). *Evolutionary artifcial neural network based on chemical reaction optimization*. *Proceedings of the 2011 IEEE congress on evolutionary computation (CEC)*, New Orleans, LA, USA.

[33] Li, Z., Nguyen, T. T., Chen, S., & Truong, T. K. (2015). A hybrid algorithm based on particle swarm and chemical reaction optimization for multi-object problems. *Applied Sofy Computing, 35*(5), 525–540.

[34] Amir Shabani, Behrouz Asgarian, Saeed Asil Gharebaghi, Miguel A. Salido, Adriana Giret, "A New Optimization Algorithm Based on Search and Rescue Operations", Mathematical Problems in Engineering, vol. 2019, Article ID 2482543, 23 pages, 2019.

[35] S. Arora, H. Singh, M. Sharma, S. Sharma and P. Anand, "A New Hybrid Algorithm Based on Grey Wolf Optimization and Crow Search Algorithm for Unconstrained Function Optimization and Feature Selection," in *IEEE Access*, vol. 7, pp. 26343-26361, 2019.

[36] Man-Wen Tian, Shu-Rong Yan, Shi-Zhuan Han, Sayyad Nojavan, Kittisak Jermsittiparsert, Navid Razmjooy,

New optimal design for a hybrid solar chimney, solid oxide electrolysis and fuel cell based on improved deer hunting optimization algorithm, Volume 249, 2020, 119414, ISSN 0959-6526, Journal of Cleaner Production

GIẢI THUẬT LAI GHÉP MỚI MPCM CHO LỚP BÀI TOÁN TỐI ƯU ĐƠN MỤC TIÊU

Abstract. Một trong những thách thức lớn nhất đối với các nhà nghiên cứu là tìm ra các giải pháp tối ưu hoặc các giải pháp gần như tối ưu cho các bài toán đơn mục tiêu.Trong bài báo này, các tác giả đã đề xuất một thuật toán mới gọi là MPCM cho các bài toán đơn mục tiêu. Giải thuật này là sự kết hợp của 4 phép toán: Mean-Search, PSOUpdate, CRO và một phép toán mới gọi là Min-Max. Các tác giả đã dùng các tham số để cân bằng giữa tìm kiếm cục bộ và tìm kiếm toàn cục để cho ra kết quả tối ưu hơn. Kết quả chứng minh rằng, với sự tham gia của thuật toán Min-Max. MPCM cho kết quả tốt trên 23 bài toán benchmark. Kết quả được so sánh với 3 giải thuật nổi tiếng, đó là Particle Swarm Optimization (PSO), Real Code Chemical Reaction Optimization (RCCRO) và Mean PSO-CRO (MPC)
Từ khóa: Tối ưu hóa toàn cục, các bài toán tối ưu đơn mục tiêu, giải thuật lai ghép

Received on: 02/03/2021

Accepted on: 28/04/2021