Nat Comput (2017) 16:339–352
DOI 10.1007/s11047-015-9537-y

A dynamic-edge ACS algorithm for continuous variables problems

Min-Thai Wu[1] · **Tzung-Pei Hong**[1,2] · **Chung-Nan Lee**[1]

Published online: 23 January 2016

Abstract Ant colony systems (ACS) have been successfully applied to solving optimization problems. Especially, they are efficient and effective in finding nearly optimal solutions to discrete search spaces. When the solution spaces of the problems to be solved are continuous, it is not so appropriate to use the original ACS to solve it. This paper thus proposes a dynamic-edge ACS algorithm for solving continuous variables problems. It can dynamically generate edges between two nodes and give a pheromone measures for them in a continuous solution space through distribution functions. In addition, it maps the encoding representation and the operators of the original ACS into continuous spaces easily. The proposed algorithm thus provides a simple and standard approach for applying ACS to a problem that has a continuous solution space, and retains the original process and characteristics of the traditional ACS. Several constrained functions are also evaluated to demonstrate the performance of the proposed algorithm.

✉ Min-Thai Wu
d953040015@student.nsysu.edu.tw

Tzung-Pei Hong
tphong@nuk.edu.tw

Chung-Nan Lee
cnlee@cse.nsysu.edu.tw

[1] Department of Computer Science and Engineering, National Sun Yat-sen University, 70 Lienhai Rd., Kaohsiung 80424, Taiwan

[2] Department of Computer Science and Information Engineering, National University of Kaohisung, 700 Kaohsiung University Rd., Kaohsiung 81148, Taiwan

Keywords Ant colony system · Constrained function · Continuous space · Distribution function · Dynamic edge

Abbreviations

AS	Ant system
ACS	Ant colony system
ACO_R	Ant colony optimization for continuous domains
DEACS	Dynamic-edge ant colony system
CACO	Continuous ant colony optimization
API	*Pachycondyla apicalis*
GA	Genetic algorithm
CIAC	Continuous interacting ant colony

1 Introduction

Ant colony systems (ACS) have been widely applied to find near-optimal solutions for NP-hard problems. An ACS adopts distributed computation and uses a constructive greedy heuristic algorithm (Jiang et al. 2005) with positive feedback to search for solutions. It is a powerful approach inspired by the behavior of ants. Ants deposit chemical trails (pheromone) on the ground to communicate with each other. This allows them to find the shortest paths between nests to destinations. ACS algorithms have thus been used to discover good solutions to many applications (Chang and Lin 2013; Gambardella et al. 2012; Hong et al. 2009, b; Madureira et al. 2012; Verma et al. 2012; Yan and Shih 2012). They are also adopted to solve algebraic equations in mathematics (Pourtakdoust and Nobahari 2004).

Unfortunately, ant-related algorithms are originally designed for a discrete solution space. Therefore, continuous

solution spaces must be encoded into discrete solution spaces for ant-related algorithms to execute. In this case, it is hard for the global optima of continuous variables problems to be contained in an encoded discrete solution space. Improving the ant-related algorithm precision to approach the global optima needs to increase encoding length as well. It causes an ant needs to select more nodes to finish its path, ant-base algorithm needs to spend more computation time to train a path and approach to the optima solution.

Ant-related methods were thus extended to support continuous solution spaces (Kuhn 2002; Li and Xiong 2008; Monmarche et al. 2000). In general, the whole process was modified to make the methods work well in a continuous solution space. However, these methods preserved few characteristics of the original ant-related algorithm, losing a lot of its advantages. For example, some approaches just get the concept of pheromone, but an ant-base algorithm always refer to previous selected node when an ant chooses an edge. In addition, ACS has good searing behavior by the well-defined pheromone updating policy (local updating and global updating). But these algorithms don't implement these operators. The paper thus proposes for continuous solution spaces an ant-related algorithm which retains the process of the original ant-related algorithm to preserve its good characteristics. The proposed algorithm recommends a new concept of representing the pheromone by a distribution function for ACS, such that it can easily simulate and handle a continuous search space. It can dynamically generate edges between two nodes and give pheromone measures for them in the solution space. In addition, it can easily extend the originally discrete operators of ACS for dispersed possible solutions in the continuous version directly. Experiments on maximizing several constrained functions are also made, and the results show that the proposed approach works well in solving continuous variables problems.

2 Review of ant colony systems

This section reviews work related to ACS. The concepts of the ant algorithm and ACS, and some previous approaches for using ACS in a continuous solution space are briefly described.

2.1 Ant system

The ant system which was first introduced by Colorni et al. (1991) and Dorigo et al. (1996) is based on observations of real ant colonies searching for food. Ants are capable of cooperating to solve complex problems, such as searching for food and finding the shortest paths between their nests and their destinations. When ants move, they deposit pheromone on the path. Subsequent ants determine the next direction on the route according to the pheromone density. Once all the ants have terminated their tours, the amount of pheromone on the tours will have been modified. The traditional ant system is shown in Fig. 1.

2.2 Ant colony system

ACS proposed by Dorigo and Gambardella (1997), is an algorithm for finding solutions to optimization problems. It modified the pheromone updating policies (local updating rule and global updating rule) and state transition rule from the original ant system in order to increase the speed of convergence and the ability of exploration and exploitation. The detailed process is shown in Fig. 2.

Details of the algorithm are described below.

2.2.1 State transition rule

The state transition rule is used by an ant to probabilistically decide its next state. Take the travelling salesman problem as an example. Assume the kth ant is currently in the city (node) j. The next city (node) s for the kth ant to visit to form a partial solution is:

$$s = \begin{cases} arg^{max}_{n \epsilon R_k(j)} \left\{ [\tau(j,n)]^{\alpha} \times [\eta(j,n)]^{\beta} \right\}, & if\ q \le q_0 \\ i\ with\ a\ probability\ P_k(j,i), & if\ q > q_0 \end{cases} \quad (1)$$

where $R_k(j)$ is the set of cities that have not been visited by the ant, $\tau(j, n)$ is the pheromone intensity which expresses the history of previous successful moves on the edge from city j to city n, $\eta(j, n)$ is the inverse of the distance between the cities j and n, and α and β (>0) are two parameters that determine the relative importance of pheromone versus distance in the problem. In addition, the parameter q is a random number uniformly distributed between 0 and 1, q_0 is a parameter ($0 \le q_0 \le 1$) predefined by users, and

```
Initialize
    Set the size of the ant colony's population and put each ant on the starting node
    do
        while (there are some ants which have not already built their solution)
            Choose an ant which has not finished its trip
            The ant applies a state transition rule to incrementally build a solution
        end
        Update the pheromone for all completed trips in this iteration
    while (end conditions are not met)
    Output the best solution
End
```

Fig. 1 AS algorithm for an optimization problem

```
Initialize
    Set the size of the ant colony's population and put each ant on the starting node
    do
        while (there are some ants which have not already built their solution)
            Choose an ant which has not finished its trip
            The ant applies a state transition rule to incrementally build a solution
            Update the pheromone using the local updating rule
        end
        Update the pheromone using the global updating rule
    while (end conditions are not met)
    Output the best solution
End
```

Fig. 2 ACS algorithm for an optimization problem

$P_k(j, i)$ is the transition probability from city (node) j to city (node) i, calculated as follows:

$$P_k(j,i) = \begin{cases} \dfrac{[\tau(j,i)]^{\alpha} \times [\eta(j,i)]^{\beta}}{\sum_{n \in R_k(j)} [\tau(j,n)]^{\alpha} \times [\eta(j,n)]^{\beta}}, & \text{if } i \in R_k(j) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

q_0 is used to balance exploration and exploitation, i.e. if $q \leq q_0$, the ACS algorithm exploits the search space by favoring the best edge; otherwise, the algorithm more explores the search space.

If the value of the random variable q is less than the predefined parameter q_0, the state transition rule selects the node with the highest $[\tau(j,i)]^{\alpha} \times [\eta(j,i)]^{\beta}$ to follow. Otherwise, it probabilistically selects a node according to Eq. (2). The above selection method is called the pseudo random proportional rule.

In the general case, the function $\eta(j, n)$ is a heuristic function which expresses desirability of the move related to the optimization. It is an important and useful design for ACS. An appropriate heuristic function is set according to the domain knowledge of the given application. Because of the local updating rule (described below) for pheromone, the influence of pheromone on local trap will be reduced if most ants pass the same edge. It could guide an ant population to a good solution effectively and efficiently and lead the population to escape local optima.

2.2.2 Global updating rule

After all the ants have completed their tours, the pheromone density of the best tour passed is updated. There are two kinds of global update. The first kind takes the best tour among the ones passed by the ants in all the so far executed iterations and then updates the amount of pheromone on the edges of the best tour. Formally, it is stated as follows:

$$\tau_{(t+1)}(j,s) = (1-\rho) \times \tau_t(j,s) + \rho \times \Delta\tau(j,s), \text{and}$$
$$\Delta\tau(j,s) = \begin{cases} (1/L_{GBest}), & \text{if } (j,s) \in \text{Global best tour} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where t is the current time (iteration), ρ is the evaporation parameter of pheromone, $0 < \rho < 1$, and L_{GBest} is the length of the globally best tour finished by the ants from the beginning to the current iteration. The other kind of global update takes the best tour among the ones passed by the ants in each individual iteration. The amount of pheromone on the edges of the best tour is updated as follows:

$$\tau_{(t+1)}(j,s) = (1-\alpha) \times \tau_t(j,s) + \alpha \times \Delta\tau(j,s), \text{and}$$
$$\Delta\tau(j,s) = \begin{cases} (1/L_{IBest}), & \text{if } (j,s) \in \text{Iteration-best tour} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where L_{IBest} is the length of the best tour finished by the ants in each iteration. The global updating rule will provide a greater increase of pheromone to a shorter best tour.

2.2.3 Local updating rule

When an ant chooses an edge between nodes j and s, it immediately updates the pheromone density of the edge to avoid local optimum as follows:

$$\tau_{(t+1)}(j,s) = (1-\rho) \times \tau_t(j,s) + \rho \times \tau_0 \tag{5}$$

where $\tau_{(t+1)}(j,s)$ and $\tau_t(j,s)$ are the pheromone amounts on the edge from city j to city s in the ith and the $(i+1)$th iterations, respectively, and ρ is a parameter for adjusting the pheromone density of the edge, $0 < \rho < 1$. If the pheromone density $\tau_t(j,s)$ on the constructed path is higher than the initial value τ_0, the local updating rule decreases the pheromone density to reduce repeated visiting probability from the edge. Otherwise, it increases the pheromone density.

The local updating rule gives a reasonable probability to any possible solution. An edge, which is the local optima for an ant but not global optima for all the ants, will decrease its pheromone, causing the ant to explore the other edges with higher probabilities and thus avoiding local optima. According to the local updating process above, an ant usually decreases the pheromone density of the edge which it chooses. It prevents the ant population from selecting similar paths and makes sure that other possible solutions have a good chance to be selected.

2.3 Existing ant-related algorithms for continuous solution spaces

Traditionally, ant-related algorithms are applied to discrete problems, which are represented as a graph with nodes and

edges. When an ant finishes its trip from start to destination, a feasible solution is produced immediately. The advantage of an ant-related algorithm is that it is a constructive algorithm, which is different from genetic algorithms (GAs). An ant-related approach will refer to previous selected edges (nodes) according to a heuristic function and the information of pheromone to choose the next edge, thus being able to produce a suitable partial solution. The ants then choose better edges and a better solution can thus be found. For the above reasons, an ant-related algorithm is suitable for problems in which good solutions (not necessarily optimal) can be found using local search algorithms.

In the past, it was difficult to use ant-related algorithms to solve problems with a continuous solution space due to the coding restriction. A common way to deal with this issue is to map a continuous solution space to its simplified discrete solution space. However, this creates the following two problems. Firstly, a continuous solution space cannot be totally mapped to a discrete solution space, such that the global optimum may not exist in the encoded space. Secondly, if the minimal discrete distance scale of the encoded space is reduced, the coding length increases, and it lowers the performance of ant-related algorithms.

Bilchev and Parmee (1995) applied an ant algorithm called continuous ant colony optimization (CACO) to a continuous solution space and some other related approaches were proposed in Dreo and Siarry (2004), Kuhn (2002), Li and Xiong (2008) and Monmarche et al. (2000). In CACO, the solution space is defined as a plane (2-dimensional solution space) and each point in the plane is a possible solution. A nest is a point in the plane and all the ants start their tours there. In each iteration, the population may or may not move its nest to a new place, depending on whether the new place approaches the global optimum. However, these methods are more similar to particle swarm optimization (PSO) than to the original ant algorithm. Each position of an ant indicates a suitable solution in the solution space. When an ant moves to another position, it generates a solution immediately. It is very different with traditional ant-base algorithm. An ant-based algorithm will construct its own solution step by step, moreover a well-defined heuristic function will guide ants to select the more reasonable edges to build a solution. CACO just utilizes the concept of pheromone but not implements other essential operators for ant-based algorithm. Thus, this paper focuses on proposing a method which has the same process and operators with the traditional ACS and supports the continuous space.

Pourtakdoust and Nobahari (2004) then proposed an extended ACS algorithm to solve the math equation. Its process is similar to that of traditional ACS. However, it is not a general algorithm for all problems with a continuous solution space. In this approach, the operators are designed by depending a mathematics equation. In other words, this algorithm is just suitable to be applied an application which has a mathematics equation as fitness function. The paper thus proposes a continuous ACS algorithm that retains benefits and characteristics of the original ACS algorithm and can be easily applied to problems with a continuous solution space.

Socha and Dorigo (2008) proposed another ACO algorithm (ACO_R) in continuous domains. In their algorithm, the probability density function (PDF) is designed for sampling candidate nodes and keeping the information of pheromone. The PDF for each dimension of variable in the continuous space is defined as a Gaussian function. Their experiments show the well-defined algorithm can get good performance in mathematical equations.

In our proposed algorithm, the PDF is simplified as a pheromone distribution function but there are several different points between our proposed ACS and Socha et al's ACO_R. First, the proposed algorithm is designed to apply the original ACS into continuous space. It not only corresponds to the classical process and operators from ACO, but also uses both the global and local pheromone updating processes in ACS, ACO_R designed its pheromone updating policy in a solution pool (table) and could not perform global and local pheromone updating processes. Second, the pheromone information in ACO_R is stored in a table which is generated by the top-*k* solutions in the recent iteration. In our proposed algorithm, the concept of pheromone is similar to the traditional ACO. That is, the amount of pheromone is derived based on accumulation and evaporation by the pheromone updating operators. It can retain the experience of ants from the previous iterations. Third, ACO_R will generate the PDF by all of the variables in the resolved formula for each solution in a table. In the process of ACO_R, an ant will generate a fixed number of variables from its path. That is, ACO_R is suitable for solving a problem which solution has a fix number of variables like finding solution for a mathematics formula. On the other hand, the behaviors of the proposed algorithm are like those of the original ACO. It is more flexible and can own a complicated searching map with different topologies by applied solution, ants construct their own path by passing different number of nodes. Thus, the proposed algorithm is more suitable for different kind of real applications in continuous space, not only solving a mathematics formula.

In the proposed algorithm, the information of pheromone is stored in the nodes, the relationship between two nodes is weak. It cannot indicate the relationship (searching map) among the nodes as the traditional ant-related algorithms can. No matter what is the value obtained from the previous selection process, it doesn't affect the function

which is used to generate the pheromone content in the current selection. We will discuss how to enhance the relationship by the pheromone distribution function in the section below.

3 Dynamic-edge ACS algorithm

This section describes the proposed ACS-based algorithm for continuous solution spaces. It is called the dynamic-edge ant colony system (DEACS), and preserves but generalizes all of the operators in the original ACS, including the state transition rule, local updating process and global updating process. The continuous version of each operator is defined by pheromone distribution functions, which are explained below.

3.1 Pheromone distribution functions

In traditional ACS algorithms, an edge that can be selected includes an amount of pheromone. Unfortunately, it is not available in the continuous space since the numbers of edges and nodes are infinite in a continuous search space. The proposed DEACS algorithm thus adopts a new mechanism to store the pheromone information for resolving the issue. Assume a possible solution is composed of several variables. The proposed approach defines a function called pheromone distribution function onto the domain of each variable. Thus, each possible value of a variable will be assigned a content of pheromone by this distribution function.

DEACS works like the conventional ACS. The initial content of pheromone is defined before the first iteration. This means that the initial pheromone distribution function may be set as a constant function. After several iterations, some influence functions that are produced by the global updating process will be added onto the initial pheromone distribution function. These influence functions and the initial pheromone distribution function will combine and form a new pheromone distribution function. An example is shown in Fig. 3.

In Fig. 3, the initial pheromone distribution function is a constant function of 1.3. A part of the pheromone distribution is a trapezoid function (called an influence function) generated by the updating rules, which will be introduced later. In Fig. 3, the initial pheromone density is 1.3. The increase is caused from a previous route being selected by an ant. In this example, the node with the value 1.0 has a pheromone density of 1.3, and the one with 2.6 has a pheromone density of 2.5. Since DEACS works in a continuous environment, there will be infinite edges and nodes. The influence functions cannot only provide the information of pheromone but also are used to produce dynamic edges.

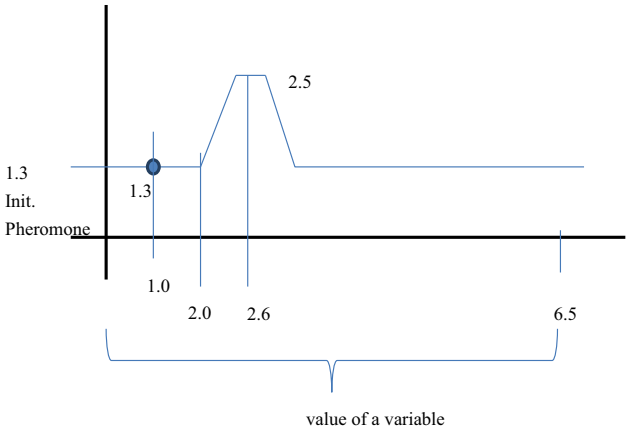


Fig. 3 Example of pheromone distribution for a variable

Different variables have their own pheromone distribution functions. For example, if a solution is composed of x and y, an ant will produce several dynamic edges using the pheromone distribution function of the x variable. After it selects a dynamic edge (value) of x, it then applies y's pheromone distribution function for determining the value of y. The length of the bases is a parameter in the DEACS, it means the range of influence from pheromone. Actually, the pheromone influence function (distribution function) is not a fixed trapezoid function. It even does not need to be a trapezoid function. The meaning of the influence function is the influential area of the selected node. If an influence function is set as a trapezoid function with wider parallel side, it has more affection of nearly area. This kind of influence functions is suitable to be applied in a smooth fitness function.

3.2 Global updating rule in DEACS

In traditional ACS, the global updating rule increases the pheromone value of the best tour and decreases those of the others. This concept is extended to DEACS in which the update is not performed only on a node (a single value of a variable), but on a set of neighboring nodes (a range of values of a variable). An influence (update) function on pheromone update is defined for achieving this task. DEACS not only increases the pheromone value of the best tour but also influence the content of pheromone in the nearby area. Therefore, the influence function is not only a single value, but more like a distribution function. The center of the function is the next node (value) selected. It indicates the influence of an ant passing the selected node on the update of the pheromone. The function can be of any type, and the flexibility is left to users. An example of a trapezoid influence function is shown in Fig. 4.

$$f_{inf}(x) = \begin{cases} h, & for\, p-s < x < p+s \\ max(h-d(x-p-s),0), & for\, x \geq p+x \\ max(h-d(p-s-x),0), & for\, x \leq p-x \end{cases} \quad (6)$$

In Fig. 4, the shape of the influence function is a trapezoid, and can be described as f_{inf} in formula (6), where h is the height of the trapezoid, p is the center of the influence function, s is the half of the upper line of the trapezoid and d is the decayed ratio for influence of pheromone. The value of the variable appearing at the center of the trapezoid represents the chosen (continuous) node of the best tour at a certain stage. Just like in traditional ACS, two kinds of global update may be used. The first one updates the pheromone distribution of the best tour among the ones passed by the ants in all the iterations, whereas the other updates that in each iteration.

When global update is executed, the influence function whose center is at the chosen value of a variable is superimposed on the original distribution function of the pheromone for the variable. For example, if the initial pheromone distribution is a uniform one, after an influence function with its center 2.6 is added, the new distribution function of pheromone becomes the one shown in Fig. 3.

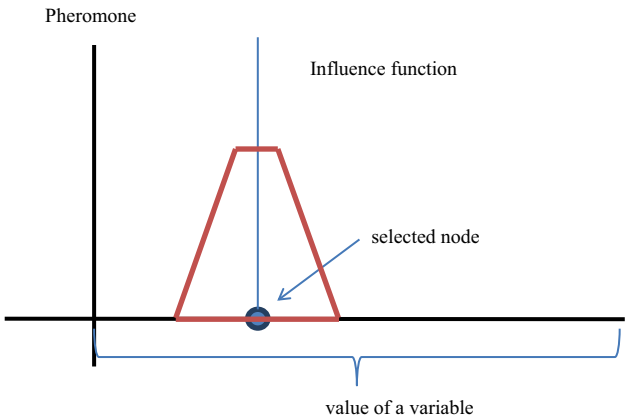


Fig. 4 Example of a trapezoid influence function

Assume the trapezoid type is selected as the influence function. Two parameters may be used to define a trapezoid shape, the span and the angle. The span defines the top flat segment of the trapezoid and the angle determines the two oblique lines, which represent the influence range of a best tour.

After some influence functions are added, the distribution function will have several summits. For example, when an influence function shown on the left of Fig. 5 is superimposed on the distribution function on the right of the figure, the result is shown in Fig. 6. If the initial pheromone is $f_{init}(x)$, which is a constant, the influence function 1 is $f_{inf1}(x)$ and the influence function 2 is $f_{inf2}(x)$. The previous pheromone distribution functions is $f_{dis}(x) = f_{init}(x) + f_{inf1}(x)$ and the updated pheromone distribution function is $f'_{dis}(x) = f_{init}(x) + f_{inf1}(x) + f_{inf2}(x)$.

In real implementation, DEACS maintains a table to record the influence functions attached to each pheromone distribution function. For example, assume that the initial pheromone density is 2.0 and that there is one influence function with center = 3.5 and height = 5. A record of the influence function is then put in the table, as shown on the left of Fig. 7. If another influence function is generated, the corresponding center and height are stored in the second record. The DEACS approach can thus easily calculate the actual distribution function from the initial function and the pheromone influence functions stored in the table.

As mentioned above, during the global updating process, the value of each node selected in the global best solution is inserted into the first column (center) of the corresponding influence table. The height value of the influence function caused from the node is calculated as:

$$Height = Pheromone_{initial} + Pheromone_{increased} \quad (7)$$

where $Pheromone_{initial}$ is the initial pheromone density for the node and $Pheromone_{increased}$ is the increased pheromone due to the selection of the node value. $Pheromone_{increased}$ may be a fixed value or a value that depends on a fitness function by which a better solution will cause larger pheromone increase. In the example of

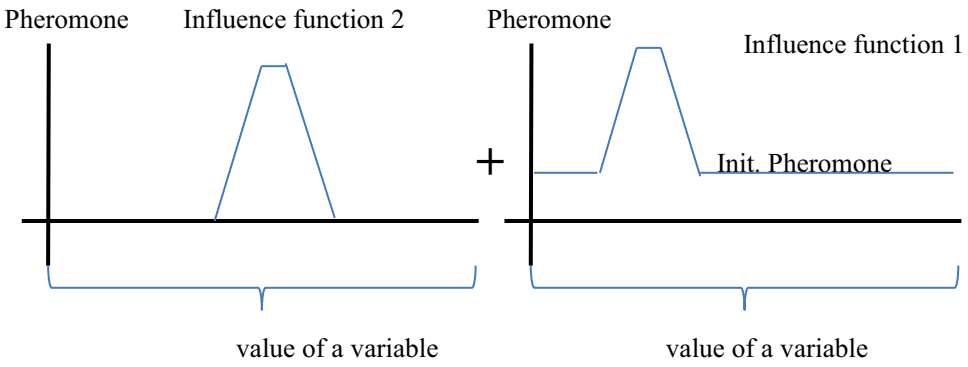


Fig. 5 Superimposition of an influence function onto a distribution function

Fig. 7, assume that $Pheromone_{increased}$ is set at 3. Thus, when the node value of 3.5 is selected, the height of the influence function for the selected node value is 2 + 3, which is 5. It is then inserted into the second column of the first record in the influence table. As in traditional ACS-based algorithms, a pheromone density decreases due to volatilization or a local update. The height of an influence function is then reduced. In the example above, assume that the height of the first influence function is reduced to 1.22 from the local update (described later). The result is shown in Fig. 8.

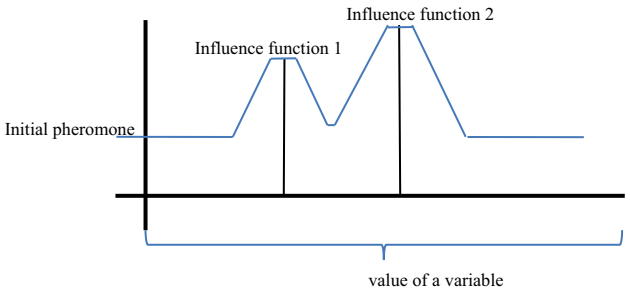


Fig. 6 Superimposition results from Fig. 5

In the next iteration, assume that the node value in the new best global solution is 7. A new influence function with center = 7 and height = 2 + 3 (=5) is then inserted into the second record of the influence table for the variable under consideration. The final distribution function and the corresponding influence table are shown in Fig. 9.

3.3 Local updating rule in DEACS

When an ant constructs a path, the local updating rule immediately reduces (volatilizes) the pheromone of the nodes on the path to prevent all ants in the population from searching toward similar solutions. This can be easily achieved by reducing the height values of the matched influence function representing the node. An influence function is matched if the value selected falls within the range of the influence function. If the reduced height value of an influence function is less than the initial amount of pheromone, the influence function is removed from the distribution function. In this way, DEACS removes some unimportant information, and thus reduces the storage space required. The new height value due to the local update is calculated as:

Fig. 7 Table for recording the influence functions of a variable

Center	Height
3.5	5

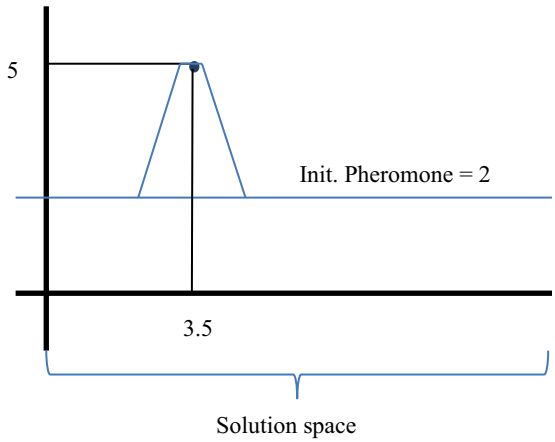
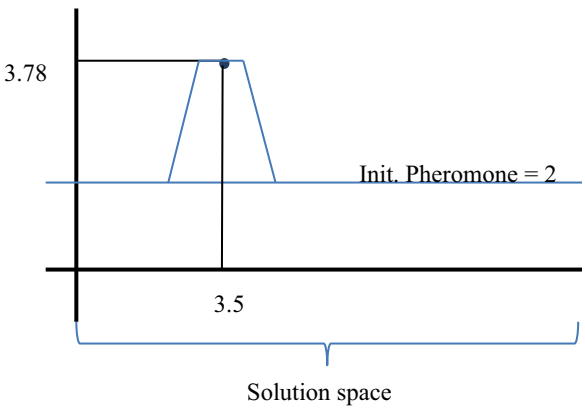


Fig. 8 Result of reducing an influence function

Center	Height
3.5	3.78



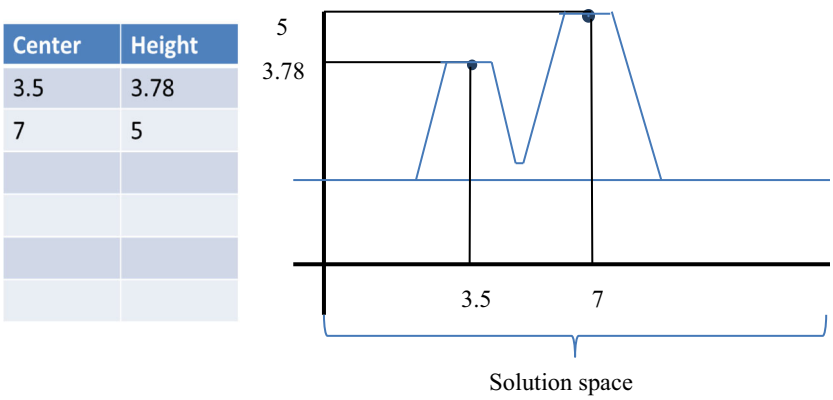


Fig. 9 Distribution function after two global updating iterations

$$Height_{new} = (1 - p) \times Height_{old} + p \times Pheromone_{base}, \quad 0 < p < 1 \tag{8}$$

where $Height_{new}$ is the new height value of the processed influence function after the local update, $Height_{old}$ is the height value before the local updating process, $Pheromone_{base}$ is a parameter value less than $Pheromone_{initial}$, and p is a weight constant controlling the two items. The parameter $Pheromone_{base}$ allows the height value of an influence function possibly less than $Pheromone_{initial}$ such that unimportant influence functions may be removed. Figure 10 shows an example of the change of the distribution function in Fig. 7 during the local updating process. An ant selects a value that falls within the range of the influence function. Assume that $Pheromone_{base}$ is set at 1.95 and p is set at 0.4. The new height value of the influence function is updated to $(1 - 0.4) \times 5 + 0.4 \times 1.95$, which is 3.78. The resulting influence function is then a smaller trapezoid with the same angle, as shown at the bottom of Fig. 10.

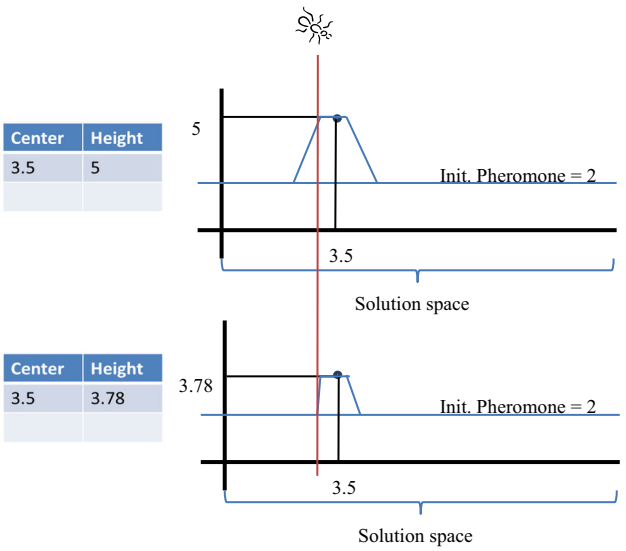


Fig. 10 Example of changing a distribution function by the local updating process

Note that the number of records in the influence table is efficiently controlled by the local updating process. After a few iterations, some records are stored in the table, some of which may be located close by. Therefore, when an ant chooses an edge to go, it is quite possible that the edge intersects several influence functions at the same time. In this case, the local updating process reduces all of them, and when the height of an influence function is below $Pheromone_{initial}$ after the update, it is removed from the table. Thus, the records in the table do not always grow.

3.4 Generating dynamic edges

When a value (node) of a variable is to be selected, there are an infinite number of choices due to the continuous space. It also means there are an infinite number of edges to be selected. Each edge starts from the node selected for the previous variable and ends at a certain value (node) of the current variable. Since there are numerous edges, the proposed approach thus dynamically generates an edge whenever necessary. It first calculates the total area A of the pheromone distribution function for the variable (dimension) being currently processed. The area A thus represents the current total amount of pheromone for the variable in solving the problem. The proposed approach then generates a random number r, whose range is among 0 to A. It then finds the value of the horizontal axis to which the integral of the distribution function from the minimum value of the variable equals r. For instance, in order to find a candidate node w, $0 \leq r \leq \int_x f(t) = A$, where x is the solution space $[v_i, v_u]$, r is a random number and f is a distribution function, the goal is to find a new $w = v_u'$ such that $\int_x f(t) = r$ where $x \in [v_i, w]$. The value of the horizontal axis can be thought of as a dynamic node for the variable, and a dynamic edge is formed between the node

selected for the previous variable and the current node. DEACS will generate several random numbers, each corresponding to a dynamic node and edge. It then selects one edge according to the pseudo-random proportional rule. Figure 11 shows an example of the above process.

Assume in Fig. 11, there is only one influence function (Center: 3.5, Height: 5) for the variable to be processed. The total amount A of pheromone in the distribution function is calculated as 26 by integral. After that, a random number within 0 to A is generated. Assume the generated number is 20. The horizontal axis value in the distribution function obtained by the proposed approach with an integral of 20 from the start is 7 as Fig. 11 shows. The ant thus generates a dynamic node with 7 for the variable (Fig. 12).

According to the characteristic of the integral function, most of the nodes generated in this way will be located in areas with high pheromone density. Moreover, no matter where a point is located in the solution space, its chance to be selected is bigger than zero. This behavior is very consistent with that of the original ACS.

3.5 Dependency between variables

In the process of node selection, each variable is handled according to its own pheromone distribution function. However, variables may depend on each other. That is, when a node of a variable is selected, it may influence the choice of the next node. The problem does not exist in traditional ant-related algorithms because pheromone is stored on edges.

For example, for finding a feasible solution of the problem $x + y = 100$, $0 < x, y < 100$ using DEACS, an ant will first select the value of x and then the value of y. If the ant selects the value 10.5 for x, then the best value of y will be 89.5. However, if the ant selects 95.1 for x, the best value of y will be 4.9. The variables x and y are thus closely related to each other. In this case, it is inappropriate to just choose the value of y according to its own distribution function without referring to the selected value of x. Taking this into consideration, DEACS adopts an optional mechanism for improving its flexibility in handling this kind of problems. It splits the solution space of each variable A into n_A equal portions, where n_A is predefined. Each portion is then attached with an individual pheromone distribution function of its next variable. For example in Fig. 13, the values of the variable x are divided into three equal intervals, and each one is linked to a different pheromone distribution function of the same next variable y.

In Fig. 13, when an ant chooses a value of x, it uses the corresponding distribution function of y according to the portion in which x is located. When more portions are adopted, the relationship between the two variables will be strengthened, but the algorithm will become more complex and need more storage.

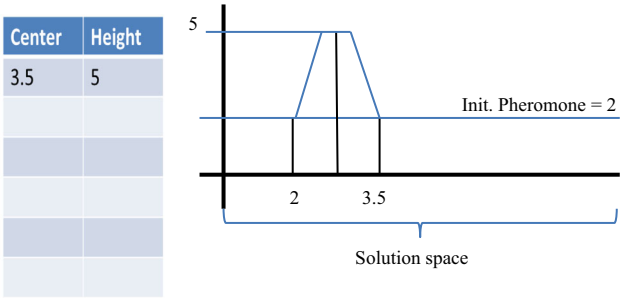


Fig. 11 Example of a distribution function with an influence function

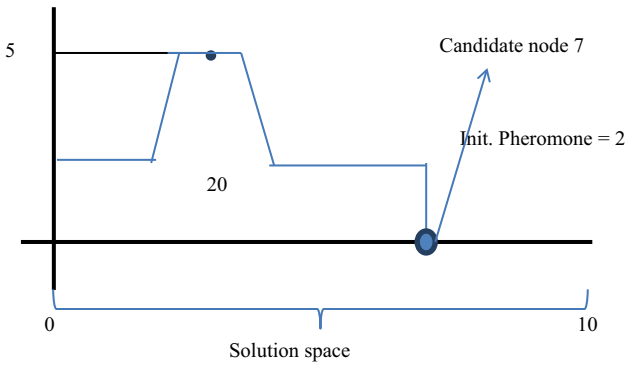


Fig. 12 Example of generating a dynamic node

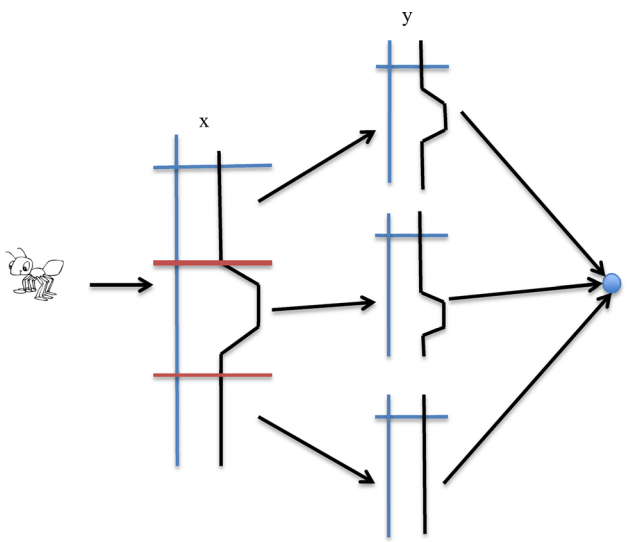


Fig. 13 Different distribution functions of y for portions of x

The strategy considers the dependency of the variables and can thus effectively improve the relationship between pheromone and chosen paths.

3.6 DEACS algorithm

The proposed DEACS algorithm incorporates the strategies mentioned above and is described as follows.

The DEACS algorithm:

INPUT: A problem to be solved, a number q of ants, an initial pheromone density τ_0, a number d of dynamic edges, a number k of portions for considering variable dependency, a maximum number G of iterations, a current best solution $S_c = \emptyset$ (empty) and give the predefined base pheromone density local updating ratio and global updating ratio.

OUTPUT: A nearly optimal solution to the problem.

STEP 1: Define the order of the variables as the stage order in the search graph.

STEP 2: Define an appropriate fitness function for evaluating paths.

STEP 3: Initially set the pheromone distribution function of each variable as the given initial pheromone density τ_0.

STEP 4: Set the initial generation number $g = 1$.

STEP 5: Build a complete route for each artificial ant by the following substeps.

STEP 5.1: Set $s = 1$, where s is used to keep the identity number of the current variable (stage) to be processed in the graph.

STEP 5.2: Get the corresponding pheromone distribution function of the sth variable among the k pheromone distribution functions for the sth variable by using the method in Sect. 3.5.

STEP 5.3: Produce d dynamic edges according to the method in Sect. 3.4.

STEP 5.4: Select a path from the d dynamic edges according to the pseudo-random proportional rule.

STEP 5.5: Initialize the tables of influence functions for current variable x_s, where s is the index for the current variable if the tables isn't exist.

STEP 5.6: Update the pheromone distribution function by modifying the record value in the table of influence functions from the selected edge according to the local updating rule mentioned in Sect. 3.3.

STEP 5.7: Set $s = s + 1$.

STEP 5.8: If the ant has not constructed its own solution (that is, s is equal to or less than the number of variables for the problem), go to STEP 5.2.

STEP 6: Evaluate the fitness value of the solution obtained by each artificial ant according to the fitness function defined in STEP 2. If S_c is $\emptyset$ (the first generation) or the best solution in this iteration is better than S_c, set S_c as the best solution.

STEP 7: Find the one with the highest fitness value among the q ants, and get the values of the variables for the best ant.

STEP 8: Generate the corresponding influence functions for the variable values found in STEP 7 and then update the distribution functions (the table of influence functions) of the variables according to the global updating process mentioned in Sect. 3.2.

STEP 9: If $g = G$, output the current best solution; otherwise, $g = g + 1$ and go to STEP 5.

The best solution S_c obtained by the algorithm is thus output after STEP 9.

4 Experimental results

Experiments were made to show the performance and behavior of the proposed DEACS. The experiments were implemented in C/C++ on a personal computer with an Intel Core 2 Quad 6600 CPU and 4 GB of RAM. Several mathematical functions with constraints for maximum values were used in the experiments. The experiments were divided into three parts. First, the experimental results show that the basic characteristics of DEACS and the difference performance between DEACS and traditional ACS by several simple tests, a traditional ACS was designed for calculating its maximum and its performance was compared to that of the proposed DEACS for showing its benefits and characteristics. The traditional ACS is designed to have a multi-stage graph, with each stage having four nodes (00, 01, 10, 11), as shown in Fig. 14. Each node represents a part of whole binary code that mean as a possible solution. For example, the ant selected the first 3 nodes are (0, 0), (0, 0) and (1, 1) in the Fig. 14. It means the ant produced a 00 00 01 which is a part of whole binary string be the head of the total encoding code. The parameters used in the two approaches were set as follows. The number of ants was 10, the initial pheromone density was 1.0, the base pheromone density was 0.8, the local updating ratio was 0.2, and the global updating ratio was 0.8, the number of dynamic edges is 5. The results were averaged by 1000 runs.

The following two functions were used to find the maximum result and x, y are between 0 and 100 for testing:

(A) $100 - (x - 35.52)^2 - (y - 13.11)^2$, $x, y \in [0, 100]$ and

(B) $100 - (35.52 - (x + y))^4 - (x - 5.35)^2 - (y - 30.17)^2$, $x, y \in [0, 100]$.

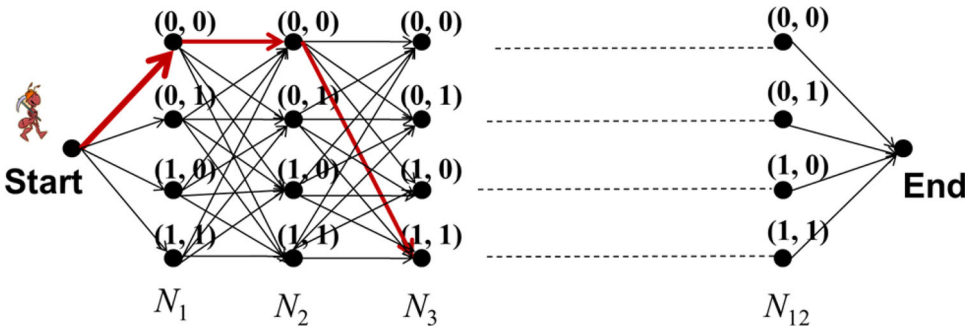


Fig. 14 Multi-stage graph for the traditional ACS algorithm

The two functions had two variables, x and y. In the first one, the two variables were independent, and in the second one, they are dependent. Experiments were first made to show the evolution of the fitness values, which is function (A) to find the maximum result by DEACS and ACS. The average fitness values (maximum function results) of the ten artificial ants along with the number of generations are shown in Fig. 15.

The figure shows that the average fitness values obtained by DEACS were better than those obtained by ACS. Besides, DEACS obtained good results even in the early stage of the process because the length of the search graph was short. Thus, the ants just need to select fewer edges then traditional ACS to finish their trip. Then it can find a good trip between start point and destination easily. The average fitness values for ACS and DEACS for function (B) are shown in Fig. 16.

Figure 16 also shows that the average fitness values obtained by DEACS were better than those obtained by ACS. DEACS did not approach the global best solution in the first 50 generations because x and y were dependent. Section 3.5 introduced a strategy for increasing the association between two variables. For verifying the effectiveness of that strategy, the solution space of each variable was split into 1 (original), 2 and 4 partitions. The average fitness values for Functions (B) are shown in Fig. 17, respectively, where DEACS(n) indicates that the solution space of the variables x and y where split into n portions.

The figures show that DEACS(2) had the best performance among the three. Even though the association of the

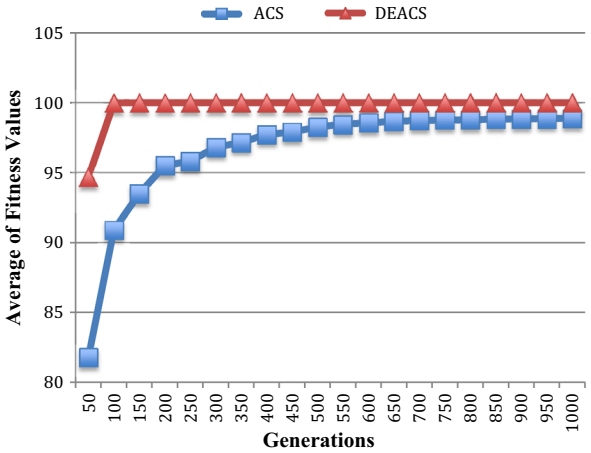


Fig. 16 Average fitness value along with number of generations for function (B)

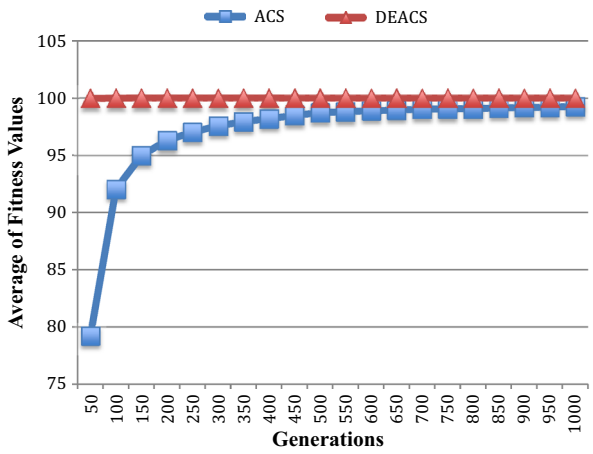


Fig. 15 Average fitness value along with number of generations for function (A)

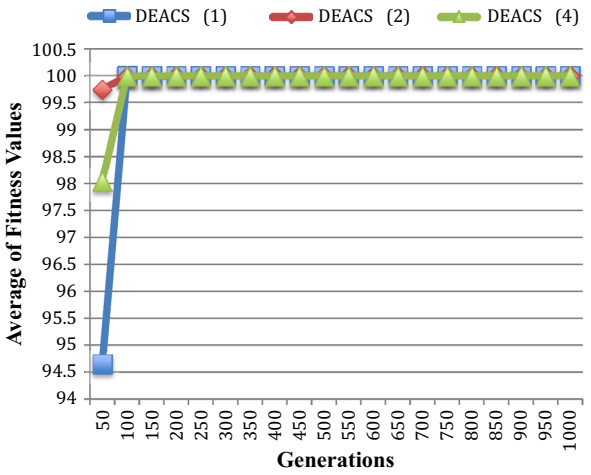


Fig. 17 A comparison for different numbers of portions on function (B)

two variables in DEACS(4) was stronger than that in DPACS(2), more pheromone distribution functions needed to be trained in DPACS(4). DEACS(4) needs more generations to find the suitable variables in more separated distribution functions. Therefore, the performance of DEACS(4) was poorer than that of DEACS(2). The number of dynamic edges is a parameter (option) in the proposed algorithm. If the number of dynamic edges is too large, it will increase the computation time and the memory usage. If the number of dynamic edges is too small, it will reduce the explorative ability of the proposed algorithm. Therefore, DEACS needs to be set a suitable number of dynamic edges according to the scale of solution space. If a program has a complicated and large scale solution space, it need to generate more dynamic edges to discover the best solution.

Experiments were made to compare the average execution time for function A of DEACS and ACS. The results are shown in Fig. 18.

The figure shows that DEACS spent less execution time than ACS. The reason was that the stages from the start to the end in DEACS are few. Note that the traditional ACS needed several nodes (stages) to encode a variable, but DEACS just needed one stage for a variable. Thus, the total stage number of DEACS is only $1/n$ of that of the traditional ACS.

In the second part, DEACS was then compared to some existing approaches including API (Monmarche et al. 2000), GA, and CACS (Pourtakdoust and Nobahari 2004) with seven benchmark functions. The functions and experimental results referred from Pourtakdoust and Nobahari (2004). The benchmark functions are tabulated in Table 1.

All of the results of formulas have a minimum value of 0. The algorithms calculated these formulas and try to find the minimum value. Thus the more minimum value of results (close to zero) means the better preference of these methods. The experimental results are summarized in Table 2.

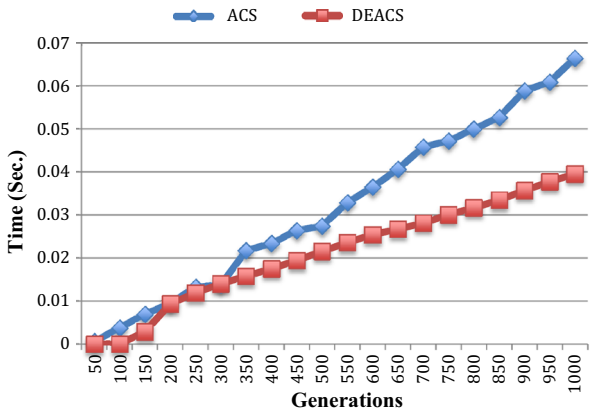


Fig. 18 Average execution time along with number of generations by DEACS and ACS

CACS is a special-purpose ACS for resolving mathematical formulas. It uses some complex designs to approach the optimal solution for a formula. For example, it applies a normal probability distribution function to calculate the content of pheromone and adjusts the parameters by using the best fitness value. GA is the traditional genetic algorithm and it decodes the solution space as a binary string. API is a method, which is inspired by a primitive ant's recruitment behavior. The process performs two ants and leads them to gather on a same hunting site. The recruitment technique makes the population proceed towards the optimum solution. In this experiment, no heuristic function was designed. The same parameter settings from the previous experiments were used. From Table 2, it could be observed that DEACS and CACS could obtain satisfactory solutions for these formulas when compared to the others. Because of the encoding spaces are dispersed in GA and API and formulas 1 and 2 are simple formulas, GA and API can obtain the minimum value 0 as results in these two formula. DEACS and CACS just get the values that are very close to 0. In the other complex formulas, DEACS and CACS can obtain better performance than other two methods. CACS has shown good performance in resolving formulas. In contrast, DEACS is a general continuous ACS for any problem and it can also obtain well results in resolving formulas. Thus, DEACS is as flexible as traditional ACS, but can also have good performance in the continuous problem.

In the last part of experiments, DEACS was compared with some other previous ant-related continuous algorithms (Bilchev and Parmee 1995; Dreo and Siarry 2004; Monmarche et al. 2000) according to the same test in Socha and Dorigo (2008). The same stopping criterion was used as follows:

$$|f - f^*| < \epsilon_1 f + \epsilon_2 \tag{9}$$

where f is the value of the best solution found by a method and f^* is the optimal value for the given test problems. ϵ_1 and ϵ_2 are set as 10^{-4}. The benchmark functions are shown in Table 3.

Table 4 shows the performance of these algorithms for the benchmark functions. Note that Function 7 uses ten variables. The value in the table represents the relative ratio of the mean number of function evaluations by an approach over that of the best one. The best performing algorithm on a given problem was thus recorded as 1.0, and the others were counted as a multiple of it. The actual mean number of function evaluations is given in parentheses only for the best performing algorithm on a given problem. The value in a square bracket indicates the percentage of successful

Table 1 Seven benchmark functions in part 2

1. $x_1^2 + x_2^2 + x_3^2$, $x_i \in [-5.12, 5.12]$ (Sphere Function)
2. $100\left(x_1^2 - x_2\right)^2 + (1 - x_1)^2$, $x_i \in [-5.12, 5.12]$ (Rosenbrock)
3. $50 + \sum_{i=1}^{5}\left(x_i^2 - 10cos(2\pi x_i)\right), x_i \in [-5.12, 5.12]$ (Rastrigin)
4. $1 + \sum_{i=1}^{2}\frac{x_i^2}{4000} - \prod_{i=1}^{2} cos\left(\frac{x_i}{\sqrt{i}}\right), x_i \in [-5.12, 5.12]$ (Griewank with 2 variables)
5. $1 + \sum_{i=1}^{5}\frac{x_i^2}{4000} - \prod_{i=1}^{5} cos\left(\frac{x_i}{\sqrt{i}}\right), x_i \in [-5.12, 5.12]$ (Griewank with 5 varables)
6. $0.5 + \left(sin^2\left(x_1^2 + x_2^2\right)^{1/2} - 0.5\right)/\left(1 + 0.001\left(x_1^2 + x_2^2\right)\right), x_i \in [-100, 100]$
7. $\left(x_1^2 + x_2^2\right)^{0.25}\left(1 + sin^2 50\left(x_1^2 + x_2^2\right)^{0.1}\right)$, $x_i \in [-100, 100]$

Table 2 Comparison of DEACS and three other methods

Method	min(f_1)	min(f_2)	min(f_3)	min(f_4)	min(f_5)	min(f_6)	min(f_7)
DEACS	3.11e−7	5.33e−8	3.3e−2	3.6e−3	5.9e−7	4.2e−3	4.5e−2
CACS	1.5e−67	1.2e−31	4.8	5.0e−3	1.1e−2	4.6e−3	4.2e−6
API	0.0	0.0	7.47651	0.00413	0.25034	0.00659	0.09307
GA	0.0	0.0	2.12457	0.03095	0.13955	0.07376	0.13358

Table 3 Six benchmark functions in part 3

1. $\sum_{i=1}^{2} 100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2, x_i \in [-5, 5]$ (Rosenbrock R_2)
2. $x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$, $x_i \in \{-5.12, 5.12\}$ (Sphere)
3. $\left(1 + (x_1 + x_1 + 1)^2\left(19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right)$
   $\cdot \left(30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 - 48x_2 - 36x_1x_2 + 27x_2^2\right)\right), x_i \in \{-5.12, 5.12\}$
   (Goldstein and Price)
4. $(x_1 - x_2)^2 + \left(\frac{x_1 + x_2 - 10}{3}\right)^2, x_i \in \{-20, 20\}$ (Martin and Gaddy)
5. $x_1^2 + 2x_2^2 - \frac{3}{10}cos(3\pi x_1) - \frac{2}{5}cos(4\pi x_2) + \frac{7}{10}, x_i \in \{-100, 100\}$ (B_2)
6. $\sum_{i=1}^{5} 100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2$, $x_i \in \{-5, 5\}$ (Rosenbrock R_5)
7. $1 + \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right), x_i \in \{-5.12, 5.12\}, n = 10$ (Griewangk GR_{10})

Table 4 Comparison of DEACS and the other four methods

Method	f_1	f_2	f_3	f_4	f_5	f_6	f_7
DEACS	1.0 (766)	1.0 (546)	1.35	1.22	1.0	1.57	1.0 (1156)
ACO_R	1.07	1.43	1.0 (384)	1.0 (345)	1.06	1.0 [97 %] (2487)	1.2 [61 %]
CACO	8.88	40	14	5	–	–	36
API	12.84	18.59	–	–	–	–	–
CIAC	14.98	91.52	61 [56 %]	34 [20 %]	23.32	16 [90 %]	36 [52 %]

runs. These experiments could estimate the convergence speed of the proposed DEACS.

It could be observed from Table 4 that the number of convergence generations of DEACS was similar to that of ACO_R. These two approaches were better than the other compared methods. Besides, there was no absolute rank between DEACS and ACO. Due to different pheromone policies between DEACS and ACO_R, DEACS usually performed more rough searches in a complex problem. The local pheromone updating rule, however, caused ants to escape from local traps easily. For example, DEACS could stably approach the global optimal solution in Function 4 even it used more function evaluations.

The experimental results showed that DEACS could obtain good performance in some formulas in which there is weak relationship among variables. For example, it

performs well in Function 7 because if the value of a variable approaches zero, the result of this function will approach zero no matter what the values of other variables are. Besides, DEACS retains the ability of escaping from local optimal solutions because it keeps the original process and characteristics of ACS. It can usually obtain a solution near a global optimal solution, even though the number of variables is large.

5 Conclusion and future work

This work proposed an extended ACS algorithm for solving continuous variables problems. The proposed algorithm is different from the existing ant-related algorithms in that it does not have fixed paths and nodes. Instead, it dynamically produces paths in the continuous solution space by applying distribution functions of the pheromone. The proposed DEACS algorithm retains all the benefits and the operators (including the policy of pheromone update and state transition rule) of the traditional ACS algorithm. The experimental results show that DEACS is very competitive to the existing ACS and some other evolutionary algorithms. In addition, it is a general algorithm that can easily be applied to a wide range of continuous optimization problems.

Even though the performance of the proposed DEACS is good and is very similar to the traditional ACS algorithm, there are still some issues to be further explored. First, it simply stores pheromone in nodes, which weakens the relationship between two nodes. Although the method of dividing the solution space is used to resolve this problem, it does not completely resolve the problem as the searching process becomes not very smooth. When the process chooses a new, untrained partition, it has to start training all over again.

Second, influence function represents the impact of the path on the surrounding area. However, we haven't had discussion about the impact of different process. In the future, the structure of DEACS will be further improved. For example, pheromone can be stored somewhere between two nodes to further represent the relations between them. The applications of various types of influence functions to different kinds of problems may also be discussed. Finally, DEACS retains the operators and the process of the traditional ACS. It obtains good performance in some formulas in which there is weak relationship among variables. Besides, it is more flexible to other real applications (in addition to mathematical formulas) than the compared ant-based algorithms. as long as a suitable heuristic function is designed. Thus, we will apply it to some applications with reality in the future.

References

Bilchev G, Parmee IC (1995) The ant colony metaphor for searching continuous design spaces. In: The AISB workshop on evolutionary computation, vol 993, pp 25–39

Chang MS, Lin HY (2013) An immunized ant colony system algorithm to solve unequal area facility layout problems using flexible bay structure. In: The Institute of Industrial Engineers Asian Conference, pp 9–17

Colorni A, Dorigo M, Maniezzo V (1991) Distributed optimization by ant colonies. In: European Conference on Artificial Life, Paris, France, pp 134–142

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

Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26(1):29–41

Dreo J, Siarry P (2004) Continuous interacting ant colony algorithm based on dense hierarchy. Future Gener Comput Syst 20(5):841–856

Gambardella LM, Montemanni R, Weyland D (2012) An enhanced ant colony system for the sequential ordering problem. In: Operations research, pp 355–360

Hong TP, Tung YF, Wang SL, Wu MT, Wu YL (2009a) An ACS-based framework for fuzzy data mining. Expert Syst Appl 36(9):11844–11852

Hong TP, Tung YF, Wang SL, Wu YL (2009b) Multi-level ant-based algorithm for fuzzy data mining. In: North American Fuzzy Information Processing Society Annual Conference

Jiang WJ, Xu YH, Xu YS (2005) A novel data mining algorithm based on ant colony system. In: International conference on machine learning and cybernetics, Guangzhou, vol 3, pp 1919–1923

Kuhn L (2002) Ant colony optimization for continuous space. A thesis submitted to The Department of Information Technology and Electrical Engineering The University of Queensland

Li H, Xiong S (2008) On ant colony algorithm for solving continuous optimization problem. In: International conference on intelligent information hiding and multimedia signal processing, pp 1450–1453

Madureira A, Falcao D, Pereira I (2012) Ant colony system based approach on single machine scheduling problems: weighted tardiness scheduling problem. In: Nature and biologically inspired computing, pp 86–91

Monmarche N, Venturini G, Slimane M (2000) On how *Pachycondyla apicalis* ants suggest a new search algorithm. Future Gener Comput Syst 16(8):937–946

Pourtakdoust SH, Nobahari H (2004) An extension of ant colony system to continuous optimization problems. In: Lecture Notes in Computer Science, vol 3172/2004, pp 158–173

Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. Eur J Oper Res 185(3):1155–1173

Verma OP, Kumar P, Hanmandlu M, Chhabra S (2012) High dynamic range optimal fuzzy color image enhancement using artificial ant colony system. Appl Soft Comput 12:394–404

Yan S, Shih YL (2012) An ant colony system-based hybrid algorithm for an emergency roadway repair time-space network flow problem. Transportmetrica 8:356–361