Iranian Journal of Operations Research
Vol. 11, No. 1, 2020, pp. 76-92
DOI: 10.29252/iors.11.1.76

Utilizing the Unified Ant Colony Algorithm by Chaotic Maps

Hamid Reza Yousefzadeh[1,*], Davood Darvishi[2], Arezoo Sayadi Salar[3]

Ant colony optimization (ACO_R) is a meta-heuristic algorithm for solving continuous optimization problems (MOPs). In the last decades, some improved versions of ACO_R have been proposed. The $UACO_R$ is a unified version of ACO_R that is designed for continuous domains. By adjusting some specified components of the $UACO_R$, some new versions of ACO_R can be deduced. By doing that, it becomes more practical for different types of MOPs. Based on the nature of meta-heuristic algorithms, the performance of meta-heuristic algorithms are depends on the exploitation and exploration, which are known as the two useful factors to generate solutions with different qualities. Since all the meta-heuristic algorithms with random parameters use the probability functions to generate the random numbers and as a result, there is no any control over the amount of diversity; hence in this paper, by using the best parameters of $UACO_R$ and making some other changes, we propose a new version of ACO_R to increase the efficiency of $UACO_R$. These changes include using chaotic sequences to generate various random sequences and also using a new local search to increase the quality of the solution. The proposed algorithm, the two standard versions of $UACO_R$ and the genetic algorithm are tested on the CEC05 benchmark functions, and then numerical results are reported. Furthermore, we apply these four algorithms to solve the utilization of complex multi-reservoir systems, the three-reservoir system of Karkheh dam, as a case study. The numerical results confirm the superiority of proposed algorithm over the three other algorithms.

***Keywords**: Ant colony algorithm; Continuous optimization; Chaotic sequences; Multi-reservoir systems; Genetic algorithm.*

Manuscript was received on 01/10/2020, revised on 02/29/2020 and accepted for publication on 04/09/2020.

1. Introduction

The first version of the ACO algorithm, namely the Ant System (AS), was attributed to Marco Dorigo in 1992, which is proposed in his Ph.D. thesis [21]. Ant-quantity, ant-cycle, and ant-density are the three main algorithms of the AS. The main contributions of these algorithms are based on when, how, and the density-value of pheromone that was deposited from the ants. For example, in the ant-cycle type, a pheromone is deposited when all ants had generated the path, and quality of the tour was considered as a function to update the pheromone' levels (refer to [21] to study more about how each of them works). Based on empirical results, the ant-cycle algorithm outperformed the other two algorithms and hence it was used to describe the AS. We can find numerous improvements and variants for the basic ACO since 1991 in the literature that are proposed, and studied by researchers. For more study, we can refer to the earlier ACO's developments briefly, such as the Elitist AS (see, e.g. [21], [23], [24] in 1991, 1992, and 1996 respectively), the Ant-Q

* Corresponding Author.
[1] Department of Mathematics, Payame Noor University (PNU) usefzadeh.math@pnu.ac.ir
[2] Department of Mathematics, Payame Noor University (PNU), d_darvishi@pnu.ac.ir.
[3] Department of Mathematics, Payame Noor University (PNU) sayadisalararezoo@yahoo.com

([18] in 1995), the Ant Colony System for TSP ([22] in 1996), the Max-Min Ant System ([42] and [43] in 1996 and 1997 respectively), the Rank-based AS ([4] in 1997), the EANTS ([46] in 1999), the Best-worst AS ([34] and [35] in 2000 and 2002 respectively), the Hyper Cube ACO ([8] in 2001), the Population based ACO ([25] in 2002), the Beam-ACO (see, e.g. [25] and [9] in 2002 and 2005 respectively).

The nature of the initial ACO algorithm was designed to combinatorial optimization problems, and later on, it was updated for the continuous ones. Discretizing the real-valued variables is the most straightforward approach to apply the ACO algorithm for continuous optimization problems (COPs). Calling the ACO by utilizing this methodology has been implemented to the Protein-ligand docking problem [3].

After presenting the ACO algorithm for discrete optimization problems, several ant-inspired algorithms for COPs have been proposed (See, e.g. references [5], [10], [13], and [27]). The main difference between these algorithms is based on their focus on the search mechanisms [13].

The first ACO algorithm considering the continuous domains is proposed by Socha and Dorigo 2008 in [6]. They suggested an extended version of the ACO algorithm called the ACO_R (where the index R indicates that the variables are real-valued) where they explicitly used the Gaussian kernel function instead of the probability density function.

Although numerical results show that the ACO_R may be successful in some problems but for the problems with high feasible-dimensions (i.e., in real-world applications) has not been investigated yet. More studies show that the ACO_R has a poor performance for large scale problems because it quickly loses its variety, and therefore negatively affects the quality of solutions.

Leguizamón and Coello in 2010 proposed an extension of ACO_R namely, $DACO_R$ (D stands for diversity), to increase the quality of the solutions and reduce the required computational time simultaneously. Based on their experimental results, the $DACO_R$ outperforms the ACO_R for unconstrained large scale COPs [15]. In this algorithm, despite more exploration of the response space, but they do not have any local exploration mechanism to control more accurate the solution space. Furthermore, automatic decision-making between local and global exploration based on the observed diversity is not considered.

In 2011, Liao et al. proposed an incremental variant of ACO_R entitled by $IACO_R$-LS. It uses a local search and a growing solution archive to diversify the search and expand the exploration [16]. To do that, they used different types of local search methods in their experiments, such as the Powell's conjugate directions set [30], Powell's BOBYQA [31], and Lin-Yu Tseng's Mtsls1 [44]. Their results showed that the $IACO_R$-LS conjunction with Mtsls1 (named by the $IACO_R$-Mtsls1) is not only outperforms the ACO_R, but also it is competitive with other state-of-the-art algorithms on the COPs. In later years, Guo et al. 2012 in [29] and Kumar et al. 2015 in [45] presented improved versions of the ACO_R algorithm. Yang et al. 2017 also introduced an extended variant of ACO_R for multi-modal optimization problems [33]. In 2018, Singsathid, and Wetweerapong presented a new continuous ACO, called PACO to make high precision solutions. They constructed and updated the pheromone matrix which is used to find a better solution to reduce and repartition the continuous variable domains iteratively. They proposed some suitable parameters for PACO and then compare it with those of other ACO in continues domains [37]. Duca, et al., 2019, studied the efficiency of ACO_R algorithm on the electromagnetic optimization problems. They used the ACO_R for solving two benchmark electromagnetic problems that are referred to the coils configuration's optimization. After choosing the appropriate population's size, the ACO_R algorithm performances are compared with results obtained with the GA and the Particle Swarm Optimization (PSO). The ACO_R outperforms the GA and the PSO for one problem, whereas for another one the PSO is the best algorithm [11]. Omran and Al-Sharhan, in 2019, proposed the $IACO_R$ (it uses a random-walk for selection operator) and $LIACO_R$, two versions of ACO_R, to improve the performance of ACO_R on real-world COPs. On the other hand, $LIACO_R$ uses a local search method to enhance the quality of solutions. In another word, they try to balance exploration and exploitation, simultaneously [19]. In

2020, Peng Wang et al. embedded the genetic algorithm (GA) and the cloud model into the ACO (named it CG-ACO) to avoid trapping in local optimums and increase the rate of ACO's convergence [32]. Their numerical results showed that the CG-ACO outperform the ACO_R, simple genetic algorithm (SGA), some other optimization algorithms which are embedded with cloud models such as CQPSO [32], and CAFSA [48] and the global optimal solution was more likely to achieve.

In 2014, Liao et al. proposed a unified structure of the ACO algorithm (which is known as $UACO_R$) for continuous optimization problems [17]. It provides a selection of particular components' algorithms to generate a various versions of ACO_R. The $UACO_R$ contains the algorithmic components from three ACO algorithms for continuous optimization problems i.e. the components of the ACO_R, the $DACO_R$ and the $IACO_R$-LS, that have been previously studied. Thus, from the $UACO_R$, one can be extracted from each of the three mentioned earlier algorithms; furthermore, from the proposed $UACO_R$, some new ACO algorithms for continuous space can be generated, which have not been investigated previous in the literature. In other words, from the $UACO_R$, some new continuous ACO algorithms can be derived automatically by enabling the use of the other algorithms configuration techniques. They generated two new ACO_R algorithms, entitled $UACO_R$-s and $UACO_R$-c, to investigate the flexibility of the $UACO_R$. Their numerical results showed the $UACO_R$ algorithms have high potentials for continuous optimization problems. Moreover, based on the experimental results, the $UACO_R$ outperforms all the previous versions that existed in the literature [20].

In this paper, by making some changes in the two main components of the $UACO_R$ algorithm, i.e., how to create diversity and increase the quality of solutions, we improve the performance of this algorithm in solving the COPs. These changes include the use of chaotic sequences to diversify the solutions and control the amount of variation created, and the use of a new local search method to increase the quality of the generated solutions.

The structure of the paper is organized as follows. Some related works due to the ACO_R algorithm are summarized in Section 2. We also describe the unified ACO_R algorithm i.e. $UACO_R$. The proposed method contains the chaotic sequence, and a new local search method is described in Section3. In Section 4, the benchmark test functions are presented. The parameters setting for two types of $UACO_R$, and the GA algorithms are determined in Section 5. Experimental settings and detailed results related to the comparison of the proposed algorithm regarding the three well-known algorithms are reported in Sections 6. The utilization of multi-reservoir systems of Karkheh Dam as a case study was investigated in Section 7. Finally, a summary and conclusions are provided in Section 8.

2. Unified Ant Colony Optimization Algorithm ($UACO_R$)

As mentioned before, the $UACO_R$ algorithm is a unified version of the ACO algorithm for the COPs [16]. This algorithm combines the various components of the ACO_R, the $DACO_R$, and the $IACO_R$-LS to achieve a tuned version of the ACO, i.e. $UACO_R$. It is called unified, because the originally mentioned algorithms, i.e. the ACOR, the DACOR, and the IACOR-LS can be generated by using parameter settings and particular combinations of the operational components. If we set the parameters and combinations of some components in the $UACO_R$ inappropriate manner, we can find some different variants of ACO from the $UACO_R$. It will be done by combining some related components from all the existing combinations.

Before describing the outline of $UACO_R$, which contains seven important groups of components, we need to abstract the main parts of the ACO_R, the $DACO_R$, and the $IACO_R$-LS.

2.1. Basic Ant Colony for COPs (ACO_R)

The outline of basic ACO algorithm is explained in Algorithm 1.

Algorithm1. Four main stages of ACO Algorithm

1- **Initialization:** set the all needed parameters
2- **While** the stopping criteria is not met, **DO**
   2.1 **Constructing solutions:** By considering pheromones values and the other related information, a set of ant solutions are built.
   2.2 **Local search procedure:** Improve the constructed ant solutions
2.3 **Updating pheromone:** Update the pheromone based on the search of ants' experience

In this algorithm, artificial ants follow a random approach to generate candidate solutions, using a pheromone model and existing heuristic information. The main parts of the ACO algorithm include generating the solution and updating the pheromone information (see e.g. [17] and [23]).

After introducing the original ACO for combinatorial problems (See Dorigo et al., 1991 [23] and Dorigo, Maniezzo, and Colorni, 1996 [24]), the whole of proposed Ant related algorithms for COPs use different kinds of search mechanisms regarding the original ACO [38] (See e.g. Bilchev and Parmee, 1995 [5]; Dréo and Siarry, 2004 [10]; Hu, Zhang, Chung, Li, and Liu, 2010 [14]).

The ACO_R that are proposed by Socha, and Dorigo, 2008 [38] is the first algorithm as an ACO algorithm that is designed for solving the COPs. The discrete probability distributions are replaced by the continuous probability density functions (PDFs) for constructing the solution archive in the ACO_R. Each PDF can be obtained during the search process. It builds a solution $x=(x_1, x_2, \ldots, x_n) \in R^n$ regarding the component x_j ($\forall j = 1,\ldots,n$) successively by applying the Gaussian kernel as follows:

$$G^j(x) = \sum_{i=1}^{k} \omega_i g_{ij}(x) = \sum_{i=1}^{k} \omega_i \frac{1}{\sigma_{ij}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ij})}{2(\sigma_{ij})^2}}$$

where k, ω_i, μ_{ij} and σ_{ij} is the size of solutions archive T, the weight, mean and standard deviation associated with the $g_{ij}(x)$ (i.e. one-dimensional Gaussian functions) respectively.

In order to calculate the G^j corresponding the component of x_j, we need to calculate the three vector parameters $\omega=(\omega_1, \omega_2,\ldots, \omega_k)$, $\mu_j=(\mu_{1j}, \mu_{2j},\ldots, \mu_{kj})$, and $\sigma j=(\sigma_{1j}, \sigma_{2j},\ldots, \sigma_{kj})$. To do that, set the mean vector μ_j as $\mu_j = (\mu_{1j}, \mu_{2j},\ldots, \mu_{kj}) := (x_{1j}, \ldots, x_{kj})$ and obtain the standard deviation vector $\sigma_j = (\sigma_{1j}, \sigma_{2j},\ldots, \sigma_{kj})$ as

$$\sigma_{ij} = \xi \sum_{e=1}^{k} \frac{|x_{ej} - x_{ij}|}{k-1} \qquad i = 1,\ldots,k$$

,

where the role of coefficient $\xi > 0$ is the same as a parameter of evaporation rate in the ACO algorithm. In another word, the lower the ξ's value, the higher the convergence rate of the algorithm. For updating the archive T, firstly, rank the newly generated solutions obtained during the search process, then choose the best solutions which maintain the cardinality k.

2.2. $DACO_R$ algorithm for COPs

The $DACO_R$ is an extended version of ACO_R, which concentrates on solutions' diversity and maintains the number of ants equal to the size of solution archive (i.e. k) where at each stage, a new solution is constructed by each ant. How the guide solution (Sol_{guide}) is chosen can say the other difference of the $DACO_R$ regarding the ACO_R. In other words, the best solution (Sol_{best}) in the solution archive is considered by the ant j as Sol_{guide} with probability p_{best} and, with probability ($1-p_{best}$), the solution S_j is considered as Sol_{guide} by the ant j. Generating the new solution (S_{new}) in the $DACO_R$ is the same as described in the ACO_R. Later on, S_{new} must be compared to the S_j (which of Sol_{best} or S_j was considered the Sol_{guide}). If the S_{new} is better than the S_j, it is replaced by S_{new} and put in the archive; otherwise, it is omitted. Note that in the ACO_R, all the solutions in the solutions archive are compared to all the newly generated solutions [15].

2.3. $IACO_R$-LS algorithm for COPs

An incremental solutions' archive *T* over iterations, and a local search method are the distinctive characteristics of the $IACO_R$-LS algorithm than the ACO_R. These two features can be enhanced the diversification and the search intensification, respectively. Furthermore, the $IACO_R$-LS algorithm chooses the Sol_{guide} in a different way than the ACO_R. For doing so, at each iteration, the $IACO_R$-LS algorithm chooses the Sol_{best} from *T* as the Sol_{guide} with a probability of $ElQ_{best} \in [0,1]$ and with the probability of 1- ElQ_{best}, the Sol_{guide} is selected from T to generate a new solution. With this selection rule, two cases at each iteration may be occurred: a new solution is generated by an ''elite'' Sol_{guide} or k different ants construct k new solutions. Note that each process of constructing a new solution similar to the way that the ACO_R uses. In the end, Sol_{guide} and the new solution are compared. If the Sol_{guide} is worse than the new solution, the new solution replaces it in the *T* set; otherwise, it is deleted from the further calculations. The size of archive T in the $IACO_R$-LS is initialized with pre-specified number solutions. At each iteration, a new solution is added to the T set until the cardinality of *T* is not greater than a maximum size. The $IACO_R$-LS calls a local search method at each iteration. If the local search method generates a better solution than the original one in the T, the older solution will be replaced by the better solution [16].

2.4. $UACO_R$ algorithm for COPs

Now we describe the $UACO_R$ algorithm that is suggested for solving the COPs. As mentioned before, the $UACO_R$ involves the main components from the ACO_R, the $DACO_R$, and the $IACO_R$-LS algorithms, been stated in previous sections. In addition to the three mentioned algorithms, some new ACO_R algorithms can be derived from the $UACO_R$, which has not been studied before in the literature.

The components of the $UACO_R$ algorithm can be stated as follow:

- *Mode*: There are two different modes for the $UACO_R$ called elite mode and default mode. The default mode uses several ants in each reproduction of the algorithm to construct the solutions. In the elite mode in each reproduction, an elite ant is used with the probability of $ElQ_{best} \in [0,1]$. The elite ant chooses Sol_{best} in the solution archive as Sol_{guide} to make a new solution.
- *Number of the ants*: There are two choices to determine the number of ants used in the $UACO_R$ algorithm. *Na* defines the number of ants as an independent parameter ($Na \leq k$) where the *k* is the size of the solution archive; while *NoIsAS* defines the number of ants equal to k (*NoIsAS* means that the number of ants is equal to the solution archive).
- *Choosing the Guide Solution*: This factor determines how to select Sol_{guide} for sampling from new solutions. To this end, we have three cases to choose from:
    1. $Sol_{guide} = Sol_{best}$ with probability $ElQ_{best} \in [0,1]$
    2. Choosing Sol_{guide} from the solution archive based on their weights
    3. Choosing the current solution Sol_L as Sol_{guide}
- *Updating the solution archive T*: updating the solution archive deals with adding the new solutions in the archive *T*. There are two following cases for doing that:
    1. The local worst parameter determines that $UACO_R$ will generally remove *Na* worst solutions from total $k + Na$ solutions or decide on the acceptance of Sol_L locally.
    2. The Sol_{new}-G_{sol} parameter specifies that the generated new solution by the ant *L* is compared with Sol_{guide} or with Sol_L, and then remove the worst solution.
- *Local search*: In general, we can use the *LS* method in different ways. If the local parameter type equals *F* (or False), no *LS* method is used. Otherwise, the local type selects one of three local searches the Mtsls1 and conjugate directions of Powell in $IACO_R$-LS (See [16] for more study) or the evolutionary CMA-ES of Molina et al. in [26].

- *Incrementing in archive*: Here, we can increase the size of the solution archive. If parameter *Inc=F* (or equal to False), then the mechanism of the incremental archive is not applied. Otherwise, the $UACO_R$ applies an incremental archive mechanism.
- *Restarting technique*: Three options are set for this technique. If parameter *ResType* = F, the restarting technique is not used. Otherwise, *ResType* uses one of the two-restart techniques introduced in $IACO_R$-LS.

3. Proposed Algorithm

In this section, some changes to the $UACO_R$ algorithm are studied. These changes include two main phases, which are described below.

Phase 1: Using Chaotic Sequences

The issue of optimization algorithms based on the chaotic sequence has been studied studied by many researchers. The nature of chaotic dynamic algorithms is suitable for solving optimization problems. Since the chaotic variables can search the whole solution space non-repeatedly, then optimization algorithms based on the chaotic sequence can be capable of hill-climbing to avoid trapping into local optima. Numerical results show that the chaotic search is more effective than the random search [50]. The chaotic ACO_R algorithm based on the chaotic sequence can be considered as a chaotic optimization algorithm and successfully applied to the process of ACO_R. Hence, we apply the chaotic ACO_R algorithm to overcome some drawbacks of the ACO_R by increasing the variety. The rate of diversity in solutions increases, if the algorithms use the chaos maps (i.e. evolution function) through their process. Nearly in all random meta-heuristic algorithms (i.e. algorithms with random components), the random numbers are generated by using probability functions, mainly the Gaussian functions. Instead of using probability functions, chaotic maps can be applied as alternative ones. To this end, we investigate the behavior of combinations of meta-heuristic $UACO_R$ algorithms with the proper chaotic map.

In the first phase, the chaotic maps are used to generate sequences of numbers randomly. The chaotic sequences are used to initialize the solution archive. This will allow the solutions to be distributed over a fuller domain of search space, and thus, the chance of finding the global optimal solution is increased. To this end, according to our numerical investigations (see section 5.2) the following logical map is applied in our proposed algorithm as a chaotic map:

$$x_{n+1} = \exp(-4.90x_n^2) - 0.58, \tag{1}$$

The pseudo-code shown in Algorithm 1 illustrates the initialization of the solutions in the solution archive using chaotic sequences. In this pseudo code, *k* is the size of the solution archive, s_i^0 is the solution *i* in the solution archive, the *LB*, and the *UB* are the vectors with the same size of the corresponding solution, and each their components is the lower and the upper threshold for each component of the decision variable. *CS* is the vector with the same size as the solution created by Generate-Chaotic-Sequence that contains a random sequence generated by one of the chaotic maps. This is done by calling *Ch_Map_Name*, which represents the map type that is used. Note that *D* is the solution's dimension.

Algorithm 1. Pseudo code of Initialization phase with chaotic map

For *i* =1: *k*
CS= *Generate_Chaotic_Sequence* (*D*, *Ch_Map_Name*);
s_i^0 =*LB*+*CS (UB-LB)*;
End.

Phase 2: Local Search (LS) Method

In the LS phase, the neighborhood of each solution is searched for obtaining a better solution regarding the current solution. For this phase, the proposed algorithm uses the pseudo-code that is given in Algorithm 2. In this pseudo code, S_{new} is a new solution generated by the LS method that starts from the current solution S_i. Index m is due to the solution that is chosen randomly from the solution archive, and j is the component number that is randomly selected among the solution components. The j^{th} component of the i^{th} solution can be changed in step 5 of Algorithm 2. In this pseudo-code, the rand-select function generates a random integer number in the given domain D.

Algorithm 2. Pseudo code of the LS phase

For i= 1: k
m= *rand_Select*(1: k) and $m \neq i$;
$S_{new} = S_i$;
j= *rand_Select*(1: D)
$S_{new\,j} = S_{i\,j} + rand\ (0, 1).\ (S_{m\,j} - S_{i\,j})$;
fit_{new}= *Evaluate*(S_{new});
If ($fit_{new} < fit_i$)
$fit_i = fit_{new}$;
$S_i = S_{new}$;
End if
End.

Evaluation of the fitness function and replacement of the generated new solution by the LS phase is performed simultaneously. This can help to increase the rate of convergence of the algorithm to the optimal solution. The purpose of the LS that is described in Algorithm 2 is to make a small change in the current solutions. These small changes can yield some improvement in the quality of a solution. In the $UACO_R$ algorithm, the generation of a solution is done by many steps.

It is worth noting that the initialization step's chaotic map led to increasing the randomness and hence, variety in generated solutions. On the other hand, the existence of proper LS shrinks the length of steps towards the global optimal solution and then increases the solutions' accuracy. In other words, this phase can create a balance between centralization and diversification issues.

4. Benchmark Problems

Suganthan, et al., in 2005 designed and proposed some real-world problems as benchmark problems (which are known to the "CEC05"), including properties and mathematical formulas, evaluation criteria, and codes which are executables. The CEC05 set contains 25 benchmark functions that are carried on some optimization algorithms. The corresponding codes can be found in *Matlab*, *Java*, and *C* (for more details, see [12]).

Summary of the twenty-five CEC05 functions can be categorized as follows:

a) **Unimodal Functions**

There are five unimodal functions which are named by **(F_1 –F_5)**: F_1 (Shifted Sphere), F_2 (Shifted Schwefel's), F_3 (Shifted Rotated High Conditioned Elliptic Function), F_4 (Shifted Schwefel's with Noise in Fitness), F_5 (Schwefel's with Global Optimum on Bounds)

b) **Multi-modal Functions**

The number of multi-modal functions is twenty and classified as:

b1) **seven basic functions** (F_6 -F_{12}): F_6: Shifted Rosenbrock's Function, F_7: Shifted Rotated Griewank's Function without Bounds, F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds, F_9: Shifted Rastrigin's Function, F_{10}: Shifted Rotated Rastrigin's Function, F_{11}: Shifted Rotated Weierstrass Function, F_{12}: Schwefel's Problem

b2) **two expanded functions** (F_{13} -F_{14}): F_{13}: Rosenbrock's Function (F8F2) plus Expanded Extended Griewank's, F_{14}: Shifted Rotated Expanded Scaffer's F6,

b3) **eleven hybrid composition functions** (F_{15} –F_{25}): F_{15}: Hybrid Composition Function, F_{16}: Rotated Hybrid Composition Function, F_{17}: Rotated Hybrid Composition Function with Noise in Fitness, F_{18}: Rotated Hybrid Composition Function, F_{19}: Rotated Hybrid Composition Function with a Narrow Basin for the Global Optimum, F_{20}: Rotated Hybrid Composition Function with the Global Optimum on the Bounds, F_{21}: Rotated Hybrid Composition Function, F_{22}: Rotated Hybrid Composition Function with High Condition Number Matrix, F_{23}: Non-Continuous Rotated Hybrid Composition Function, F_{24}: Rotated Hybrid Composition Function, F_{25}: Rotated Hybrid Composition Function without Bounds.

In this study, to evaluate the proposed algorithm, the standard benchmark functions, i.e., CEC05 have been used as test functions. The reported results are made of applying the proposed algorithm, the UACOR-c, the UACOR-s, and the GA for 25 evaluation functions with 30-dimensions and are compared to each other.

5. Parameters' setting

5.1. Settings for the $UACO_R$ and GA

In this study, the parameters are set in two ways and called by the UACOR-s (*pset1*) and the UACOR-c (*pset2*) and with the same way which is done by Liao, et al., 2014 [17]. For example, in the first kind of parameter setting or UACOR-c the setting parameters are as follows (refer to [17] for more details):

1- Default-Mode is set to True, i.e. the default mode is followed. If Default-Mode=False, the elitism mode is selected.

2- The parameter *NoIsAS* is equal to True; that is, the number of ants is equal to the size of archive *T*. If the *NoIsAS* =False, the case of $Na \leq k$ is activated (See the UACOR-s).

3- The parameter of Weight-G_{sol} is set to True, i.e. the Sol_{guide} is selected from the archive T regarding the weights are defined in the ACO_R. When the Weight-G_{sol} = False, the current solution Sol_L is considered as the Sol_{guide}.

4- The parameter of Local-Worst is considered equal to False. The Local-Worst in the UACOR-s algorithm is set to Ture. In this case, the parameter Sol_{new}-G_{sol} can be considered as True or False. In the case Sol_{new}-G_{sol}=True, each newly generated solution is compared with the corresponding Sol_{guide} and the worst one is deleted. Otherwise, the new solution compares with the Sol_L and the worse solution is removed.

5- The Mtsls1 is used as a local search method i.e. Local-Type= Mtsls1 (Note that the CMA-ES is called in UACOR-s as a local search).

Both of the UACOR-c, UACOR-s algorithms use the incremental solution archive mechanism (Inc=True)

The corresponding parameters settings of the UACOR-c, UACOR-s algorithms, and the GA are summarized in Table 1.

Table 1. Parameters settings of the $UACO_R$-c, the $UACO_R$-s and the GA

Algorithms	Parameters
UACOR-s (pset1)	**Default-Mode=True**
	NoIAs=False
	Weight-Gsol=False
	Local-Worst=False
	Sol_{new}-G_{sol}=True
	Local-Type=CMA-ES
	Inc=True
UACOR-c (pset2)	**Default-Mode=True**
	NoIsAS=True
	Weight-G_{sol}=True
	Local-Worst=False
	Local Type=Mtsls1
	Inc=True
GA	**Population size=20**
	Mutation rate=0.3
	Crossover= one point crossover
	Selection= Elitism

The results obtained by applying the $UACO_R$-s, $UACO_R$-c algorithms regarding each parameter set (See Table 1) namely *pset1* and *pset2*, respectively, on test functions F_1-F_{25}, are shown in Figure 1.

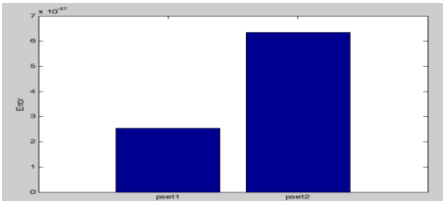

Figure 1. The average relative errors for the pset1 and pset2 on the test functions

The first parameters set (i.e. *pset1*) have less average relative error than the second type (i.e. *pset2*). Therefore, generally speaking, the quality of the obtained solutions for *pset1* is higher than the second type.

5.2. Choosing the Chaotic Map

In this section, we study the behavior of some well-known chaotic map such as Tent map (a real-valued function), The logistic map (a polynomial map of degree 2), the Gauss map (a nonlinear iterated map which is known as mouse map or Gaussian map), circle map and Sinai map (See e.g. [49], [50] and [39]).

To choose the proper chaotic maps, we examine the effect of the mentioned chaotic maps on the ACO_R algorithm. The results of the ACO_R algorithm with the various chaotic maps on test functions are shown in Figure 2 as the bar chart.

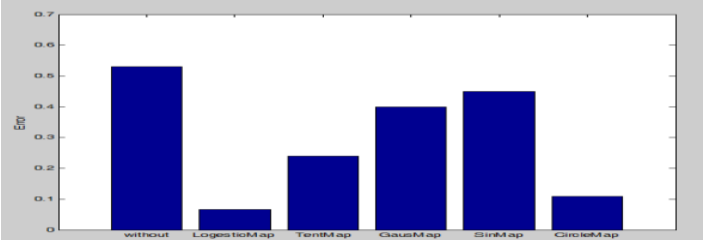

Figure 2. The average relative errors for applying chaotic maps on test functions

We observe that irrespective of what chaotic maps are used, the performance of ACO_R without using any chaotic map leads to high computational error than the ACO_R, which is utilized by a chaotic map (See Figure 2). Furthermore, according to the numerical results, the logical map has less average relative error compared to the other maps, and hence, we can generate better solutions in quality. Hence, we applied the logical maps in our proposed algorithm.

5.3 Influence of Local Search

In this section, the corresponding results regarding the utilization of the proposed algorithm with the three local searches such as the Mstls1, the CMA-EA, and the newly *LS* are examined. We tested the proposed algorithm on the benchmark CEC05, which is utilized by the three mentioned local searches. The numerical results show that the new *LS* has a better performance than the Mstls1 and the CMA-EA (see Figure 3). The average relative errors in the new LS are less than the Mstls1 and the CMA-EA.

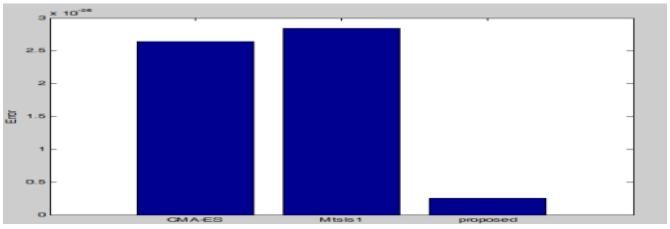

Figure 3. The effect of local searches on the test functions CEC05

The results indicate that the performance of the proposed *LS* is better than the two other methods, and therefore, it is implemented for further investigation in our algorithm. It is worth noting that, at the end of the total runs of each algorithm, the solution with the best fitness is considered as the approximation of the global optimal solution for each test function, and the number of iterations for each algorithm is limited by 30,000 iterations. Moreover, we use the average relative errors and Friedman's test to compare the obtained results.

6. Experimental Results

Whenever each algorithm satisfies the stopping criteria, then the relative deviation of the best solution from the optimal solution is considered as the relative error (or "error" for convenience) regarding the function that is used. For the given conditions, each algorithm runs 50 times on each function, and then the average error is reported. The average errors corresponding to the UACOR-s, the UACOR-c, the GA, and the proposed algorithm are shown in Table 2.

Table 2. Average errors of the proposed algorithm, UACOR-s, UACOR-c, and GA

Test Functions	GA	UACOR-c	UACOR-s	Proposed $UACO_R$
F_1	2.5329E+03	1.9245E-24	3.3265E-24	**2.5435E-27**
F_2	2.3422E+04	**4.3548E-32**	2.3448E-30	5.5430E-29
F_3	1.2543E+08	**1.2536E+05**	1.2846E+05	1.4521E+05
F_4	1.7358E+04	3.2234E-05	**2.2478E-05**	3.6456E-04
F_5	1.4921E+04	2.4836E+02	2.5428E+02	**2.8955E-03**
F_6	2.6687E+08	1.7326E+00	1.4916E+00	**1.4159E+00**
F_7	4.7158E+03	8.4551E-03	**7.5651E-03**	2.8503E-02
F_8	3.2648E+01	1.9948E+00	2.1048E+00	**6.5938E-02**
F_9	1.4436E+02	2.5987E-08	2.5847E-08	**2.2198E-09**
F_{10}	1.4358E+02	5.8524E+01	**5.3497E+01**	5.6314E+01
F_{11}	2.4335E+01	6.2559E-01	**9.2479E-03**	8.8183E-01
F_{12}	1.7519E+05	2.3491E+02	2.9472E+02	**3.2598E+00**
F_{13}	2.2531E+01	2.7812E+00	2.5361E+00	**3.4139E-01**
F_{14}	4.4546E+01	5.6247E+00	5.7048E+00	**5.5311E+00**
F_{15}	8.9478E+02	1.8754E+01	1.3416E+01	**6.9146E+00**
F_{16}	4.3419E+02	1.8556E+02	1.3695E+02	**2.3793E+01**
F_{17}	1.2536E+03	1.2379E+02	3.9238E+01	**3.9147E+01**
F_{18}	1.3478E+03	4.8635E+02	7.8412E+02	**4.6462E+02**
F_{19}	1.2452E+03	4.6462E+02	7.3475E+02	**5.5943E+01**
F_{20}	1.2463E+03	8.6723E+02	8.3165E+02	**8.4462E+01**
F_{21}	1.2541E+03	7.3290E+02	7.2860E+02	**5.6549E+01**
F_{22}	1.5403E+03	8.4463E+02	8.4423E+02	**1.0017E+01**
F_{23}	1.2491E+03	**1.8469E+02**	5.1986E+02	1.8635E+02
F_{24}	2.1300E+03	2.1500E+02	2.0000E+02	**3.1988E+01**
F_{25}	5.6422E+02	2.1023E+02	2.0000E+02	**3.3372E+01**

The bold values are less than all the values in that row, which indicates that, the corresponding algorithm outperforms the three other algorithms for the related test function CEC05. The values shown in grey indicate that based on the Friedman's test, the results obtained from the corresponding algorithm have a significant difference regarding the three other algorithms for the related CEC05's function. We observed that the proposed algorithm outperforms the UACOR-s, the UACOR-c, and the GA algorithms. Note that, although the UACOR-s or the UACOR-c performs better than the other mentioned algorithms in some test functions (see e.g. F_3 and F_{10}), according to Friedman's test, there is no significant difference between the performance of the four algorithms.

7. Utilization of Multi-Reservoir Systems (Case study: Karkheh Dam)

In this section, the proposed algorithm is used to optimize the utilization of a group of water resources, which include a three-reservoir system (Karkheh, Sazbon, and Mashoureh) and also four agricultural regions (including three regions 1, 2 and 3, and regions (4, 5, and 6) as a unit region) to evaluate the performance of the models (See Figure 4 that is adapted from [7]).

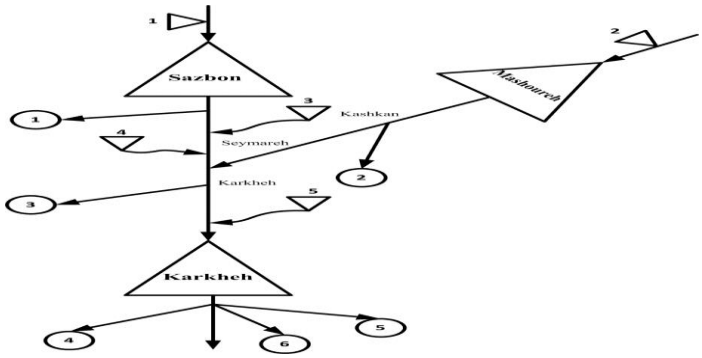

Figure 4. The three-reservoir system of Karkheh and four agricultural areas [7]

In Figure 4, the notations are described as below:

O :

1. Agricultural demand for Sazbon
2. Agricultural demand for Kashkan
3. Agricultural demand for Baalam
4. Agricultural demand for Dasht-e-Abbas
5. Agricultural demand for Dosalq & Arayez and Bagheh
6. Agricultural demand for Karkhe

∇ :

1. Inlet to Sazbon dam
2. Inlet to Mashoureh dam
3. Inlet to downstream of Sazbon dam
4. Inlet to downstream of Kashkan dam
5. Inlet to upstream of Karkheh dam

The Karkheh dam is a large multi-purpose earthen embankment dam built in Iran on the Karkheh River in 2001 by the Islamic Revolutionary Guards Corps. This dam is located 21 km away from the Northwest of Andimeshk and was constructed on the Karkheh River in Khuzestan province of Iran. The dam is about 127 meters high and 3030 meters long. The type of dam is a clay core with a total volume of 7300 million cubic meters, and its dewatering started in February 1999. One of the main applications of this reservoir is to control, and regulate the surface water flow of the Karkheh river (in order to provide the land with water in the nearby plains including Evan, Dosalq, Arayez, and Bagheh as well as the Hamidieh, Qods, Azadegan plain, Dasht-e-Abbas, Fakeh and Ainkhosh). The other applications of this dam include hydropower production, controlling season floods and avoiding damages to the downstream area. Sazbon reservoir is located 30 km east of the Ilam province, and was constructed on the Seymareh River. The Mashoureh reservoir is located 90 km away from Khorramabad, and was constructed on the Kashkan River in Chaharmahal-e Bakhtiari province, Iran.

As mentioned above, in this system, optimal utilization of the reservoirs of Sazbon, Mashoureh and Karkheh are needed to meet the needs of the four mentioned agricultural regions (i.e. regions 1, 2 and 3 and regions (4, 5 and 6)). Also, at the downstream of each reservoir and split into agricultural region 3, the provision of the minimum environmental flow requirements in the river is mandatory. The priority is to meet the environmental flow needs in the river and agricultural regions, respectively. Table 3 shows the average monthly inflows to the system in a year.

Table 3. Monthly average inflows for the three-reservoir system of Karkheh

Reservoir	Sep.	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.	May.	June.	July.	Aug.
Sazbon	57	139	221	233	322	529	768	507	162	69	49	42
Mashoureh	14	21	68	88	72	67	93	66	23	13	13	13
Entering Branch 1	25	25	34	44	31	25	24	31	39	23	19	20
Entering Branch 2	34	52	84	57	141	216	329	233	85	55	40	31
Entering Branch 3	18	39	84	84	80	112	89	130	151	62	37	29

The agricultural needs for the four mentioned regions and the monthly distribution of net evaporation from the reservoir's surfaces are presented in Tables 3 and 4, respectively.

Table 4. Monthly absolute evaporation from the surface of the three-reservoir system of Karkheh

Reservoirs	Sep.	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.	May.	June.	July.	Aug.
Sazbon	0.167	0.074	0.0	0.0	0.0	0.0	0.056	0.114	0.213	0.249	0.243	0.220
Mashoureh	0.149	0.068	0.015	0.008	0.181	0.017	0.067	0.121	0.181	0.216	0.220	0.191
Karkheh	0.208	0.136	0.081	0.061	0.070	0.099	0.148	0.211	0.266	0.280	0.274	0.244

Table 5. Monthly agricultural needs of four agricultural regions

Regions	Sep.	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.	May.	June.	July.	Aug.
1	48.5	48.5	60.6	60.6	60.6	60.6	60.6	0.0	0.0	0.0	0.0	0.0
2	20.4	1.2	0.0	0.0	0.0	0.0	0.8	55.8	69.1	70	51.8	31.7
3	37.2	37.2	46.5	46.5	46.5	46.5	46.5	0.0	0.0	0.0	0.0	0.0
(4, 5, 6)	326.7	219.9	148.7	157.9	156.2	276.0	413.8	411.6	231.9	406.5	403.7	445.8

The environmental requirements for the minimum flow in four intervals of the river, including the downstream of Sazbon and Mashoureh reservoirs and the upstream and downstream of the Karkheh reservoir, were fixed at 75, 1.43, 75 and 75 million cubic meters per month, respectively.

The nonlinear programming problem (2)-(6) is considered regarding the three-reservoir system of Karkheh. This model is formulated for 47 years (564 months) from the solar year 1954 until 2001. The optimal solution (or optimal utilization path) of this model contains the optimal harvesting of each reservoir and the optimal supply for each of the agricultural needs during the months. Regardless of the constraints due to the minimum environmental requirement in the monthly intervals, the numbers of decision variables are 3948 variables, which will face a severe challenge by any well-known algorithm to solve it. The objective function (Z) for this problem is minimized by the quadratic cost function for the cases of deficiency (one-way cost function) and is presented concerning the other corresponding constraints that are given in constraints (3) - (6).

$$Minimize\ Z = \begin{cases} \sum_{t=1}^{564}\sum_{j=1}^{4}(D_t^j - Rg_t^j)^2 & Rg_t^j \le D_t^j \\ 0 & o.w. \end{cases} \quad (2)$$

Subject to:

$$S_{t+1}^{NR} = S_t^{NR} + Q_t^{NR} - R_t^{NR} - E_t^{NR}, \quad NR = 1,2,3 \quad (3)$$

$$R_t^{NR} \ge R_{\min}^{NR}, \quad NR = 1,2,3 \quad (4)$$

$$S_{\min}^{NR} \le S_t^{NR} \le S_{\max}^{NR}, \quad NR = 1,2,3 \quad (5)$$

$$S_1^{NR} \le S_{t+1}^{NR} \le S_{\max}^{NR}, \quad NR = 1,2,3 \quad (6)$$

In this regard, the notations S_t, Q_t, R_t, E_t, and D_t are respectively, the storage volume at the beginning of period t, the amount of input in period t, the released rate for period t, the evaporation volume from the reservoirs surfaces in period t and the monthly demands for agricultural regions in period t. The notation NR is the reservoir number. Rg^j is the amount of water allocated to the j^{th} agricultural region. Storage at the beginning of the first period and the end of the last period for all the reservoirs was unknown but assumed equal. This condition is specified in Eq. (6).

For solving this problem, the amount of storage volume for the first period for all the reservoirs is initialized randomly in the feasible space. It is worth noting that almost all methods use penalty expression for repairing the infeasible solutions and getting the feasible ones. The decision variables include the amount of storage in each reservoir (S^{NR}) and the assignment to each region (Rg^j), and, as indicated, the numbers of variables are 3948. Increasing the number of decision variables makes the problem difficult to solve optimally. Therefore, we consider utilizing multi-reservoir systems as an optimization problem, as illustrated by (2) - (6).

Here, we will solve the optimization problem multi-reservoir systems with the four algorithms mentioned in Section 6 and evaluate the obtained results to we examine the performance of the proposed algorithm on a real instance.

Note that the number of reservoirs in this dam is 3 and the number of agricultural regions is 4. We apply each of three algorithms with 50 independent runs by considering the optimization problem (2)-(6) related to the multi-reservoir optimization systems, and the average relative errors from the best solution of the proposed algorithm, the UACOR-s, the UACOR-c, and the GA are shown in Table 6.

Table 6. The average errors of the proposed algorithm, UACOR-s, UACOR–c, and GA regarding the multi-reservoir system of Karkheh

GA	UACOR-s	UACOR-c	Proposed $UACO_R$
5.7381E+02	5.9311E-05	9.5311E-05	**3.5311E-06**

According to Table 6, the better performance of the proposed $UACO_R$ algorithm than the three other algorithms shows that the proposed components used in this algorithm, such as a chaotic map and the local search approach, can be affected on quality of solutions. Making more solutions, variety and more accurate exploring the search space may be the main reason for the proposed algorithm's better performance compared to its original version.

8. Summary and Conclusion

Ant colony optimization algorithm for continuous domains (ACO_R) is the well-known meta-heuristic algorithms for solving continuous optimization problems (COPs) that have been considered by many researchers in recent decades. The unified ant colony algorithm ($UACO_R$) is a kind of ACO_R that provides a unified framework for making some new versions of the ACO algorithm. In the present study, some changes have been made to the $UACO_R$. These changes include the use of chaotic maps in the initialization phase, which increases the diversity in the initial population. Furthermore, a local search method based on differential evolution has also been applied. The proposed *LS* improved the algorithm's performance, because it may lead to an increase in the solutions' variety by defining a different way to search the solutions' neighborhoods. The proposed $UACO_R$ and two well-known versions of $UACO_R$ (namely $UACO_R$-s and $UACO_R$-c) and the GA are tested on the benchmark optimization functions "CEC05" that contains 25 unimodal and multi-unimodal functions. Moreover, the proposed algorithm has been applied to optimize the utilization of multi-reservoir systems in the Karkheh dam as a case study. The corresponding results confirm that the proposed algorithm outperforms the three other algorithms i.e. $UACO_R$-s and $UACO_R$-c and GA algorithms. The numerical results indicate that making some changes (e.g., using chaotic sequences to generate initial solutions) on the exploitation and exploration can improve the performance of heuristic algorithms without any expensive cost.

References

[1] Agarwal, R., Tiwari, M.K., and Mukherjee, S.K. (2007). Artificial immune system based approach for solving resource constraint project scheduling problem, International Journal of Advanced Manufacturing Technology, 34, 584-593.
[2] Akbari, R., Zeighami, V., and Ziarati, K. (2011). Artificial Bee colony for resource constrained project scheduling problem, International Journal of Industrial Engineering Computations, 2, 45-60.
[3] Aleem, A. (2019). Evolution of Ant colony optimization algorithm: a brief literature, Computing Research Repository, abs/1908.08007, 1-11.
[4] Bernd, B., Richard, F., and Christine, S. (1997). A new rank based version of the ant system. A Computational Study.
[5] Bilchev, G., and Parmee, I. (1995). The ant colony metaphor for searching continuous design spaces. In T. Fogarty (Ed.), AISB workshop on evolutionary computing, 25-39. Springer-Verlag.
[6] Blum, C., and Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Computing Surveys (CSUR), 35(3), 268-308.
[7] Borhani darian, A., and Moradi, A. (2011). Application of ant colony based algorithms to multi reservoir water resources problems, Journal of Water and Wastewater, 21(4), 81-91 (Persian).
[8] Christian, B., Andrea, R., and Marco, D. (2001). Hc–ACO: The hyper-cube framework for ant colony optimization. In Proceedings of MIC, 2, 399-403.
[9] Christian, B. (2005). Beam-ACO hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers and Operations Research, 32(6), 1565–1591.
[10] Dréo, J., and Siarry, P. (2004). Continuous interacting ant colony algorithm based on dense heterarchy. Future Generation Computer Systems, 20(5), 841–856.
[11] Duca, A., Ciuprina, G., Lup, S., and Hameed, I., (2019). ACO_R algorithm's efficiency for electromagnetic optimization benchmark problems, 11th International Symposium on Advanced Topics in Electrical Engineering (ATEE), Bucharest, Romania, 1-5.
[12] http://www.ntu.edu.sg/home/EPNSugan
[13] Hu, X. M., Zhang, J., and Li, Y. (2008). Orthogonal methods based ant colony search for solving continuous optimization problems. Journal of Computer Science and Technology, 23, 2–18.

[14] Hu, X. M., Zhang, J., Chung, H.S., Li, Y., and Liu, O. (2010). Sam ACO: variable sampling ant colony optimization algorithm for continuous optimization. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 40, 1555–1566
[15] Leguizamón, G., and Coello, C.A., (2010). An alternative ACOR algorithm for continuous optimization problems, In ANTS Conference, 48-59.
[16] Liao, T., Montes, M.A, Aydin, D., Stützle, T., and Dorigo, M., (2011). An incremental ant colony algorithm with local search for continuous optimization, In Proceedings of the 13th annual conference on genetic and evolutionary computation, 125-132.
[17] Liao, T., Stützle, T., and Dorigo, M., (2014). A unified ant colony optimization algorithm for continuous optimization, European Journal of Operational Research, 234(3), 597-609.
[18] Luca, M.G. and Marco, D. (1995). Ant-Q: a reinforcement learning approach to the traveling salesman problem. In Machine Learning Proceedings, 252–260.
[19] Mahamed G.H. and Omran, S. (2019). Improved continuous Ant colony optimization algorithms for real-world engineering optimization problems. Engineering Applications of Artificial Intelligence, 85, 818-829.
[20] Manuel, L., Jeremie, D.L., Leslie, P.C., Mauro, B., and Thomas, S. (2016). The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3, 43–58.
[21] Marco, D. (1992). Optimization, learning and natural algorithms. PhD Thesis, Politecnico diMilano.
[22] Marco, D. and Luca, M.G. (1997). Ant colonies for the traveling salesman problem. Biosystems.
[23] Marco, D., Vittorio, M., and Alberto, C. (1991). The ant system: an autocatalytic optimizing process.
[24] Marco, D., Vittorio, M., and Alberto, C. (1996). Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26(1), 29–41, 1996.
[25] Michael, G. and Martin, M. (2002). A population based approach for ACO. In Workshops on Applications of Evolutionary Computation, 72–81.
[26] Molina, D., Lozano, M., Snchez, A., and Herrera, F. (2011). Meme tic algorithms based on local search chains for large scale continuous optimization problems: MASSW-Chains, Soft Computing-A Fusion of Foundations, Methodologies and Applications, 15, 2201–2220.
[27] Monmarché, N., Venturini, G., and Slimane, M. (2000). On how Pachycondyla apicalis ants suggest a new search algorithm. Future Generation Computer Systems, 16(9), 937–946.
[28] Peng, W., Jiyun, B., and Jun, M. (2020). A hybrid genetic ant colony optimization algorithm with an embedded cloud model for continuous optimization. Journal of Information Processing Systems, 16(5), 1169–1182. doi.org/10.3745/JIPS.01.0059.
[29] Ping, G. and Lin, Z. (2012). Ant colony optimization for continuous domains. In Natural computation (ICNC), eighth international conference, 758-762. IEEE.
[30] Powell, M. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives, The Computer Journal,7(2), 155.
[31] Powell, M. (2009). The BOBYQA algorithm for bound constrained optimization without derivatives, Cambridge NA Report NA2009/06, University of Cambridge, UK.
[32] Qi, M. and Yang, A. (2012). Quantum particle swarm optimization based on cloud model cloud droplet strategy. Computer Engineering and Applications, 48(24), 49-52.
[33] Yang, Q., Chen, W.N., Yu, Z., Gu, T. Li, Y., Zhang, H. and Zhang, H. (2017). Adaptive multimodal continuous ant colony optimization. IEEE Transactions on Evolutionary Computation, 21(2), 191–205.
[34] Oscar, C., Inaki, F.V., Francisco, H., and Llanos, M. (2000). A new ACO model integrating evolutionary computation concepts: The best-worst ant system.
[35] Oscar, C., Inaki, F.V., and Francisco, H. (2002). Analysis of the best-worst ant system and its variants on the tsp. Mathware and soft computing, 9 (2).
[36] Rueymaw, C. (2011). Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem, Expert Systems with Applications, 38, 7102-7111.
[37] Singsathid, P., and Wetweerapong, J. (2018). Solving continuous optimization problems by ant colony optimization with domain partitioning technique, ASEAN Foreign Ministers' *Meeting* (*AMM*), Bangkok, Thailand.
[38] Socha, K., and Dorigo, M. (2008). Ant colony optimization for continuous domains, European Journal of Operational Research, 185(3), 1155-1173.
[39] Sole, R.V., and Miramontes, O., and Goodwin, B.C. (1993). Oscillations and chaos in ant societies, Journal of Theoretical Biology, 161, 343-357.

[40] Stützle, T., and Dorigo, M. (1999). ACO algorithms for the quadratic assignment problem, New Ideas in Optimization, 33.
[41] Tianjun, L., Thomas, S., Marco, A.M., and Marco, D. (2014). A unified ant colony optimization algorithm for continuous optimization. European Journal of Operational Research, 234(3), 597–609.
[42] Thomas, S. and Holger, H. (1996). Improving the ant system: a detailed report on the max-min ant system. FG Intellektik, FB Informatik, TU Darmstadt, Germany, Tech. Rep. AIDA-96-12.
[43] Thomas. S. and Holger, H. (1997). Max-min ant system and local search for the traveling salesman problem. In Evolutionary Computation, IEEE International Conference on, 309-314.
[44] Tseng, L.Y. and Chen, C. (2008). Multiple trajectory search for large scale global optimization. IEEE Congress on Evolutionary Computation, 3052-3059. 10.1109/CEC.2008.4631210.
[45] Udit, K., Sumit, S., et al. (2015). Enhancing $IACO_R$ local search by mtsls1-bfgs for continuous global optimization. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 33–40.
[46] Vittorio, M. (1999). Exact and approximate non-deterministic tree-search procedures for the quadratic assignment problem. INFORMS journal on computing, 11(4), 358–369.
[47] Vanhoucke, M., and Peteghem, V.V. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem, European Journal of Operational Research, 201, 409-418.
[48] Wei, X. Zeng, H. Zhou, Y. (2010). Cloud theory-based artificial fish swarm algorithm. Computer Engineering and Applications, 46(22), 26-29.
[49] Xianhan, Z., and Yang, C., (2014). A novel chaotic map and an improved chaos-based image encryption scheme, The Scientific World Journal, https://doi.org/10.1155/2014/713541.
[50] Zhang, C., Guomin, C., and Fuyu, P. (2016). A novel hybrid chaotic ant swarm algorithm for heat exchanger networks synthesis, Applied Thermal Engineering, 104, 707-719.