Journal of the
Brazilian Computer Society

RESEARCH Open Access

An accelerated and robust algorithm for ant colony optimization in continuous functions

Jairo G. de Freitas[1,2*] and Keiji Yamanaka[2]

* Correspondence: jairo@iftm.edu.br
[1]Instituto Federal do Triângulo Mineiro (IFTM), Uberaba, MG, Brazil
[2]Universidade Federal de Uberlândia (UFU), Uberlândia, MG, Brazil

Abstract

There is a wide variety of computational methods used for solving optimization problems. Among these, there are various strategies that are derived from the concept of ant colony optimization (ACO). However, the great majority of these methods are limited-range-search algorithms, that is, they find the optimal solution, as long as the domain provided contains this solution. This becomes a limitation, due to the fact that it does not allow these algorithms to be applied successfully to real-world problems, as in the real world, it is not always possible to determine with certainty the correct domain. The article proposes the use of a broad-range search algorithm, that is, that seeks the optimal solution, with success most of the time, even if the initial domain provided does not contain this solution, as the initial domain provided will be adjusted until it finds a domain that contains the solution. This algorithm called ARACO, derived from RACO, makes for the obtaining of better results possible, through strategies that accelerate the parameters responsible for adjusting the supplied domain at opportune moments and, in case there is a stagnation of the algorithm, expansion of the domain around the best solution found to prevent the algorithm becoming trapped in a local minimum. Through these strategies, ARACO obtains better results than its predecessors, in relation to the number of function evaluations necessary to find the optimal solution, in addition to its 100% success rate in practically all the tested functions, thus demonstrating itself as being a high performance and reliable algorithm. The algorithm has been tested on some classic benchmark functions and also on the benchmark functions of the IEEE Congress of Evolutionary Computation Benchmark Test Functions (CEC 2019 100-Digit Challenge).

Keywords: Ant colony optimization, Continuous optimization, Optimization problems, Nature-inspired heuristic approaches

Introduction

Many important and practical problems can be expressed as optimization problems. These problems consist of finding the best solution among an exponentially large set of possible solutions [1]. Optimization is the branch of mathematics that encompasses the study of the quality of optimal solutions and the methods for finding these [2].

© The Author(s). 2021 **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Optimization problems occur in most disciplines such as engineering, physics, mathematics, economics, administration, commerce, social sciences, and even politics. Optimization problems abound in several engineering fields, such as electrical, mechanical, civil, and chemical engineering.

The objective of the optimization process is to find the values of the decision variables that result in a maximum or minimum of a function called the objective function, provided that these values meet a set of restrictions or conditions that, necessarily, must be answered [3].

The numerical decision variables, manipulated by the optimization problems, can be divided into continuous and discrete. A continuous variable can assume an infinite number of values between two points. On the other hand, a discrete variable is one that has a finite number of values between any two points, representing discrete quantities [4].

Many classic optimization algorithms inspired by nature, based on the use of population, have been proposed to solve optimization problems, for which robust solutions are difficult or impossible to find in polynomial time using traditional approaches [5].

The fundamental principle of some of these algorithms uses a constructive method for obtaining the initial population (initial feasible solutions) and a local search technique that gradually improves the solutions generated, considering that the individuals (solutions) of this population evolve according to specified rules, which consider the exchange of information between individuals [6].

Within the evolutionary approach, emphasis is placed on evolutionary algorithms (EA), such as Genetic Algorithms (GA) [7], Differential Evolution (DE) [8], and Genetic Programming (GP) [9], and on the swarm-based optimization algorithms (SOA), such as ant colony optimization (ACO) [10–12], artificial bee colony (ABC) [13], and particle swarm optimization (PSO) [14].

The ant colony optimization strategy was initially created to solve discrete optimization problems, such as combinatorial optimization problems (COPs). These problems can be mapped through graphs, such as the traveling salesman problem [12], shortest path problem [15], vehicle routing [16], and scheduling [17].

Dorigo et al. [10–12] developed the ACO (ant colony optimization). It is a computational strategy, based on the foraging behavior of real ants, which simulates the use of ants, with the objective of solving the traveling salesman problem (TSP). The problem is represented by a graph, where the vertices represent the cities, and the edges represent the path between the cities. To solve the problem, artificial ants are randomly arranged on the edges of the graph. Each ant chooses the next city on its route through a probabilistic calculation, which considers the amount of artificial pheromone distributed on that path and the distance between the two cities. It is not possible to visit the same city twice in one solution. During the route, each ant will release a certain amount of pheromone along each part of the path followed, to influence the decision of the path taken by the ants that come next. The amount of pheromone released is inversely proportional to the length of the path traveled. The more pheromone there is on a path, the more attractive that route becomes. Likewise, the shorter the distance between cities on this path, the more attractive that route becomes. This process is repeated until the total route of the traveling salesman for each of the ants is complete. Hence, the paths between two cities on the graph that the ants traveled along the most will have a greater amount of pheromone, compared to the less traveled paths. The meta-heuristic will be repeated several times, where new ants will be generated at the edges of the graph for further execution. The pheromone along each path will be maintained, but just as in nature, it will gradually evaporate with each run, allowing unprofitable paths to run out of pheromone after several runs of the algorithm. Over the new iterations that are realized, it is possible to find better solutions, as the pheromones of these solutions are reinforced, until the optimum path or a path near to the optimum path is found.

The simplest approach for applying the ACO to continuous problems would be to discretize the real value domain of the variables, that is, convert the real values into a finite range of values [18]. Discretizing continuous variables is a complex task, since the interval where the search will be carried out can be very wide, thus making discretization impossible [19]. Another possible problem is that the optimal solution may require a higher degree of precision than that contemplated by the values that have been discretized. In such cases, if the optimal value is in a space not covered by discretization, it will not be found.

The ACO proved to be very effective for working with discrete variables, but it demonstrated limitations in problems with continuous variables. There are several proposals put forward for solving this problem.

In the Continuous ACO (CACO) method [20], ants start the search process from a base point, called a nest. In each iteration, ants store their best solutions in a set of vectors. These best solutions are used probabilistically to guide the search process in the next iteration. CACO has variations like CACO-DE [21], which performs a discrete coding of continuous variables. The Continuous Interacting Ant Colony (CIAC) [22] uses an interaction mechanism between ants, as well as the information left by pheromones along the paths traveled, which guides the search process. Another approach that realizes optimization of problems with continuous variables is the after Pachycondyla APIcalis (API) [23]. In this method, ants conduct their research in parallel around a starting point, called a nest. The nest is moved periodically, based on the most successful searches. According to Chen et al. [24], the abovementioned algorithms were inspired on the ACO, but do not strictly follow the structure of the ACO. Therefore, they are considered algorithms related to ants and not, real extensions of the ACO for continuous functions.

The algorithms cited below can be classified as ACO extensions for continuous functions [24]. Socha and Dorigo [19] extended ACO so that it could also solve continuous problems, with the name Extended ACO for continuous domains (ACO_R). In this approach, each variable of an ant obtains its new value through a probabilistic sampling of a probability density function (PDF). The most widely used PDF for this process is Gaussian. Gaussian represents a model of distribution of pheromone in the environment, based on the population archive. The population file is initialized with the m possible solutions, where m is a parameter that represents the number of solutions that will be stored. With the representation adopted using the Gaussian function, the values represented with the highest probability of sampling refer to the best solutions found. The value of each variable in a new solution is calculated by sampling using two values: the mean and the standard deviation. The value used as an average is chosen within the population archive, and values that produced better results are more likely to be selected. The standard deviation is calculated based on the distance between the value used as an average and the other solutions in the population archive. When all ants receive their new values for their variables, the pheromone is updated. At each iteration, the pheromone is updated through a process, in which the new solutions found by the ants are added to the population archive, which will always remain with the m best solutions.

There are other algorithms derived from ACO_R for the optimization of problems with continuous functions, such as Diversity ACO_R ($DACO_R$) [25]. $DACO_R$ is more appropriate for continuous optimization problems with a large number of dimensions, as it tries to avoid loss of diversity in the first iterations. The objective is to preserve diversity for as long as possible, to explore more regions of the search space, before the algorithm converges. Another variation is the Incremental Ant Colony Algorithm with Local Search ($IACO_R$-LS) [26]. $IACO_R$-LS is a variation of ACO_R, which has a research diversification strategy that results in an increase in the solutions archive. In addition, a local search procedure is added to accelerate the process of finding the solution. The Unified Ant Colony Optimization (UACOR) [27] includes components of the ACO_R, $DACO_R$, and $IACO_R$-LS algorithms, being able to instantiate each one, choosing specific components of the algorithm and allowing for the automatic adjustment of various parameters. Adaptive Multimodal Continuous Ant Colony Optimization (AM-ACO) is an extension of ACO_R for multimodal optimization [28]. AM-ACO uses niching strategies, dividing the total population into smaller parts rather than working with the total population, and executing an adaptive adjustment of some parameters in this first stage. Subsequently, a differential evolution mutation operator is used to accelerate the convergence speed. Finally, a local search process is executed, based on the Gaussian distribution. The Ant Colony Optimization Algorithm for Continuous Domains Based on Position Distribution Model of Ant Colony Foraging [29] is based on the principle that the ants' food source is everywhere in the continuous space, and only the quality of the food source is different. Each ant checks the quality of its position; checks the pheromone concentration in the rest of the space, using a group pheromone density function; and migrates to areas of higher concentration, where it is able to explore unknown regions during this movement.

All the methods of optimization of continuous functions above belong to a category called limited-range-search algorithms, that is, they find the ideal solution, but within the predetermined domains [24]. The problem with these algorithms is that they are dependent on the initial domain. If the initial domain is not estimated correctly and does not have the optimal solution, the algorithm will not obtain the correct solution, as ants cannot leave the domain. In real-world problems, where it is not always possible to precisely define the search space, with its set of restrictions, these algorithms may not solve the problem in the most appropriate manner. Chen et al. [24] propose a broad-range search algorithm, which is able to find the optimal solution, even if the initial domains provided do not contain the optimal values. Robust Ant Colony Optimization for continuous functions (RACO) uses the grid method to discretize continuous variables and applies self-adaptive strategies for domain adjustment, pheromone increment, domain division, and ant size, to enable the search to be successfully executed. This method is used successfully in cases where the initial domain has the optimal values for solving the problem, and in cases in which it does not.

The objective of the proposed article is to contribute to the improvement of the results of the application of the ACO in problems of continuous optimization, through the presentation of a new broad-range search algorithm, derived from RACO, called the Accelerated and Robust Algorithm for Ant Colony Optimization in Continuous Functions (ARACO). ARACO uses adaptive parameters related to domain adjustment. The acceleration of these parameters, in opportune moments, leads the domain that contains the optimal solution to be found more quickly, as well as allows, within the correct domain, ants to converge to the optimal solution more quickly. In this case, ARACO allows the optimal values for the variables to be found in a lower number of function evaluations, when compared to the other algorithms.

With the acceleration of the adaptive domain adjustment parameters, it is possible that the domain values have become stagnant in a region, which makes the best solution found to be a local minimum. In this case, ARACO allows the domain of the variables to be gradually increased around the best solution found so far, allowing the determination of a new domain that is outside the region where the optimal location is found. In this case, ARACO is able to reach regions of the domain that are not reached by its predecessor RACO, thus allowing the optimal values to be achieved with a higher success rate.

This article is organized as follows. In the second section, the main features of ARACO will be introduced and a step-by-step guide to the algorithm will be shown. In the third section, ARACO's experimental results will be presented. These results will be compared with the results of other methods based on ant colony optimization in continuous functions and with methods that use other principles for optimization. Finally, in the fourth section, conclusions about the research and future work that can be executed will be presented.

ARACO

In order to present the proposed metaheuristic, this section will define the problem that will be addressed by the article, in addition to the main features and steps taken by ARACO in the search for the optimal solution.

Problem definition

The proposed algorithm aims at solving problems of continuous optimization, locating the optimal solution, regardless of the domain provided. A continuous optimization problem can be formally defined with the following model: $P = (X, \Omega, f)$ [24], where X is a solution vector with n continuous variables x_i ($i = 1, 2, ..., n$), Ω is a set of restrictions that must be met by the variables, and f is the objective function to be optimized. In the case of a minimization problem, the objective is to find the value of X^*, which minimizes the function: $f(X^*) \leq f(X), \forall X \in S$. If this is a maximization problem, the objective is to find the value of X^*, which maximizes the function: $f(X^*) \geq f(X), \forall X \in S$, with S representing the initial domain from where the search for the value of X will be carried out.

Discretization of variables

The ARACO algorithm uses the grid method for the discretization of variables. Through it, the defined continuous domain is converted into a discrete domain. This process happens at each iteration of the algorithm, as each iteration defines a new continuous domain for *X*. By definition, *X* is a vector of solutions with *n* dimensions with continuous variables x_i $(i = 1, 2, ..., n)$.

In the adopted process, a continuous domain (x^i_{min}, x^i_{max}) will be determined for each iteration, where x^i_{min} represents the lowest domain value for variable *x*, in dimension *i*, while x^i_{max} represents the highest value of the domain for variable *x*, in dimension *i*. The algorithm will seek the solution of the problem within this domain. A variable *k* is used for discretization. The initial domain will be divided into *k + 1* parts for each of the *n* variables.

At this moment, the domain will be divided into *k + 1* discrete values. The discrete values of the domain can be represented by a matrix of *n x (k + 1)* dimensions, as shown in Fig. 1, where *n* represents the number of variables in the problem. The Eq. (1) is used to calculate the value of each element of the matrix.

$$h_i = \left(x^i_{\max} - x^i_{\min}\right)/k \tag{1}$$

where *i* represents each variable of the problem, varying between 1 and *n*.

Building a new solution

The main role of ants in ARACO is the construction of new solutions. Each ant will store a possible solution, that is, a value for each variable of the optimization problem. To build a new solution, the ants will initially take several random routes to distribute the initial pheromone, which will be used as a basis to guide the construction of the solutions together with the heuristic value of the function that will be optimized. In this process, each ant randomly selects, for each variable, a value from the possible discrete values that were obtained through the grid method. After this step, the random solutions created will have their heuristic value calculated, according to the function that will be minimized or maximized. The random routes created are ordered, according to their heuristic value, and only the best solutions created will have their pheromone distributed.

To distribute the pheromones, a matrix, called τ, will be used. This matrix has *n x (k + 1)* dimensions. The pheromone values of the best solutions are deposited in the τ pheromone matrix, according to (2).

$$\tau_{ij} = \tau_{ij} + Q/f \tag{2}$$

where τ_{ij} represents the pheromone value in variable *i*, at point *j*, mapped according to the discretization of the variables; *f* represents the heuristic value of the solution;

x^1_{min}	$x^1_{min}+h_1$	$x^1_{min}+2*h_1$	...	$x^1_{min}+k*h_1$
x^2_{min}	$x^2_{min}+h_2$	$x^2_{min}+2*h_2$	...	$x^2_{min}+k*h_2$
...	...	...	...	...
x^n_{min}	$x^n_{min}+h_n$	$x^n_{min}+2*h_n$	...	$x^n_{min}+k*h_n$

Fig. 1 Domain discrete value matrix

and Q represents the adaptive pheromone increment, which will be described later, but which is calculated by (3).

$$Q = 10^{OM_{min}+1} \tag{3}$$

where $OM_{\min}$ represents the order of magnitude, that is, the exponent value of the best solution found for the function.

When the pheromone of the best randomly generated solutions is deposited in the matrix, ARACO will go to a new stage. From that moment, new ants will be generated and they will choose their routes, using a probability calculation among the available routes, represented by (4).

$$p_{ij} = \tau_{ij} / \sum_{i=1}^{k+1} \tau_{ij} \tag{4}$$

where p_{ij} represents the probability that, in variable i, point j will be selected as the route for the new ant. It is important to highlight that, based on the previous equations, the probability of a value being selected as a solution for a new ant is calculated using information related to the heuristic value and the amount of pheromone that the route has, in the same way adopted in the basic concepts of ACO. Thus, solutions that have a lower heuristic value and a higher amount of pheromone are more likely to be selected as a route for new ants.

After the ants create all new routes, the pheromone evaporation process will occur. This process prevents the pheromone from accumulating in only a few points, making it difficult to diversify solutions. The new pheromone value at each point will be given by (5).

$$\tau_{ij}^{new} = (1-\rho)*\tau_{ij}^{old} \tag{5}$$

where ρ represents the evaporation rate and its value must be determined by a number between 0 and 1. In the ARACO algorithm, the recommended value for ρ is 0.5. This value was determined empirically, that is, exhaustive tests were performed to determine the value that produced the best results.

After evaporation, the best route among the created routes will have its pheromone reinforced in the pheromone matrix. The process of building new solutions will be repeated, until the number of ants determined by variable m creates their routes, so that the solutions are evaluated again and the pheromone of the best route is reinforced again. The number of times this process is repeated is determined by the variable *nc_max*.

Adaptive domain adjustment

The adaptive domain adjustment strategy allows the optimal solution to be found, even if the initial assigned domain does not contain the optimal values. To this end, the domain must be adjusted automatically for each iteration. The domain adjustment process is performed by analyzing the τ pheromone matrix. Each line i of the matrix τ represents the pheromones distributed in a variable i, which has the values mapped in the matrix in Fig. 1.

It can be said that if the pheromone is concentrated near the center of the domain, there is a greater possibility that the ideal solution is located within the domain. On the other hand, if the pheromone is concentrated near the ends of the domain, there is a greater possibility that the ideal solution is located outside the domain. This analysis and adjustment is executed variable by variable, since it is possible that for one variable the solution is outside the domain and for another variable the solution is within the domain.

To decide how the adjustment will be made, the algorithm uses the variable θ that represents the percentage of *(k+1)* positions, which will determine if the best solution is inside or outside the current domain. It also uses the variable r_i, which represents which position, between 0 and (*k+1*), has the highest concentration of pheromone. Using these two variables, in an example where the variable θ has a value of 0.2, that is, if r_i is located in the first or last 20% of (*k+1*) positions, the ideal solution tends to be located outside the domain. The recommended value for the variable θ is between 0.1 and 0.3. This range of values was defined in this way, as it is the range determined in the RACO algorithm. The values of the parameters that exist in the two algorithms are the same, in order to be able to better evaluate the new features implemented in ARACO. Thus, if $r_i \leq \theta * (k+1)$ or $r_i \geq (1 - \theta) * (k+1)$, the adaptive adjustment of the domain must be executed to move and expand the current domain, to find a domain that includes the ideal solution.

In this case, the new values $x^i_{\min}$ and $x^i_{\max}$ of the variable i in the domain must be determined considering r_i as the center of the new domain, as in Eqs. (6–7).

$$x^i_{\min} = r_i - \left({}^{k}\!/_{2} + \Delta_1\right) * h_i \tag{6}$$

$$x^i_{\max} = r_i + \left({}^{k}\!/_{2} + \Delta_1\right) * h_i \tag{7}$$

where the parameter Δ_1 must be small, in order that the new domain does not become much bigger than the old domain. In the ARACO algorithm, the recommended value is 1.25, but it is important to highlight that the value is changed dynamically to speed up the solution conversion process, as shown in item 2.8 of the article. This initial value was defined in this way, as it is the value used in the RACO algorithm.

The inverse situation, when the ideal solution tends to be located within the current domain, is represented when $\theta * (k+1) \leq r_i \leq (1-\theta) * (k+1)$. In this scenario, the current domain must be reduced in order to execute a more detailed search.

In this case, the new values $x^i_{\min}$ and $x^i_{\max}$ of the variable i in the domain must be determined following the Eqs. (8–9).

$$x^i_{min} = x^i_{min} + \left(x^i_{max} - x^i_{min}\right) * \Delta_2 \tag{8}$$

$$x^i_{max} = x^i_{max} - \left(x^i_{max} - x^i_{min}\right) * \Delta_2 \tag{9}$$

where the parameter Δ_2 determines the percentage of domain reduction, its value should not be too large so that the domain is not reduced enough to produce a condition where the ideal solution is no longer within it. For each iteration that uses this adjustment, the new domain will be $2 * \Delta_2$ smaller than the old domain. In the ARACO algorithm, the recommended value is 0.05, but it is important to highlight that the value is changed dynamically to speed up the solution conversion process, as shown in item 2.8 of the article. This initial value was defined in this way, as it is the value used in the RACO algorithm.

There is another mechanism for speeding up the process in situations where the ideal solution is outside the current domain. In this situation, case $r_i \leq \theta * (k+1)$, that is, the domain displacement was executed in the direction of the value of $x^i_{\min}$, the value of $x^i_{\min}$ will be preserved and the value of x^i_{max} will be changed according to (10).

$$x^i_{\max} = x^i_{\max} - \left(x^i_{\max} - x^i_{\min}\right) * \Delta_3 \tag{10}$$

On the other hand, if $r_i \geq (1-\theta) * (k + 1)$, that is, the domain displacement was executed in the direction of the value of x^i_{max}, the value of x^i_{max} will be preserved and the value of x^i_{min} will be changed according to (11).

$$x^i_{\min} = x^i_{\min} + \left(x^i_{\max} - x^i_{\min}\right) * \Delta_3 \tag{11}$$

where parameter Δ_3 determines the percentage that the domain will be reduced, at its lower or upper limit, depending on the value r_i. In the ARACO algorithm, the recommended value is $2 * \Delta_2$, that is, 0.1, but it is important to highlight that the value is changed dynamically to speed up the solution conversion process, as shown in item 2.8 of the article. This initial value was defined in this way, as it is the value used in the RACO algorithm.

Adaptive pheronome increment

To calculate the values of the τ pheromone matrix, the heuristic value of the solution (f) and the value of the variable Q are used. The value of the variable Q cannot remain constant throughout the execution of the algorithm, because as the algorithm interactions are executed, the value of f is changed, since better solutions are found and the domain is changed. With the value of f being altered at each iteration and, as $\tau_{ij} = Q / f$, the values of Q and f could become very disproportionate, causing the pheromone matrix to possess similar values, making it impossible for the algorithm to converge. It is important to note that the problem of the pheromone matrix having only similar values could not be avoided if was set for Q a very low value or very high, as in any case, the variation in the value of f would cause the values to become similar at some moment.

The solution found was to define the pheromone increment of the τ_{ij} matrix through an adaptive strategy, making the variable Q as different as possible at each iteration, as the domain is changed. This value must always be proportional to the heuristic value of the best randomly generated solution, at the beginning of each iteration, therefore $Q = 10^{OM_{min}+1}$. The creation of random routes at the beginning of each iteration allows for an initial pheromone to exist before ants build their routes. This initial pheromone will cause the routes to be determined by the ants in a non-random manner, as they will be directed by the initial pheromone, causing the convergence of the algorithm to be increased.

Adaptive domain division

One of the main characteristics of ARACO is to be able to find the best solution, even if it is not contained in the initial domain provided. This is only possible because an adjustment is made in the domain at each iteration, using among other parameters the variable k, which also directly influences the domain discretization process. At each iteration, the domain is changed and discretized. If the value of k is very low over the entire algorithm, the domain will be converted into a few discrete points, which may not be able to direct the ants to the ideal solution of the problem, as it is not close to an optimal region. On the other hand, if the value of k is very high over the whole algorithm, the domain will be converted into many discrete points, which may have similar heuristic values, making it slow to find the most promising region among those available.

The fact is that the degree of precision of the domain discretization depends on the variable k, which will have its value defined through an adaptive method in ARACO. At the beginning of the execution, k may have a lower value, so that the algorithm can quickly converge to a promising area. However, at some point, the value of k will limit the algorithm, causing it not to find better solutions with each iteration. The value of k must then be increased by one, so that the accuracy of the search is increased and the algorithm can find solutions that were not possible with the previous value of k. The recommended initial value for variable k is 11. This initial value was defined in this way, as it is the value used in the RACO algorithm.

Adaptive number of ants

Another condition that influences the success of the algorithm is the number of ants that will execute the search at each iteration. The number of ants depends directly on the discretization process of the domain, because if the continuous domain is converted into a few points, few ants will be able to find the most promising region to search. However, if the domain is converted at many points, many ants will be needed to find the best solutions.

Thus, the number of ants used must also be an adaptive parameter, because when the domain is converted into a few points, that is, k has a low value, the m number of ants is also low. As the value of k increases, due to the need for greater precision in the search for the best solutions, the number of ants should also increase. The Eq. (12) determines the number of ants.

$$m = k + \Delta_m \tag{12}$$

where the parameter Δ_m represents how much m must be greater than k, to enable ants to find the best solutions, without spending a lot of time searching. The recommended value for Δ_m is 2. This initial value was defined in this way, as it is the value used in the RACO algorithm.

Acceleration of adaptive domain adjustment parameters

In order to speed up the convergence process of the ARACO algorithm for the best solution, adaptive values are used for the parameters Δ_1, Δ_2, and Δ_3, responsible for adjusting the domain at each iteration. The parameters are adjusted under certain situations to accelerate the algorithm, but then return to their initial values when this acceleration is no longer recommended.

If the optimal solution is located near to the center of the current domain, it is necessary that the domain be reduced, using (8–9), so that a more detailed search is executed. The parameter responsible for determining how much the domain will be reduced is Δ_2 and its initial value is 0.05. ARACO verifies whether the point with the highest concentration of pheromones is located in the central position of the current domain or in the position before or after the center, in other words, whether $r_i = (k+1) / 2$ or $r_i = ((k+1) / 2) - 1$ or $r_i = ((k+1) / 2) + 1$. In these cases, parameter Δ_2 will increase by 0.05, as the ideal solution tends to be located near to the center of the domain; thus, it is possible to ignore some values located at the edges of the domain, while reducing the domain at a faster speed, so that ants can find the best solution in less iterations. In other cases, where the ideal solution tends to be within the domain, but not near to the center, parameter Δ_2 will increase by 0.005. The acceleration of parameter Δ_2 has a limit and this must be executed until it reaches a maximum value of 0.15. Noteworthy here is that, when the point with the highest concentration of pheromones is not located within the domain, Δ_2 returns to its initial value.

If the ideal solution is located outside the current domain, it is necessary that the domain be moved and expanded, using (6–7), so that the correct domain is found. The parameter responsible for determining how much the domain will be displaced and expanded is Δ_1, where its initial value is 1.25. ARACO verifies whether the point with the highest concentration of pheromones is located on the lower or upper edge of the domain, in other words, whether $r_i = 1$ or $r_i = (k + 1)$. In these cases, the parameter Δ_1 will be increased by 0.25, because as the ideal solution tends to be far from the edge of the domain, a more significant displacement and increase can be performed, so that the correct domain can be located more quickly. In other cases, where the ideal solution tends to be outside the domain, but not so far from the edge, parameter Δ_1 will increase by 0.025. The acceleration of parameter Δ_1 has a limit and must be executed until it reaches a maximum value of 1.75. Emphasis is here placed on the fact that when the point with the highest concentration of pheromones is no longer located outside the domain, Δ_1 returns to its initial value.

These increment values for the variables Δ_1 and Δ_2 were determined empirically, that is, exhaustive tests were performed to determine the values that produced the best results.

The parameter Δ_3 is also dynamically adjusted, but its variation depends directly on the adjustment of Δ_2, as previously stated $\Delta_3 = 2 * \Delta_2$.

Expansion of the domain around the best solution

In some cases, the acceleration of the parameters Δ_1, Δ_2, and Δ_3 can cause the algorithm to converge to a region with a local minimum and cannot get out. For such a situation of stagnation of the algorithm, ARACO has a domain adjustment strategy, which causes a new domain to be defined around the best solution found so far. This adjustment is executed when there are 30 iterations of the algorithm without an improvement of at least 10% in the heuristic value of the best solution found. A new domain will be determined for only one of the variables, so that this expansion can happen gradually. The choice of the variable that will have its domain changed is defined through a test, which locates the variable that would proportionally have less variation with the extension of the domain. This variable tends to be more stagnant than the others. The Eqs. (13–14) determine the value of the new domain.

$$x^i_{min} = best^i - {}^{k}\!\Big/_{2} * \Delta_1 * H_i * e^i \tag{13}$$

$$x^i_{max} = best^i + {}^{k}\!/_{2} * \Delta_1 * H_i * e^i \tag{14}$$

where $best^i$ represents the value of the best solution found so far, H_i represents the value of h_i when the best solution was found and the parameter e^i represents to which degree the new domain needs to expand in order to leave the local minimum region.

The parameter e^i, for each variable i, is initialized with 1. Whenever 30 iterations are performed and the best solution remains stagnant, it will have its value doubled to the variable that had its domain changed. When the best solution leaves the local minimum region, that is, when a new solution is found that is at least 10% better than the best solution found, the parameter returns to 1, for all variables i. In this case, there is the guarantee that the new domain generated will gradually increase around the best solution found, allowing new values to be located outside regions of local minimum.

The value of the parameters responsible for detecting and treating stagnation were determined empirically, that is, exhaustive tests were carried out to determine the values that produced the best results.

As previously mentioned, the H_i parameter represents the value of h_i when the best solution was found. However, for the new domain not to become so small that it is necessary to execute this extension several times, which would impair the performance of the algorithm, or so large that it is not possible to find a best solution in just 30 iterations, the value of the H_i parameter must be within the range $0.1 \leq H_i \leq 1$. Also noted here is that, after the domain expansion process around the best solution found, it is necessary to reset Δ_1, Δ_2 and Δ_3, to the initial values.

The steps of the ARACO algorithm

After detailing the main features of ARACO, this topic presents the structure of the algorithm. First, a step-by-step will be shown, describing in detail all the actions performed by the algorithm, from the assignment of the initial parameters, until the completion of the algorithm, when the termination condition is reached. Following this, a flowchart is presented that represents the sequence and the interaction between the actions of the algorithm.

Step by step

In this section, the main steps followed by the algorithm to find the optimal solution are shown and explicated. One of the main improvements generated by the ARACO algorithm is related to obtaining a better performance, with the execution of step 17. Through this step, the algorithm is able to find the optimal value in a smaller number of iterations compared to other algorithms, thanks to the acceleration of the responsible parameters by adjusting the domain at appropriate times. Another improvement is related to achieving greater reliability, with the execution of step 18. Through this step, the algorithm is able to find the optimal value in practically all the performed executions, by applying the process that treats algorithm stagnation. This process expands the domain around the best solution found so far, through preventing the algorithm from getting stuck in a region of local minimum. This occurs regardless of whether the initial domain provided contains or not the optimal solution, allowing the algorithm to find the solution using fewer function evaluations than the other algorithms cited in the article, as will be shown in item 3. All steps for the algorithm are listed below:

Step 1—Initialize the values of all variables, such as k, θ, ρ, nc_max, Δ_m, Δ_1, Δ_2, Δ_3, F_{MIN}, t, and the number of iterations without improvement. In addition to the value of x^i_{min} and x^i_{max}, which represent the initial domain for each variable, i.e., the minimum and maximum value of the dimensions of the domain and the termination condition of the algorithm.
Step 2—Divide the initial domain into k equal points for each variable, according to (1). At that moment, the process of discretization over the variables occurs using the grid method, in which the continuous domain provided is converted into a discrete domain. This process is important because the rest of the algorithm will work with the discretized domain. The result is shown in Fig. 1.
Step 3—Initialize the τ pheromone matrix, which represents the amount of pheromone stored at each point in the domain discrete value matrix. In addition, initialize the adaptive number of ants m, which has a dynamic value, due to the fact that when the domain is divided into a few discrete points, the small number of ants are sufficient to perform the search. However, when the domain is divided into many discrete points, will be necessary too many ants to perform the search. Similarly, initialize the variable f_{min}, which stores the heuristic value of the best solution found in the current iteration, and the variable nc, which controls the number of times that ants will generate new routes in each iteration.
Step 4—Generate some possible random solutions.
Step 5—Calculate the heuristic value of the random solutions created, according to the function that will be minimized or maximized by the algorithm. Sort the solutions according to the heuristic value.
Step 6—Calculate the value of the adaptive increment Q of the pheromone in this iteration, according to (3), so that the value of Q remains proportional to the heuristic value of the solutions found in the current solution, thus maintaining a proportional pheromone distribution in the τ pheromone matrix. Distribute the initial pheromone over the best solutions created, to supply the τ pheromone matrix with some initial information, and as such guide the first ants towards more attractive solutions, thus accelerating the algorithm conversion process.
Step 7—Create new routes for ants using the roulette wheel method by use of (4). Each ant creates its new route, variable by variable, performing a probabilistic calculation, which considers the pheromone existing at each point in the pheromone matrix. As the calculation of the amount of pheromone uses the heuristic value of the solutions, it can be said that the same criteria defined in the initial concept of ACO are used.
Step 8—Evaluate the solutions generated in this nc iteration, according to the function that will be minimized or maximized by the algorithm. The objective of this step is to find and store the best solution, among the solutions created in this iteration.
Step 9—Execute the pheromone evaporation process, according to (5). This process ensures that paths that have less pheromone become less and less attractive, until they

are no longer covered. While the paths that have more pheromone are able to guide ants in the search for better solutions.
Step 10—Increase the pheromone at all points of the best solution found in this iteration, using (2), so that the pheromone matrix always has a greater amount of pheromone on the paths that are more attractive.
Step 11—Update the value of f_{min} to the value of the best solution found in the *nc* current route, if this value is less than f_{min}, the f_{min} must store the heuristic value of the best solution found in the current iteration.
Step 12—Increase the value of *nc*. If *nc* <= *nc_max*, go back to step 7, as the number of routes necessary for the domain adjustment was not generated. Otherwise, this is, if *nc* > *nc_max*, proceed to step 13.
Step 13—Check if the best solution found in this iteration (f_{min}) is the best global solution found (F_{MIN}). If so, go to step 14. Otherwise, jump to step 15.
Step 14—Store the value of the new global best solution in the variable F_{MIN} and verify if the new solution is 10% better than the previous one to define if the algorithm is stagnant and reset the H_i value. Knowledge of when the algorithm is stagnant is important, as it allows for the subsequent execution of a strategy to deal with this state of stagnation, allowing the algorithm to leave the local minimum region and continue on to find better solutions. To this end, it is necessary to store the value of H_i, at the moment when a new value of F_{MIN} is found.
Step 15—Increase the variable that controls the amount of iterations without improvement to detect the stagnation of the algorithm and also the variable *t*. The variable *t* will increment *k*, responsible for the adaptive domain division, when the value of *t* is 15. The variable *k* is an adaptive parameter, which needs to have a small initial value, so that the algorithm can locate a promising region, but its value is gradually increased to enable new searches to be carried out with a greater degree of precision, when better results are not attained.
Step 16—Perform the adaptive domain adjustment process, based on the position of the τ pheromone matrix which has a higher concentration of pheromones. For this, if the pheromone is concentrated near the edge of the domain, the best solution found in this iteration tends to be located outside the domain. In this case, to locate a domain that contains the optimal solution, it is necessary to expand and adjust the domain using (6–7, 10–11). On the other hand, if the pheromone is concentrated away from the domain edge, the best solution found in this iteration tends to be within the domain. In this case, it will be necessary to reduce the domain using (8–9) to perform the search in the next iteration with a greater degree of precision.
Step 17—Increase the speed of convergence of the algorithm, accelerating the adaptive domain adjustment parameters, if appropriate. In other words, accelerate the value of Δ_1 when the pheromone is concentrated around the central positions of the domain and, accelerate the value of Δ_2 when the pheromone is concentrated at the upper or lower edge of the domain. When these opportune situations cease to exist, parameters Δ_1 and Δ_2 return to their initial values.
Step 18—Increase the success rate of the algorithm, i.e., the number of executions in which the algorithm can find the optimum value, through the process of treating the algorithm stagnation. That is, if the algorithm is stagnant, redefine the domain and

gradually expand it around the best solution found so far, using (13–14), allowing the algorithm to escape from a local minimum region.

Step 19—Check if the termination condition has been reached. If so, go to step 20. Otherwise, go back to step 2.

Step 20—Finalize the algorithm.

Flowchart

Figure 2 describes the entire process performed during the ARACO algorithm. It is important to highlight that each action performed in the flowchart below is identified with a number, which is the step number of the section Step by step to which it corresponds.

Experimental results

In this section, the efficiency of ARACO will be verified, by performing tests in the same scenarios as the tests performed by the RACO algorithm, since ARACO is proposed as an algorithm derived from RACO. ARACO will be compared to RACO throughout this section. At first, the tests were performed in a scenario where the problem is solved by a limited-range-search algorithm, i.e., the initial domain provided contains the ideal solution for the function. Tests were also performed in a scenario where the problem is solved by a broad-range search algorithm, i.e., the initial domain provided does not contain the ideal solution. After these initial tests, new tests were performed using the benchmark functions used in the IEEE Congress of Evolutionary Computation Benchmark Test Functions (CEC 2019 100-Digit Challenge). In both scenarios tested, the values of the main ARACO parameters are $k = 11$, $\theta = 0.2$, $\rho = 0.5$, $nc\_max = 50$, $\Delta_m = 2$, $\Delta_1 = 1.25$, $\Delta_2 = 0.05$, $\Delta_3 = 0.1$. The parameter values were defined in this way, as these are the values used in the RACO algorithm, and since the ARACO algorithm was compared in all scenarios with the RACO algorithm, it is necessary that the initial parameters of the two have the same values, so that the comparison is fairer. Noted here is that the values of k, Δ_1, Δ_2 e Δ_3 are changed during the execution of the algorithm, so the assigned values are only the initial values. Another parameter of interest here is that 100 random routes are generated at each beginning of the algorithm iteration and the 30 routes that generate the lowest heuristic value for the function that will be optimized will deposit their pheromone in the τ matrix. This provides the algorithm with the initial information needed to find solutions. Another important factor is that the algorithm is considered stagnant, and therefore, the domain adjustment strategy is executed around the best solution found, when 30 iterations are executed without finding a solution that has a heuristic value, which is at least 10% less than the heuristic value of the best solution found so far. The value of the parameters responsible for detecting and treating stagnation were determined empirically, that is, exhaustive tests were carried out to determine the values that produced the best results.

All comparisons between the algorithms are based on the number of function evaluations performed until the termination condition is reached. Noteworthy here is this was the criterion chosen, due to the other algorithms cited in the article also using this criterion. Thus, it becomes possible to compare ARACO to these, as they use the same termination condition. ARACO considers a function evaluation when the process of

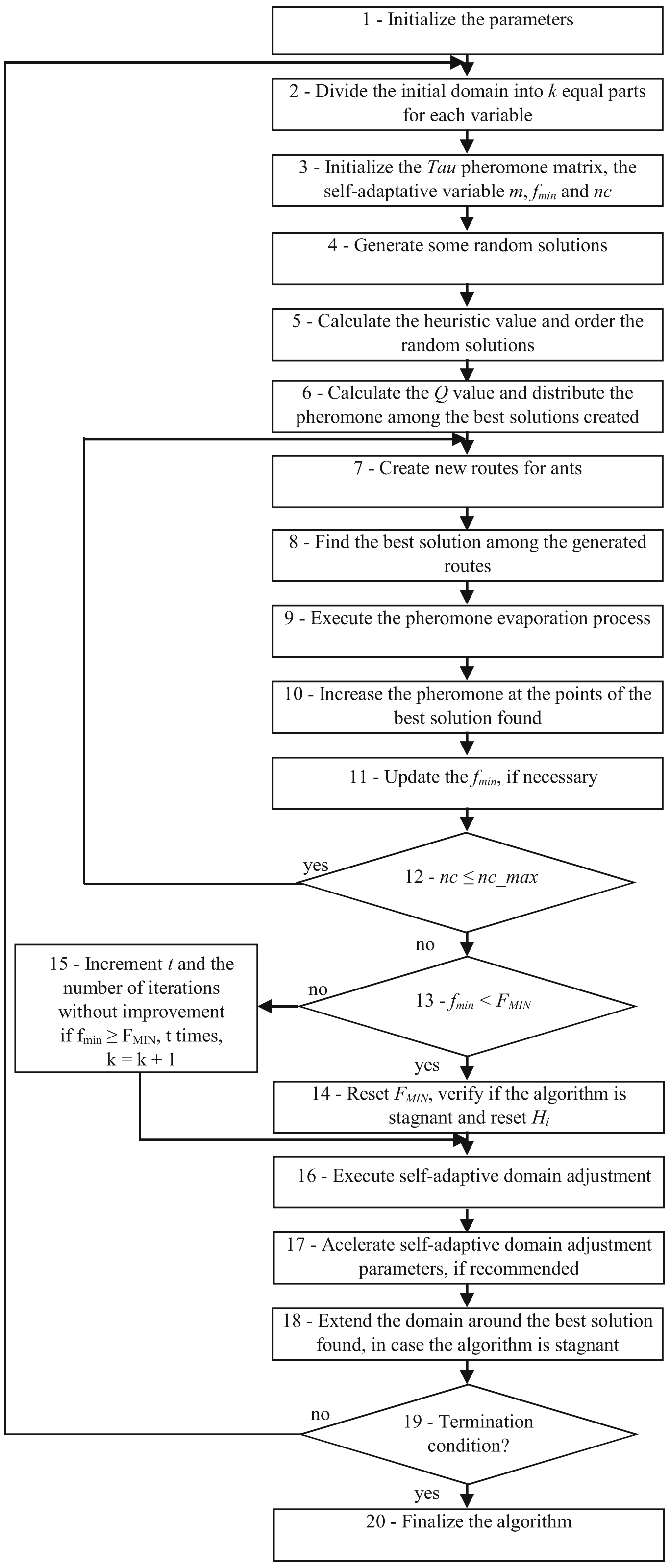
1 - Initialize the parameters
2 - Divide the initial domain into k equal parts for each variable
3 - Initialize the *Tau* pheromone matrix, the self-adaptative variable m, f_{min} and nc
4 - Generate some random solutions
5 - Calculate the heuristic value and order the random solutions
6 - Calculate the Q value and distribute the pheromone among the best solutions created
7 - Create new routes for ants
8 - Find the best solution among the generated routes
9 - Execute the pheromone evaporation process
10 - Increase the pheromone at the points of the best solution found
11 - Update the f_{min}, if necessary
12 - $nc \leq nc\_max$
yes
no
13 - $f_{min} < F_{MIN}$
no
yes
14 - Reset F_{MIN}, verify if the algorithm is stagnant and reset H_i
15 - Increment t and the number of iterations without improvement if $f_{min} \geq F_{MIN}$, t times, $k = k + 1$
16 - Execute self-adaptive domain adjustment
17 - Acelerate self-adaptive domain adjustment parameters, if recommended
18 - Extend the domain around the best solution found, in case the algorithm is stagnant
19 - Termination condition?
no
yes
20 - Finalize the algorithm

Fig. 2 ARACO's flowchart

adaptive adjustment of the domain is carried out, which refers to step 16 of the flowchart presented in the section “Step by step”.

In all tests performed, the programming language used was MATLAB, in version R2016A. The equipment used for the tests was a notebook, with an AMD A10-4600M Quad-core processor with a clock speed of 2.3GHz, and 8GB of DDR3L RAM, with the Windows 10 - 64-bit operating system.

Tests where the initial domain contains the optimal solution

The results obtained in the ARACO algorithm were compared with the results found in the ACO_R algorithm [19] and with the methods it cites, in addition to the results found by the RACO algorithm [24], which was the reference used for the creation of ARACO.

As in Socha and Dorigo [19], the comparisons are divided into three groups: probability-learning methods that model and sample probability distributions, metaheuristics developed for combinatorial optimization and adapted to continuous domains, and methods inspired on the behavior of ants. In the three groups of comparisons, the comparison with ACO_R and RACO will be added.

For an impartial comparison between the algorithms, the established criterion was the number of function evaluations performed, instead of the execution time or other measures that may be related to the performance of the equipment or the programming language used.

Probability-learning methods that model and sample probability distributions

The methods for comparison in this section are three versions of evolutionary strategies: (1+1) ES (Evolution Strategy with 1/5th-Success-Rule), CSA-ES (Evolution Strategy with Cumulative Step Size Adaptation), CMA-ES (Evolution Strategy with Covariance Matrix Adaptation), IDEA (Iterated Density Estimation Evolutionary Algorithm), and MBOA (Mixed Bayesian Optimization Algorithm). The population size used in the above algorithms is chosen for each algorithm-problem pair [30]. The smallest population is selected from the set $p \in [10, 20, 50, 100, 200, 400, 800, 1600, 3200]$. ACO_R parameters are m (number of ants) = 2, n (speed of convergence) = 0.85, q (locality of the search process) = 10^{-4}, and k (archive size) = 50. The RACO parameters are the same as ARACO.

Each benchmark function was run 20 times and the comparison criterion used in this section is the median number of function evaluations (MNFE) executed until the termination condition was reached. The Eq. (15) determines the termination condition.

$$|f - f*| < \in \tag{15}$$

where f is the best heuristic value found by ARACO, f^* is the optimal value found in the literature for the benchmark function, and $\in$ is 10^{-10}.

Table 1 shows the benchmark functions used in this scenario. All have 10 dimensions. The Function column shows the name of the benchmark function. The Formula column shows the formula used to calculate the value of the function that will be minimized or maximized. The Optimal column shows the ideal value for each variable of the function. The Minimum column shows the minimum value of the function, when the optimal values for each of the variables are found

Table 1 First part of benchmark functions

Function	Formula	Optimal x*	Minimum f(x*)
Sphere	$f(\vec{x}) = \sum_{i=1}^{n} {x_i}^2$	$\vec{x}^* = (0, \ldots, 0)$	$f_{min} = 0$
Ellipsoid	$f(\vec{x}) = \sum_{i=1}^{n} (100^{\frac{i-1}{n-1}} x_i)^2$	$\vec{x}^* = (0, \ldots, 0)$	$f_{min} = 0$
Cigar	$f(\vec{x}) = {x_1}^2 + 10^4 \sum_{i=2}^{n} {x_i}^2$	$\vec{x}^* = (0, \ldots, 0)$	$f_{min} = 0$
Tablet	$f(\vec{x}) = 10^4 {x_1}^2 + \sum_{i=2}^{n} {x_i}^2$	$\vec{x}^* = (0, \ldots, 0)$	$f_{min} = 0$
Rosenbrock	$f(\vec{x}) = \sum_{i=1}^{n-1} [100({x_i}^2 - x_{i+1})^2 + (x_i - 1)^2]$	$\vec{x}^* = (1, \ldots, 1)$	$f_{min} = 0$

The results used to compare ACO$_R$, (1 + 1) ES, CSA-ES, CMA-ES, IDEA, and MBOA were obtained from Dorigo et al. [19], whereas the results used for comparison with RACO were obtained from Chen et al. [24]. Table 2 shows the median number of function evaluations (MNFE) necessary for each method to find the termination condition. The median is shown in parentheses, after 1.0, only for the algorithm that has the best result for a function, the other medians can be calculated proportionally based on the value shown in the table. For example, if an algorithm has its median shown with 1.0 (86), it means that it is the algorithm that has the best median and the value of this median is 86. On the other hand, if an algorithm that does not have the best result, its median is shown in the table as 1.8, that is to say that the median is 1.8 multiplied by the best result found for that function. Some methods failed to find the optimal value in all runs for the Rosenbrock function. These cases were represented in the table with an *. Worthy of mention here is that in four out of the five functions, ARACO is able to optimize the functions with less than half the number of function evaluations of the second best algorithm. If the performance is compared with the other algorithms, the difference becomes even more impressive, being more than 90% higher. The superiority of ARACO in this scenario was obtained thanks to the acceleration of the adaptive domain adjustment parameters Δ_1, Δ_2, and Δ_3, at opportune moments, since the tests showed that there was no stagnation of the algorithm. With the acceleration of these parameters, it is possible to reach a small domain faster, which guarantees not only finding the optimal solution, but also a greater precision compared to RACO.

Figure 3 shows a comparison between the two algorithms that have the best results, in relation to MNFE, for the benchmark functions in Table 2.

Table 2 Comparison of results of ARACO, RACO, ACO$_R$, and probability-learning methods that model and sample probability distributions

Function	ARACO	RACO	ACO$_R$	(1+1) ES	CSA-ES	CMA-ES	IDEA	MBOA
Sphere $\vec{x}:[-3,7]^n$, n = 10	1.0 (86)	2.23	17.52	15.93	25.48	20.70	79.65	764.65
Ellipsoid $\vec{x}:[-3,7]^n$, n = 10	1.0 (103)	2.11	112.33	2851.45	4752.42	43.20	69.12	604.85
Cigar $\vec{x}:[-3,7]^n$, n = 10	1.0 (114.5)	2.05	46.95	20457.64	26829.69	33.53	154.27	402.44
Tablet $\vec{x}:[-3,7]^n$, n = 10	1.0 (89)	2.28	28.84	1326.76	1874.77	49.03	83.64	692.22
Rosenbrock $\vec{x}:[-5,5]^n$, n = 10	*1.0 (997)	1.25	*7.93	*367.79	1298.09	7.21	*1514.44	*7932.79

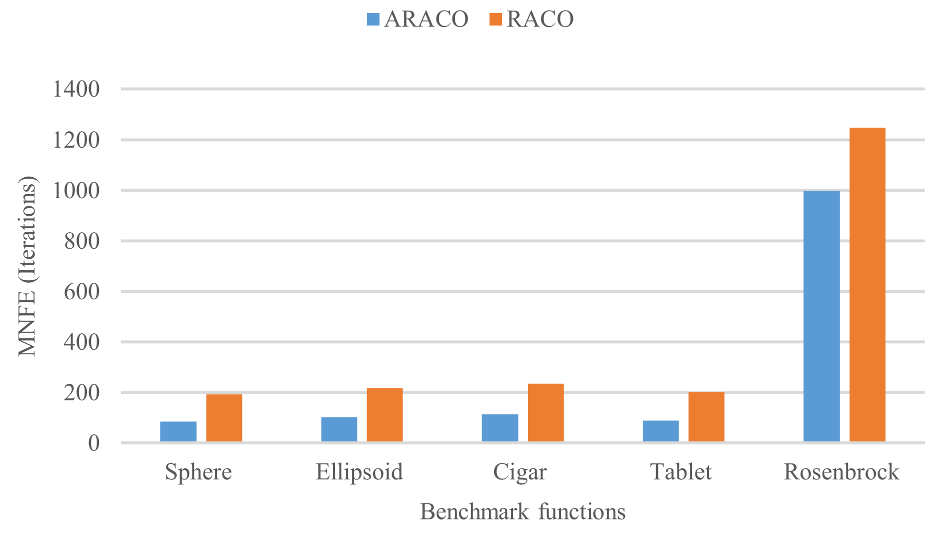

Fig. 3 Comparison of results of ARACO and RACO

Metaheuristics developed for combinatorial optimization and adapted to continuous domains

The metaheuristics used for comparison in this scenario is CGA (Continuous Genetic Algorithm), ECTS (Enhanced Continuous Tabu Search), ESA (Enhanced Simulated Annealing), and DE (Differential Evolution). The parameters for the aforementioned algorithms are essentially chosen through a trial-and-error procedure [19]. The results used for comparison of ACO_R, CGA, ECTS, ESA, and DE were obtained from Dorigo et al. [19], whereas the results used to compare RACO were obtained from Chen et al. [24].

The benchmark functions used in this scenario are shown in Table 3 below, except for the Rosenbrock function, previously shown in Table 1. The Function column shows the name of the benchmark function. The Formula column shows the formula used to calculate the value of the function that will be minimized or maximized. The Optimal column shows the ideal value for each variable of the function. The Minimum column shows the minimum value of the function, when the optimal values for each of the variables are found. The ARACO algorithm was run 100 times for each benchmark function and the comparison criteria used in this section are the average number of function evaluations (ANFE) executed until the termination condition was reached, in addition to the success rate. The Eq. (16) determines the termination condition.

$$|f - f*| < \in_1 * f * + \in_2 \tag{16}$$

where f is the best heuristic value found by ARACO, f^* is the optimal value found in the literature for the benchmark function, and $\in_1 = \in_2 = 10^{-4}$.

Table 4 shows the average number of function evaluations (ANFE) necessary for each method to find the termination condition. The average is shown in parentheses only for the algorithm that has the best result for a function; the other averages can be calculated proportionally based on the value shown in the table. When there is no value shown on the table for a determined algorithm, it means that the result for that benchmark function was not available. In relation to ANFE, ARACO obtains better results in

Table 3 Second part of benchmark functions

Function	Formula	Optimal x*	Minimum f(x*)
Branin RCOS	$f(\overrightarrow{x}) = (x_2 - \frac{5x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10$	3 optimums	$f_{min} = 0.397887$
B2	$f(\overrightarrow{x}) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$	$\overrightarrow{x^*} = (0, 0)$	$f_{min} = 0$
Easom	$f(\overrightarrow{x}) = -\cos(x_1)\cos(x_2)\exp(-((x_1-\pi)^2 + (x_2-\pi)^2))$	$\overrightarrow{x^*} = (\pi, \pi)$	$f_{min} = -1$
Goldstein and Price	$f(\overrightarrow{x}) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2)][30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$\overrightarrow{x^*} = (0, -1)$	$f_{min} = 3$
Zakharov	$f(\overrightarrow{x}) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4$	$\overrightarrow{x^*} = (0, \ldots, 0)$	$f_{min} = 0$
De Jong	$f(\overrightarrow{x}) = x_1^2 + x_2^2 + x_3^2$	$\overrightarrow{x^*} = (0, 0, 0)$	$f_{min} = 0$
Hartmann ($H_{3,4}$)	$f(\overrightarrow{x}) = -\sum_{i=1}^{4} c_i \exp(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2)$ $a_{ij} = \begin{vmatrix} 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \\ 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \end{vmatrix}$ $c_i = \begin{vmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{vmatrix}$ $p_{ij} = \begin{vmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0382 & 0.5743 & 0.8828 \end{vmatrix}$	$x_1^* = 0.114$ $x_2^* = 0.555$ $x_3^* = 0.855$	$f_{min} = -3.8628$
Shekel ($S_{4,k}$, k = 5, 7, 10)	$f(\overrightarrow{x}) = -\sum_{i=1}^{k} (\sum_{j=1}^{4} (x_j - a_{ji})^2 + c_i)^{-1}$	$x_1^* = 4$ $x_2^* = 4$ $x_3^* = 4$ $x_4^* = 4$	$f_{min}^{k=5} = -10.1532$ $f_{min}^{k=7} = -10.4029$ $f_{min}^{k=10} = -10.5364$

Table 3 Second part of benchmark functions *(Continued)*

Function	Formula	Optimal x*	Minimum f(x*)
	$a_{ij} = \begin{vmatrix} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 3.0 & 7.0 & 3.0 & 7.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 3.0 & 5.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 3.6 & 7.0 & 3.6 \end{vmatrix}$ $c_i = \begin{vmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{vmatrix}$		
Hartmann ($H_{6,4}$)	$f(\overrightarrow{x}) = -\sum_{i=1}^{4} c_i \exp(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2)$ $a_{ij} = \begin{vmatrix} 10.0 & 3.00 & 17.0 & 3.50 & 1.70 & 8.00 \\ 0.05 & 10.0 & 17.0 & 0.10 & 8.00 & 14.0 \\ 3.00 & 3.50 & 1.70 & 10.0 & 17.0 & 8.00 \\ 17.0 & 8.00 & 0.05 & 10.0 & 0.10 & 14.0 \end{vmatrix}$ $c_i = \begin{vmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{vmatrix}$	$x_1{}^* = 0.201$ $x_2{}^* = 0.150$ $x_3{}^* = 0.477$ $x_4{}^* = 0.275$ $x_5{}^* = 0.311$ $x_6{}^* = 0.657$	$f_{min} = -3.3223$

Table 3 Second part of benchmark functions *(Continued)*

Function	Formula	Optimal x*	Minimum f(x*)
	$p_{ij} = \begin{vmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{vmatrix}$		
Griewangk	$f(\overrightarrow{x}) = \sum_{i=1}^{n} \frac{{x_i}^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$\overrightarrow{x^*} = (0, \ldots, 0)$	$f_{min} = 0$
Martin & Gaddy	$f(\overrightarrow{x}) = (x_1 - x_2)^2 + (\frac{x_1 + x_2 - 10}{3})^2$	$\overrightarrow{x^*} = (5, 5)$	$f_{min} = 0$

Table 4 Comparison of results of ARACO, RACO, ACO_R, and metaheuristics developed for combinatorial optimization and adapted to continuous domains

Function	ARACO	RACO	ACO_R	CGA	ECTS	ESA	DE
Branin RCOS $\overrightarrow{x}$:[−5,15]^n, $n = 2$	1.0 (17.8)	4.48	48.17	34.41	13.76	-	-
B_2 $\overrightarrow{x}$:[−100,100]^n, $n = 2$	1.0 (19.8)	4.09	28.23	21.71	-	-	-
Easom $\overrightarrow{x}$:[−100,100]^n, $n = 2$	1.0 [96%] (49.7)	1.11 [70%]	15.53 [98%]	29.51	-	-	-
Goldstein and Price $\overrightarrow{x}$:[−2,2]^n, $n = 2$	1.0 (17)	2.88	23.1	24.45	13.58	46.2	-
Rosenbrock (R_2) $\overrightarrow{x}$:[−5,10]^n, $n = 2$	1.0 (74.8)	1.05	10.9	12.83	6.41	10.9	8.34
Zakharov (Z_2) $\overrightarrow{x}$:[−5,10]^n, $n = 2$	1.0 (15.5)	1.29	18.87	40.25	12.58	1019	-
De Jong $\overrightarrow{x}$:[−5.12,5.12]^n, $n = 3$	1.0 (11)	3.81	35.63	67.70	-	-	35.63
Hartmann ($H_{3,4}$) $\overrightarrow{x}$:[0,1]^n, $n = 3$	1.0 (10.8)	2.48	31.66	53.83	50.66	63.33	-
Shekel ($S_{4,5}$) $\overrightarrow{x}$:[0,10]^n, $n = 4$	6.06 [87%]	1.0 [56%] (47.9)	16.55 [57%]	12.73 [76%]	17.82 [75%]	24.19 [54%]	-
Shekel ($S_{4,7}$) $\overrightarrow{x}$:[0,10]^n, $n = 4$	3.32	1.0 [92%] (48.9)	15.29 [79%]	13.9 [83%]	18.07 [80%]	25.03 [54%]	-
Shekel ($S_{4,10}$) $\overrightarrow{x}$:[0,10]^n, $n = 4$	2.97	1.0 [97%] (49.6)	14.41 [81%]	13.1 [83%]	18.34 [80%]	23.58 [50%]	-
Rosenbrock (R_5) $\overrightarrow{x}$:[−5,10]^n, $n = 5$	2.01	1.0 (184.3)	13.94 [97%]	22.08	11.62	29.05	-
Zakharov (Z_5) $\overrightarrow{x}$:[−5,10]^n, $n = 5$	1.0 (36.3)	1.7	20.02	38.05	62.08	1922.64	-
Hartmann ($H_{6,4}$) $\overrightarrow{x}$:[0,1]^n, $n = 6$	3.31 [97%]	1.0 [50%] (28.6)	25.24	32.81	53.01	93.40	-
Griewangk $\overrightarrow{x}$:[−5.12,5.12]^n, $n = 10$	16.78	1.0 [97%] (29.7)	46.8 [61%]	-	-	-	430.57

nine out of the fifteen functions tested. Moreover, in half of the functions, in which ARACO obtains the best results from among all the methods, it is able to optimize the functions with less than half the number of function evaluations of the second best algorithm. If the performance is compared with the other algorithms, the difference becomes even more impressive, where it is shown as being more than 90% better.

Emphasis is placed here upon the fact that, in the six functions that ARACO does not obtain the best results, it is in second place, showing a worse performance only to RACO. It is possible to improve these results by expanding the area around the lowest value found, in all variables, instead of expanding only in the most stagnant variable. However, tests showed that this action results in a lower number of function evaluations, but causes a considerable decrease in the values of another important parameter shown in Table 4—the success rate. The success rate is important as not all algorithms find the optimal solution in every execution. However, through the success rate, it is possible to measure the percentage of executions of the algorithm that find the optimal solution for each function. The percentage shown in square brackets in the table shows the success rate of the algorithms. When there is no percentage in square brackets, it means that all executions were performed with success. In all the proposed functions, ARACO is able to find the optimal solution with a success rate equal to or higher than its main competitor, RACO. For example, the RACO success rate for the Hartmann function ($H_{6,4}$) is 50% and for the Shekel function ($S_{4,5}$) it is 56%, whereas for ARACO these values are 97% and 87%, respectively. This superiority in terms of success rate can be explained by the strategy used to avoid stagnating the algorithm, through the expansion of the domain around the best solution found. In this way, there is a greater possibility of a continuous improvement of the values found, instead of the algorithm being stuck in a region of local minimum, making the success rate of ARACO equal to or near 100%, for all fifteen functions.

In this way, it can be said that the performance of ARACO is superior to the other algorithms, as it manages to find the optimal value, using a lesser amount of function evaluations for this. In addition to achieving a higher success rate, that is, it finds the optimal solution in a higher percentage of executions, when compared to the other algorithms.

Figure 4 shows a comparison between the two algorithms that have the best results, in relation to ANFE, for the first 5 benchmark functions in Table 4.

Methods inspired on the behavior of ants

The methods used for comparison in this scenario are CACO (Continuous ACO), API (after Pachycondyla APIcalis), and CIAC (Continuous Interacting Ant Colony). The parameters used for the algorithms are the following: for CACO, the ant size m = 10, number of regions r = 200, mutation probability p_1 = 0.5, fashion crossover probability p_2 = 1; for API, the ant size m = 20, number of explorations for each ant t = 50, failed search times P_{lacal} = 50; for CIAC, the ant size m = 100, ranges distribution ratio σ = 0.5, persistence of pheromonal spots ρ = 0.1, initial messages number μ = 10; for ACO_R, the ant size m = 2, speed of convergence n = 0.85, locality of the search process q = 10^{-1} and archieve size k = 50 [19]. The RACO parameters are the same as ARACO.

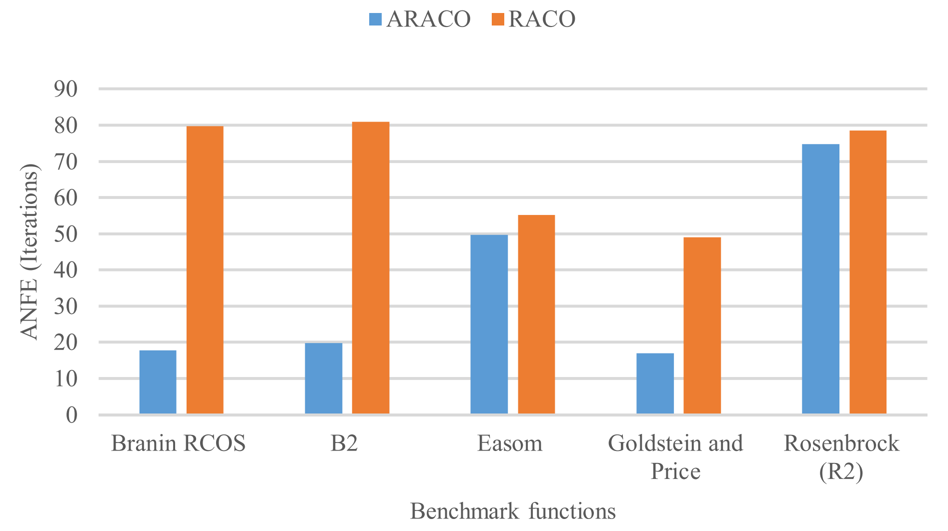

Fig. 4 Comparison of results of ARACO and RACO

The results used to compare ACO_R, CACO, API, and CIAC were obtained from Dorigo et al. [19], whereas the results used for comparison with RACO were obtained from Chen et al. [24].

The benchmark functions used in this scenario are found on the previously presented Tables 1 and 4. The ARACO algorithm was run 100 times for each benchmark function and the comparison criterion used in this section is the average number of function evaluations (ANFE) executed until the termination condition was reached, in addition to the success rate. The Eq. (17) determines the termination condition.

$$|f - f*| < \in_1 * f* + \in_2 \tag{17}$$

where f is the best heuristic value found by ARACO, f^* is the optimal value found in the literature for the benchmark function, and $\in_1 = \in_2 = 10^{-4}$.

Table 5 shows the average number of function evaluations (ANFE) necessary for each method to find the termination condition. As in the previous section, the average is shown in parentheses only for the algorithm that has the best result for a function; the other averages can be calculated proportionally based on the value shown in the table. The percentage shown in square brackets in the table shows the success rate of the algorithms. When there is no percentage in square brackets, it means that all executions were performed with success. One notes that in relation to ANFE, ARACO obtains better or equal results in five of the eight benchmark functions tested. This algorithm comes in second place in the other three functions. However, in these cases, as well as in all evaluated functions, the success rate achieved by ARACO is equal to or higher than all other algorithms, not reaching 100% only in the Shekel function ($S_{4,5}$).

The conclusion is therefore reached that ARACO presents a superior performance over the other algorithms in relation to the average number of function evaluations necessary to reach the termination condition, and when it does not, it is able to obtain a higher success rate. This is due to algorithm that attempts to minimize the number of function evaluations needed to reach the optimal value, with the highest possible success rate.

Table 5 Comparison of results of ARACO, RACO, ACO_R, and methods inspired on the behavior of ants

Function	**ARACO**	**RACO**	**ACO_R**	**CACO**	**API**	**CIAC**
Rosenbrock (R_2) $\vec{x}$:$[-5,10]^n$, $n = 2$	1.0 (74.8)	1.05	10.96	90.98	131.55	153.47
Sphere $\vec{x}$:$[-5.12,5.12]^n$, $n = 6$	1.0 (16.4)	2.86	47.62	1333.41	619.08	3047.80
Griewangk $\vec{x}$:$[-5.12,5,12]^n$, $n = 10$	16.61	1.0 (30) [97%]	46.33 [61%]	1668	-	1668 [52%]
Goldstein and Price $\vec{x}$:$[-2,2]^n$, $n = 2$	1.0 (17)	2.88	22.58	316.23	-	1377.88 [56%]
Martin and Gaddy $\vec{x}$:$[-20,20]^n$, $n = 2$	1.0 (16)	1.0	21.56	107.81	-	733.12 [20%]
B_2 $\vec{x}$:$[-100,100]^n$, $n = 2$	1.0 (19.8)	4.07	27.37	-	-	602.31
Rosenbrock (R_5) $\vec{x}$:$[-5,10]^n$, $n = 5$	2.01	1.0 (184.3)	13.49 [97%]	-	-	215.90 [90%]
Shekel ($S_{4,5}$) $\vec{x}$:$[0,10]^n$, $n = 4$	6.06 [87%]	1.0 (47.9) [56%]	16.43 [57%]	-	-	821.5 [5%]

Figure 5 shows a comparison between the two algorithms that have the best results, in relation to ANFE, for some of the benchmark functions in Table 5.

Tests where the initial domain does not contain the optimal solution

Among all the algorithms cited in the previous section, only RACO has the capacity to find the ideal solution, given an initial domain that does not contain this solution. Therefore, all tests performed in this scenario compare only ARACO and RACO, the only broad-range search algorithms.

For the comparison between the two algorithms, 14 functions are used. Table 6 shows 8 of these functions, whereas the others were presented previously in this article. The Function column shows the name of the benchmark function. The Formula column shows the formula used to calculate the value of the function that will be minimized or maximized. The Optimal column shows the ideal value for each variable of the function. The Minimum column shows the minimum value of the function, when the optimal values for each of the variables are found. The ARACO algorithm was run 20 times for each benchmark function in each domain provided and the comparison criteria used in this section are the average number of function evaluations (ANFE) performed until the termination condition was reached, in addition to the success rate. The Eq. (18) determines the termination condition.

$$\max\left(h_1, h_2, \dots, h_n\right) < \in \tag{18}$$

where h_i $(i = 1, 2, \dots, n)$ is the value of each division of the domain grid, given by Eq. 1 and $\in$ is 10^{-5}.

For the execution of the tests, as performed by Chen et al. [24] in the implementation of the RACO algorithm, five scenarios are used, which are equivalent to five domains that do not have the ideal solution. These are:

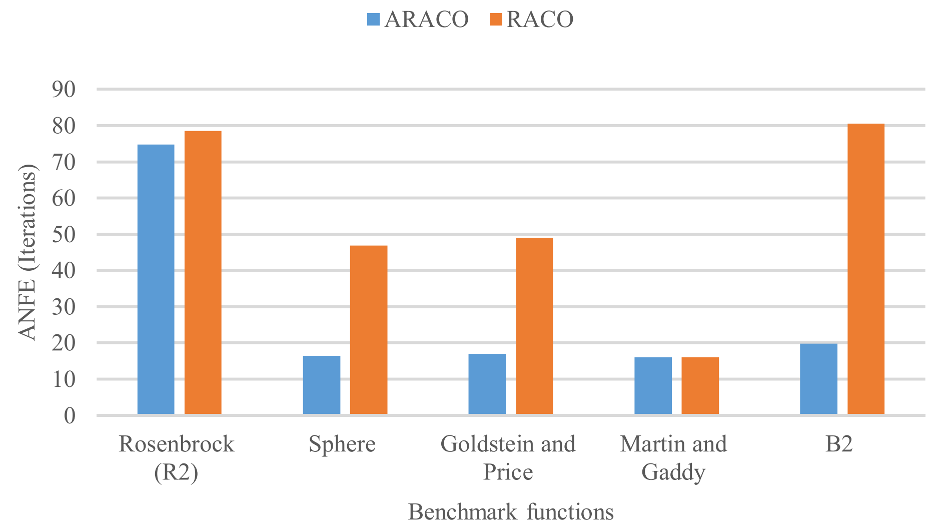

Fig. 5 Comparison of results of ARACO and RACO

Table 6 Third part of benchmark functions

Function	Formula	Optimal x*	Minimum f(x*)
Rastrigin	$f(\overrightarrow{x}) = ({x_1}^2 + {x_2}^2 - \cos(18x_1) - \cos 18x_2$	$\overrightarrow{x^*} = (0, 0)$	$f_{min} = 0.397887$
Shubert	$f(\overrightarrow{x}) = (\sum_{i=1}^{5} i\cos(i + (i+1)x_1))(\sum_{i=1}^{5} i\cos(i + (i+1)x_2))$	18 optima	$f_{min} = -186.7309$
Ackley	$f(\overrightarrow{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}{x_i}^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$	$\overrightarrow{x^*} = (0, \ldots, 0)$	$f_{min} = 0$
Levy	$f(\overrightarrow{x}) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1}[(y_i - 1)^2(1 + 10\sin^2(\pi y_1 + 1))] + (y_n - 1)^2(1 + 10\sin^2 2\pi y_n))$ $y_i = 1 + \frac{1}{4}(x_i - 1)$	$\overrightarrow{x^*} = (1, \ldots, 1)$	$f_{min} = 0$
Beale	$f(\overrightarrow{x}) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1{x_2}^2)^2 + (2.625 - x_1 + x_1{x_2}^3)^2$	$\overrightarrow{x^*} = (3, 0.5)$	$f_{min} = 0$
Six-Hump Camel-Back	$f(\overrightarrow{x}) = 4{x_1}^2 - 2.1{x_1}^4 + {x_1}^6 \Big/ 3 + x_1x_2 - 4{x_2}^2 + 4{x_2}^4$	$\overrightarrow{x^*} = (0.08983, -0.7126)$, $(-0.08983, 0.7126$	$f_{min} = 0$
Bohachevsky	$f(\overrightarrow{x}) = {x_1}^2 + {x_2}^2 - 0.3\cos(3\pi x_1) + 0.3\cos(4\pi x_2) + 0.3$	$\overrightarrow{x^*} = (0, 0.24), (0, -0.24)$	$f_{min} = -0.24$
Hansen	$f(\overrightarrow{x}) = (\cos(1) + 2\cos(x_1 + 2) + 3\cos(2x_1 + 3) + 4\cos(3x_1 + 4) + 5\cos(4x_1 + 5))(\cos 2x_2 + 1) + 2\cos(3x_2 + 2) + 3\cos(4x_2 + 3) + 4\cos 5x_2 + 4) + 5\cos(6x_2 + 5))$	$\overrightarrow{x^*} = (-1.30671, 4.85806)$	$f_{min} = -176.5418$

- Two positives and larges domains $x_{1min} = 100$, $x_{1max} = 200$, $x_{2min} = 50$ and $x_{2max} = 80$
- Two negatives and larges domains $x_{1min} = -300$, $x_{1max} = -180$, $x_{2min} = -600$ and $x_{2max} = -50$
- One positive and one negative domain, far from the ideal solution $x_{1min} = 1800$, $x_{1max} = 1900$, $x_{2min} = -230$ and $x_{2max} = -110$
- One positive and one negative domain, extremely narrow $x_{1min} = 1$, $x_{1max} = 2$, $x_{2min} = -3$ and $x_{2max} = -1$
- One small and positive domain, and one large and negative domain $x_{1min} = 100$, $x_{1max} = 110$, $x_{2min} = -300$, and $x_{2max} = -190$

Table 7 shows the comparison between the results found by ARACO and RACO, regarding ANFE and the success rate. It is important to note that, in some scenarios, RACO cannot find the optimal solution for some functions. These cases are marked with a – in the table, symbolizing that RACO was unable to find the optimal solution in any of the twenty executions. This occurs in one proposed domain for the Shubert function, three proposed domains for the Ackley function, two proposed domains for the Levy function, and one proposed domain for the Six-Hump Camel-Back function. This situation does not occur with ARACO and is one of the great advantages observed in the algorithm, as it is also able to find the optimal solution in all scenarios for all the benchmark functions tested.

If the comparison is performed according to ANFE, in the first scenario, ARACO is superior to RACO in 12 of the 14 functions proposed, reaching a result 82% lower than that provided by RACO, in the Six-Hump Camel-Back function. For the Griewangk function, in which RACO has superior performance, it is important to highlight that the success rate of RACO is 70%, whereas ARACO has a 100% success rate, which demonstrates the advantage of ARACO in finding the optimal solution in all executions, a fact that as previously stated, represents a great advantage of the algorithm.

Figure 6 shows a comparison between the RACO and ARACO algorithms in the first proposed scenario, in relation to ANFE, in some of the benchmark functions in table 7.

The second proposed scenario presents similar results in relation to ANFE, with ARACO surpassing RACO in 12 of the 14 proposed functions, reaching a result 79% lower than that provided by RACO, in the Bohachevsky function. For the Griewangk function, where RACO shows superior performance, the RACO's success rate is 80%, whereas ARACO has a 100% success rate. For the Ackley function, RACO cannot find the optimal solution in any run, whereas ARACO finds the solution in all runs.

In the third scenario, ARACO also is superior to RACO in 12 of the 14 proposed functions in relation to ANFE, reaching a result 75% lower than that provided by RACO, in the Six-Hump Camel-Back function. Noteworthy here is that, RACO achieves only 40% success rate in the Griewangk function, whereas ARACO has superior performance in relation to ANFE, in addition to reaching a 100% success rate. RACO cannot find the optimal solution to the Ackley function, whereas ARACO can find 85% of the executions.

The fourth scenario is where the difference becomes most significant, as ARACO obtains superior performance in all 14 proposed functions in relation to ANFE, reaching a result 79% lower than that provided by RACO, in the Bohachevesky function. In

Table 7 Comparison of results of ARACO and RACO when the initial domain does not contain the optimal solution

	x_1 = (100, 200) x_2 = (50, 80)		x_1 = (−300, 180) x_2 = (−600, −50)		x_1 = (1800, 1900) x_2 = (−230, 110)		x_1 = (1, 2) x_2 = (−3, −1)		x_1 = (100, 110) x_2 = (−300, −190)		Functions
	ARACO	RACO	ARACO	RACO	ARACO	RACO	ARACO	RACO	ARACO	RACO	
ANFE	380	383	161	370	2057	1559	35	111	415	292	Goldstein and Price
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	54	155	53	171	72	187	38	108	57	163	Zakharov
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	59	155	64	169	72	184	44	125	60	164	Martin and Gaddy
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	239	188	368	220	214	217	37	108	300	249	Griewangk
Success Rate	100%	70%	100%	80%	100%	40%	100%	100%	100%	25%	
ANFE	94	156	79	189	79	190	63	111	82	159	Rastrigin
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	123	151	133	282	124	284	54	-	96	266	Shubert
Success Rate	100%	100%	100%	100%	100%	100%	100%	0%	100%	100%	
ANFE	121	171	119	-	392	-	59	108	166	-	Ackley
Success Rate	100%	100%	100%	0%	85%	0%	100%	100%	100%	0%	
ANFE	48	157	52	170	61	182	37	108	52	156	B2
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	52	157	52	170	385	261	37	-	196	-	Levy
Success Rate	100%	100%	100%	100%	100%	100%	100%	0%	100%	0%	
ANFE	68	178	67	178	104	190	46	118	77	167	Beale
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	352	253	257	242	794	802	124	163	709	423	Rosenbrock
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	

Table 7 Comparison of results of ARACO and RACO when the initial domain does not contain the optimal solution *(Continued)*

	x_1 = (100, 200) x_2 = (50, 80)		**x_1 = (−300, 180) x_2 = (−600, −50)**		**x_1 = (1800, 1900) x_2 = (−230, 110)**		**x_1 = (1, 2) x_2 = (−3, −1)**		**x_1 = (100, 110) x_2 = (−300, −190)**		**Functions**
	ARACO	**RACO**	**ARACO**	**RACO**	**ARACO**	**RACO**	**ARACO**	**RACO**	**ARACO**	**RACO**	
ANFE	39	153	35	169	58	182	25	119	34	168	Bohachevesky
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	
ANFE	183	205	225	261	197	260	57	113	158	188	Hansen
Success Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	95%	
ANFE	30	166	37	174	43	177	45	-	36	176	Six-Hump Camel-Back
Success Rate	100%	100%	100%	100%	100%	100%	100%	0%	100%	100%	

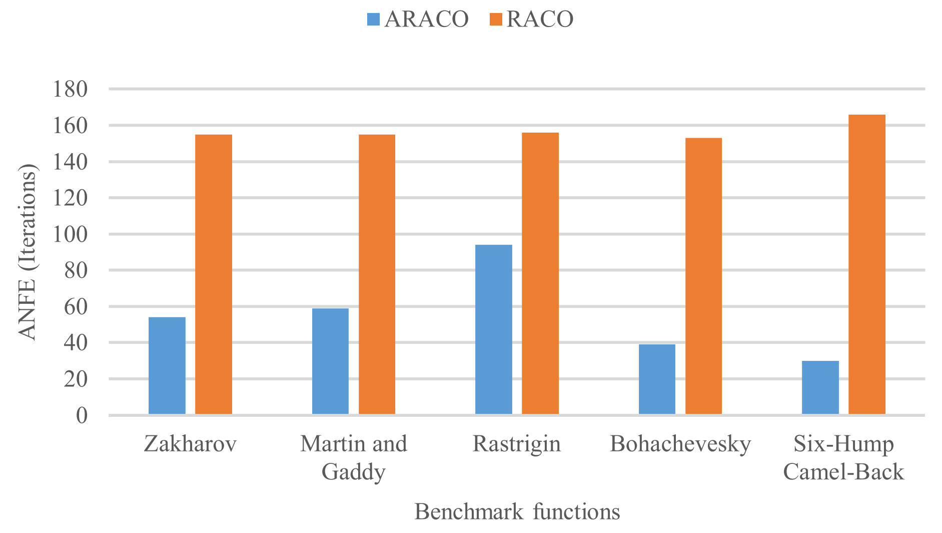

Fig. 6 Comparison of results of ARACO and RACO

addition, RACO cannot find the optimal solution in none of the executions of the Six-Hump Camel-Back, Shubert and Levy functions, whereas ARACO achieves a 100% success rate in all the proposed functions.

Finally, in the fifth scenario, ARACO surpasses RACO in 11 of the 14 functions proposed in relation to ANFE, reaching a result 81% lower than that provided by RACO, in the Six-Hump Camel-Back function. For the Griewangk function, where RACO has superior performance, emphasis is placed on the fact that RACO's success rate is 25%, whereas ARACO has a 100% success rate. In the Ackley and Levy functions, RACO cannot find the optimal solution in any of the executions, whereas ARACO achieves a 100% success rate.

All these comparisons show that ARACO has superior performance, that is, a lower ANFE's value, in 87% of the tests performed in scenarios where the initial domain provided does not contain the ideal solution. In some of the tests where ARACO does not surpass RACO, RACO cannot find the optimal solution in all runs, whereas ARACO can find the optimal solution in more runs within these scenarios. In addition, RACO cannot find the solution in none of the executions in seven proposed scenarios, whereas ARACO can find the optimal solution in all of these scenarios. All of this points to the superiority of ARACO, both in terms of finding the optimal solution in a lower average number of function evaluations (ANFE), as well as to achieve a higher success rate, and finding the optimal solution in all the proposed scenarios. Improved values for ANFE are obtained thanks to the acceleration of the adaptive domain adjustment parameters in opportune moments, allowing for a domain, where the optimal solution is present, to be found more quickly, additionally, when found, the domain adjustment process is accelerated so that the value can be found faster. Whereas, the highest success rate and the possibility of finding the optimal solution in all scenarios, are advantages obtained thanks to the strategy of expansion of the domain around the best solution found. Thus, when the algorithm enters a state of stagnation, it allows for the generation of a new domain outside the local minimum region.

Tests with the CEC 2019 benckmark functions

The CEC 2019 benchmark test functions [31] are a group of functions that are difficult to optimize, known as "The 100-digit challenge" and which were used in an annual optimization competition in 2019. The functions are presented in Table 8. The Function column shows the name of the benchmark function. The Formula column shows the formula used to calculate the value of the function that will be minimized or maximized. The Minimum column shows the minimum value of the function, when the optimal values for each of the variables are found. The metaheuristics used for comparison in this scenario are PSO-based algorithms, those being the Fitness Dependent Optimizer (FDO) [32], the Dragonfly Algorithm (DA) [33], the Whale Optimization Algorithm (WOA) [34], and Salp Swarm Algorithm (SSA) [35].

In order to facilitate the evaluation, all implementations of the CEC 2019 100-Digit Challenge benchmark functions, used in the competition in 2019 and also in ARACO and the other algorithms, were adapted so that the optimal values sought were 1. All the CEC 2019 100-Digit Challenge benchmark functions defined have 10 dimensions and the domain provided to the algorithms is $x^i_{min} = -100$ and $x^i_{max} = 100$, for each dimension i, except for the functions Storn's Chebyshev Polynomial Fitting Problem, Inverse Hilbert Matrix Problem, and Lennard-Jones Minimum Energy Cluster, which have different dimensions and different initial domains, as shown in Table 9.

The results used to compare FDO, DA, WOA, and SSA were obtained from Abdullah and Ahmed [32]. All algorithms were executed 30 times for each benchmark function and the comparison criterion used in this section is the average of the minimum value found by the algorithms, after 500 iterations have been run.

Table 9 shows the comparison among the averages of the minimum values found by the algorithms after 30 executions, with 500 iterations each execution. Noteworthy here is that the results obtained by ARACO are superior to those encountered by the other algorithms in six out of the ten benchmark functions tested. The algorithm is in second place in three other functions, and it is in third place only in the Weierstrass function.

One notes, the results found by ARACO are competitive and, still further, it is the only algorithm that has results that approach the optimal value with only 500 iterations covered, managing to find the whole number of the optimal value in 4 of the functions tested.

Figure 7 shows a comparison between the results encountered by the five algorithms covered in this scenario, in relation to the average of the minimum value found for some of the benchmark functions in Table 9, after the execution of 500 iterations.

However, it is important to highlight that ARACO achieves even better results for the proposed benchmark functions, after the 500 iterations set. However, in order for a fair comparison to be made, the same criteria and termination condition defined by the other algorithms were maintained. In spite of that, to guide future comparisons in future studies, the results achieved by ARACO, after the execution of 5000 iterations, are presented below. One notes that with this termination condition, the algorithm achieves results superior to those achieved with 500 iterations. This proves that, as iterations pass, ants are able find even better solutions. Table 10 shows the average for the minimum value achieved by ARACO after 30 executions, with 5000 iterations each execution.

Table 8 CEC 2019 100-Digit Challenge benchmark functions

Function	Formula	Minimum f(x*)
Storn's Chebyshev Polynomial Fitting Problem	$f(\overrightarrow{x}) = p_1 + p_2 + p_3$ $p_1 = \begin{cases} (u-d)^2 & , \mathit{if}\ u < d \\ 0, & , \mathit{otherwise} \end{cases}$ $u = \sum_{j=1}^{D} x_j (1.2)^{D-j}$ $p_2 = \begin{cases} (v-d)^2 & , \mathit{if}\ v < d \\ 0, & , \mathit{otherwise} \end{cases}$ $v = \sum_{j=1}^{D} x_j (-1.2)^{D-j}$ $pk = \begin{cases} (w_k-1)^2 & , \mathit{if}\ w_k > 1 \\ (w_k+1)^2 & , \mathit{if}\ w_k < 1 \\ 0, & , \mathit{otherwise} \end{cases}$ $w_k = \sum_{j=1}^{D} x_j (\frac{2k}{m}-1)^{D-j}$ $p_3 = \sum_{m=0}^{m} p_k,\ \ k = 0, 1, \ldots, m,\ \ m = 32D$ $d = 72.661\ \mathit{for}\ D = 9$	$f_{min} = 1$
Inverse Hilbert Matrix Problem	$f(\overrightarrow{x}) = \sum_{i=1}^{n} \sum_{k=1}^{n} \lvert w_{i,k} \rvert$ $(w_{i,k}) = W = HZ-I,\ \ I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$ $H = (h_{i,k}),\ \ h_{i,k} = \frac{1}{i+k-1},\ \ i, k = 1, 2, \ldots, n,\ \ n = \sqrt{D}$ $Z = (z_{i,\ k}),\ \ z_{i,\ k} = x_{i+n(k-1)}$	$f_{min} = -186.7309$
Lennard-Jones Minimum Energy Cluster	$f(\overrightarrow{x}) = 12.7120622568 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\frac{1}{d_{i,j}^2} - \frac{2}{d_{i,j}})$ 3 $d_{i,j} = (\sum_{k=0}^{2} (x_{3i+k-2} - x_{3j+k-2})^2) \quad , n = \frac{D}{3}$	$f_{min} = 1$
Rastrigin's Function		$f_{min} = 1$

Table 8 CEC 2019 100-Digit Challenge benchmark functions *(Continued)*

Function	Formula	Minimum f(x*)
	$f(\vec{x}) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$	
Griewangk's Function	$f(\vec{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1$	$f_{min} = 1$
Weierstrass Function	$f(\vec{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k_{max}}[a^k \cos(2\pi b^k(x_i + 0.5))]) - D\sum_{k=0}^{k_{max}} a^k \cos(\pi b^k)$ $a = 0.5,\ b = 3,\ k_{max} = 20$	$f_{min} = 1$
Modified Schwefel's Function	$f(\vec{x}) = 418.9829D - \sum_{i=1}^{D} g(z_i)$ $z_i = x_i + 420.9687462275036$ $g(z_i) = \begin{cases} z_i \sin(\lvert z_i\rvert^{\frac{1}{2}}) & if\ \lvert z_i \rvert \le 500 \\ (500 - mod(z_i,500))\sin(\sqrt{\lvert 500 - mod(z_i,500)\rvert}) - \frac{(z_i-500)^2}{10000D} & if\ z_i > 500 \\ (\ mod(z_i,500) - 500)\sin(\sqrt{\lvert mod(z_i,500) - 500\rvert}) - \frac{(z_i+500)^2}{10000D} & if\ z_i < -500 \end{cases}$	$f_{min} = 1$
Expanded Schaffer's F6 Function	$g(x,y) = 0.5 + \frac{\sin^2(\sqrt{x^2+y^2}) - 0.5}{(1+0.001(x^2+y^2))^2}$ $f(\vec{x}) = g(x_1,x_2) + g(x_2,x_3)\ldots + g(x_{D-1},x_D) + g(x_D,x_1)$	$f_{min} = 1$
Happy Cat Function	$f(\vec{x}) = \lvert\sum_{i=1}^{D} x_i^2 - D\rvert^{1/4} + (0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i)/D + 0.5$	$f_{min} = 1$
Ackley Function	$f(\vec{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$	$f_{min} = 1$

Table 9 Comparison of results of ARACO and other metaheuristics in tests with the CEC 2019 100-Digit Challenge benckmark functions

Function	**ARACO**	**FDO**	**DA**	**WOA**	**SSA**
Storn's Chebyshev Polynomial Fitting Problem $\overrightarrow{x}$:[−8192, 8192]^n, $n = 9$	5.60E9	4585.27	5.43E10	4.11E10	6.05E9
Inverse Hilbert Matrix Problem $\overrightarrow{x}$:[−16384, 16384]^n, $n = 16$	1.9243	4	78.0368	17.3495	18.3434
Lennard-Jones Minimum Energy Cluster $\overrightarrow{x}$:[−4, 4]^n, $n = 18$	12.6736	13.7024	13.7026	13.7024	13.7025
Rastrigin's Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	5.6495	34.0837	344.3561	394.6754	41.6936
Griewangk's Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	1.0635	2.1392	2.5572	2.7342	2.2084
Weierstrass Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	10.5288	12.1332	9.8955	10.7085	6.0798
Modified Schwefel's Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	1.0005	120.4858	578.9531	490.6843	410.3964
Expanded Schaffer's F6 Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	1.8266	6.1021	6.8734	6.909	6.37
Happy Cat Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	3.0513	2	6.0467	5.9371	3.6723
Ackley Function $\overrightarrow{x}$:[−100, 100]^n, $n = 10$	12.9822	2.7182	21.2604	21.2761	21.04

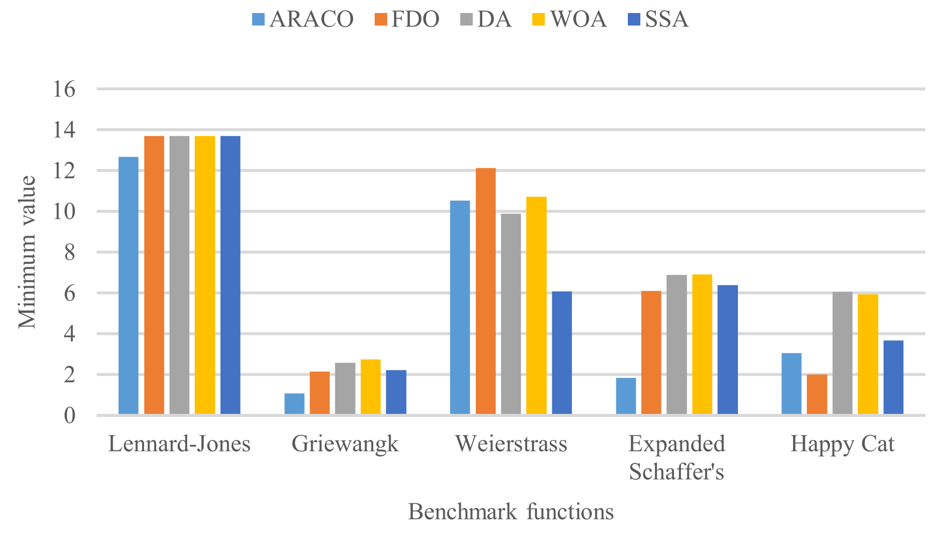

Fig. 7 Comparison of results of ARACO and other metaheuristics

Conclusions

ARACO was proposed with the objective of allowing for the solving of optimization problems in the real world, searching for the optimal solution, providing initial domains with or without the optimal solution. For such, an improvement in the RACO algorithm was proposed, allowing the acceleration of the parameters responsible for the locating and narrowing of the correct domain, in order to accelerate performance. In addition to a strategy that allows the algorithm to leave local minimum regions, permitting ARACO to find the optimal solution in practically all runs.

Tests were executed to prove the proposed strategies in relation to their operation in provided domains that contain the optimal solution. As such, ARACO provides promising results, showing superiority regarding the number of function evaluations necessary to reach the optimal value, in a majority of the tested benchmarks functions, across three proposed comparison groups. These groups were probability-learning methods that model and sample probability distributions, metaheuristics developed for combinatorial optimization and adapted to continuous domains, along with methods inspired by the behavior of ants.

Table 10 Results of ARACO after running 5000 iterations

Function	ARACO
Storn's Chebyshev Polynomial Fitting Problem $\vec{x}$:[−8192, 8192]^n, n = 9	5.23E6
Inverse Hilbert Matrix Problem $\vec{x}$:[−16384, 16384]^n, n = 16	1.3992
Lennard-Jones Minimum Energy Cluster $\vec{x}$:[−4, 4]^n, n = 18	8.3911
Rastrigin's Function $\vec{x}$:[−100, 100]^n, n = 10	3.3252
Griewangk's Function $\vec{x}$:[−100, 100]^n, n = 10	1.0085
Weierstrass Function $\vec{x}$:[−100, 100]^n, n = 10	2.2961
Modified Schwefel's Function $\vec{x}$:[−100, 100]^n, n = 10	1.0001
Expanded Schaffer's F6 Function $\vec{x}$:[−100, 100]^n, n = 10	1.2062
Happy Cat Function $\vec{x}$:[−100, 100]^n, n = 10	2.4945
Ackley Function $\vec{x}$:[−100, 100]^n, n = 10	10.8957

The obtainment of good results, through operating in provided domains that do not contain the optimal solution, is very desirable, as in real-world problems one cannot always be guarantee that the initial domain provided for the algorithm contains the optimal solution. In these cases, ARACO demonstrates a very effective performance, since in addition to being able to find the optimal value in a lower number of function evaluations than its predecessor RACO in 87% of the tested scenarios; it is able to find the optimal value in all the proposed scenarios, achieving a 100% success rate on virtually all tested functions. These advantages are obtained thanks to the acceleration of the adaptive domain adjustment parameters at opportune moments, allowing for a greater speed of convergence of the algorithm was obtained, and thanks to the extension of the domain around the best solution found, when the algorithm is stagnant. Thus, allowing for the generation of a new domain that contains values outside the local minimum region.

On the subject of the tests performed with CEC 2019 100-Digit Challenge benchmark functions, ARACO obtained excellent results, showing a superior performance over the other algorithms in most of the tested benchmark functions, as well as being the only algorithm to achieve the results closest to the optimum values in 500 iterations. In addition, the results can be further improved if more iterations are executed.

As future work, the algorithm can be expanded to allow for working with functions that have more variables, achieving a good performance, especially without compromising the accuracy and the success rate obtained. These results have already been successfully accomplished in some of the functions implemented in the work, such as the functions with 10 variables Sphere, Ellipsoid, Cigar, and Tablet. However, in other functions such as Griewangk (10 variables) and Hartmann (6 variables), the algorithm achieves high success rates, but loses in performance, performing a large number of function evaluations until it finds the optimal value. The implemented CEC 2019 100-Digit Challenge benchmark functions also demonstrate the need to determine better strategies to obtain superior results in functions that have more dimensions. In addition, a study can be performed to determine better strategies for detecting domain stagnation and obtain better domain expansion rates around the best solution found, according to the nature of the functions, to further improve the results obtained.

As one of the proposals of the ARACO algorithm is its applicability in solving real-world problems, another future work should therefore be based on the practical application of the algorithm in real-world situations.

Acknowledgements
Not applicable.

Authors' contributions
All authors read and approved the final manuscript.

Funding
The authors did not receive support from any organization for the submitted work.

Availability of data and materials
All data generated or analyzed during this study are included in this published article.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 12 February 2021 Accepted: 30 August 2021
Published online: 02 December 2021

References

1. Antoniou A, Lu WS (2007) The optimization problem. In: Antoniou A, Lu WS (eds) Practical Optimization. Springer, Boston, pp 1–26. https://doi.org/10.1007/978-0-387-71107-2_1
2. Edmonds J (2008) How to think about algorithms. Cambridge University Press, New York. https://doi.org/10.1017/CBO9780511808241
3. Saka MP, Dogan E, Aydogdu I (2013) Analysis of swarm intelligence-based algorithms for constrained optimization. In: Yang XS, Cui Z, Xiao R, Gandomi AH, Karamanoglu M (eds) Swarm Intelligence and Bio-inspired Compuation. Elsevier, Oxford. https://doi.org/10.1016/B978-0-12-405163-8.00002-8
4. Kaur SP (2013) Variables in research. Indian J Res Rep Med Sci 3(4):36–38
5. Wu Z, Xue R (2019) A cyclical non-linear inertia-weighted teaching-learning-based optimization algorithm. Algorithms 12(5):94. https://doi.org/10.3390/a12050094
6. Serapião ABS (2009) Fundamentos de otimização por inteligência de enxames: uma visão geral. Sba Controle Automação 20(3):271–304. https://doi.org/10.1590/S0103-17592009000300002
7. Goldberg DE (1989) Generic Algorithm in search, optimization and machine learning. Addison-Wesley, Reading, Boston
8. Storn R, Price K (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optimization 11(4):341–359. https://doi.org/10.1023/A:1008202821328
9. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
10. Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano
11. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B (Cybernetics) 26(1):29–41. https://doi.org/10.1109/3477.484436
12. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66. https://doi.org/10.1109/4235.585892
13. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Technical Report – TR06, Erciyes University, Engineering Faculty Computer Engineering Department Kayseri, Turkey
14. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95 – International Conference on Neural Networks 4, Perth, WA, pp 1942–1948. https://doi.org/10.1109/ICNN.1995.488968
15. Sudholt D, Thyssen C (2012) Running time analysis of ant colony optimization for shortest path problems. J Discrete Algorithms 10:165–180. https://doi.org/10.1016/j.jda.2011.06.002
16. Ding QL, Hu XP, Sun LJ, Wang YZ (2012) An improved ant colony optimization and its application to vehicle routing problem with time windows. Neurocomputing 98:101–107. https://doi.org/10.1016/j.neucom.2011.09.040
17. Blum C, Sampels M (2004) An ant colony optimization algorithm for shop scheduling problems. J Math Model Algorithms 3(3):285–304. https://doi.org/10.1023/B:JMMA.0000038614.39977.6f
18. Dorigo M, Stutzle T (2019) Ant colony optimization: overview and recent advances, Handbook of Metaheuristics. Int Ser Oper Res Manage Sci 272:311–351. https://doi.org/10.1007/978-3-319-91086-4_10
19. Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. Eur J Oper Res 185(3):1155–1173. https://doi.org/10.1016/j.ejor.2006.06.046
20. Bilchev G, Parmee I (2006) The ant colony metaphor for searching continuous design Spaces. In: Selected Papers from AISB Workshop on Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg, pp 25–39. https://doi.org/10.1007/3-540-60469-3_22
21. Huang H, Hao Z (2006) ACO for continuous optimization based on discrete encoding. In: Proceedings of the 5th International Conference on Ant Colony Optimization and Swarm Intelligence - ANTS 2006. Springer, Berlin, Heidelberg, pp 504–505. https://doi.org/10.1007/11839088_53
22. Dréo J, Siarry P (2004) Continuous interacting ant colony algorithm based on dense heterarchy. Future Generation Comput Syst 20(5):841–856. https://doi.org/10.1016/j.future.2003.07.015
23. Monmarché N, Venturini G, Slimane M (2000) On how Pachycondyla apicalis ants suggest a new search algorithm. Future Generation Comput Syst 16(8):937–946. https://doi.org/10.1016/S0167-739X(00)00047-9
24. Chen Z, Zhou Z, Luo J (2017) A robust ant colony optimization for continuous functions. Expert Syst Appl Int J 81:309–320. https://doi.org/10.1016/j.eswa.2017.03.036
25. Leguizamón G, Coello CAC (2010) An alternative ACOR algorithm for continuous optimization problems. In: Proceedings of the 7th International Conference on Ant Colony Optimization and Swarm Intelligence - ANTS 2010. Springer-Verlag, Berlin, Heidelberg, pp 48–59. https://doi.org/10.1007/978-3-642-15461-4_5
26. Liao TJ, Montes da Oca MA, Aydin D, Stutlze T, Dorigo M (2011) An incremental ant colony algorithm with local search for continuous optimization. In: Proceedings of the genetic and evolutionary computation conference – GECCO'11. Association for Computing Machinery, New York, pp 125–132. https://doi.org/10.1145/2001576-2001594
27. Liao TJ, Stutzle T, Montes da Oca MA, Dorigo M (2014) A unified ant colony optimization algorithm for continuous optimization. Eur J Oper Res 234(3):597–609. https://doi.org/10.1016/j.ejor.2013.10.024
28. Yang Q, Chen W, Yu Z, Gu T, Li Y, Zhang H, Zhang J (2017) Adaptive Multimodal Continuous Ant Colony Optimization. IEEE Trans Evol Comput 21(2):191–205. https://doi.org/10.1109/TEVC.2016.2591064
29. Liu L, Dai Y (2014) Gao J (2014) Ant colony optimization algorithm for continuous domains based on position distribution model of ant colony foraging. Sci World J 2014:1–9. https://doi.org/10.1155/2014/428539
30. Kern S, Muller SD, Hansen N, Buche D, Ocenasek J, Koumoutsakos P (2004) Learning probability distributions in continuous evolutionary algorithms – A comparative review. Nat Comput 3(1):77–112. https://doi.org/10.1023/B:NACO.0000023416.59689.4e
31. Price KV, Awad NH, Ali MZ, Suganthan PN (2018) The 100-digit challenge: Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization. Technical Report, Nanyang Technological University, Singapore

32. Abdullah JM, Ahmed T (2019) Fitness dependent optimizer: inspired by the bee swarming reproductive process. IEEE Access 7:43473–43486. https://doi.org/10.1109/ACCESS.2019.2907012
33. Mirjalili S (2015) Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective discrete and multi-objective problems. Neural Comput Appl 27(4):1053–1073. https://doi.org/10.1007/s00521-015-1920-1
34. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008
35. Mirjalili S, Gandomi AH, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM (2017) Salp swarm algorithm: a bio-inspired optimizer for engineering design problems. Adv Eng Soft 114:163–191. https://doi.org/10.1016/j.advengsoft.2017.07.002

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com