An Extended ACS Approach for Continuous Domains

Min-Thai Wu[#1], Tzung-Pei Hong[*2], Chung-Nan Lee[#3]

[#]*Department of Computer Science and Engineering, National Sun Yat-sen University*
70, Lienhai Rd., Kaohsiung 804, Taiwan, R.O.C.
[1]d953040015@student.nsysu.edu.tw
[3]cnlee@cse.nsysu.edu.tw

[*]*Department of Computer Science and Information Engineering, National University of Kaohsiung*
700, Kaohsiung University Rd., Kaohsiung 811, Taiwan, R.O.C
[2]tphong@nuk.edu.tw

Abstract— **This paper proposes a dynamic-edge ACS (DEACS) 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. The proposed algorithm retains the original process and characteristics of the traditional ACS. Several constrained functions are also evaluated to demonstrate the performance of the proposed algorithm.**

Keywords— **Ant colony system, Constrained function, Continuous space, Distribution function, Dynamic edge.**

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 [5] with positive feedback to search for solutions. ACS algorithms have been used to discover good solutions to many applications. They are also adopted to solve algebraic equations in mathematics [8].

Unfortunately, ant-based algorithms are originally designed for a discrete solution space. Therefore, continuous solution spaces must be encoded into discrete solution spaces for ACS to execute.

Several ant-based methods were proposed to support continuous solution spaces [5][6][7]. 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 ACS algorithm, losing a lot of its advantages. The paper thus proposes for continuous solution spaces an ACS algorithm, which retains the process of the original ACS 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. 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 Colony System

ACS [2], proposed by Dorigo et al., is an algorithm for finding solutions to optimization problems. Details of the algorithm are described below.

```
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
```

2.2. Existing ACS Algorithms for Continuous Solution Spaces

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

In the past, it was difficult to use ant-based 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-based algorithms.

Nest-based algorithms were then proposed for applying an ant algorithm to a continuous solution space [5][6][7]. The solution space is defined as a plane (2-dimentional solution space) and each point in the plane is a possible solution. However, these methods are more similar to particle swarm optimization (PSO) than to the original ant algorithm.

Pourtakdoust and Nobahari then proposed an extended ACS algorithm to solve the math equation [8]. Its process is similar to that of traditional ACS. However, it is not a general algorithm for all problems with a continuous solution space. 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.

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). The continuous version of each operator is defined by pheromone distribution functions, 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. **1**.

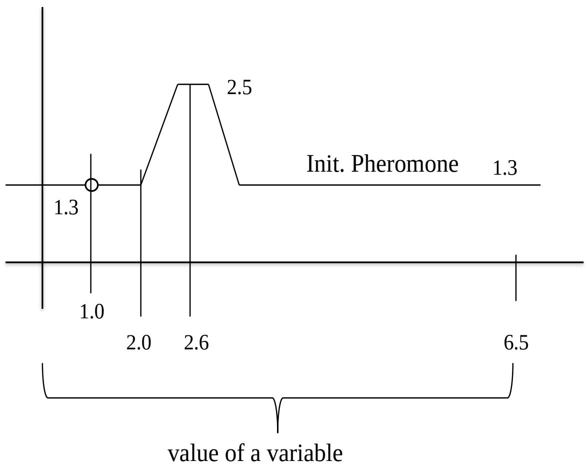

Fig. 1 Example of pheromone distribution for a variable

In Fig. **1**, 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. 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.

3.2. Global Updating Rule in DEACS

The global updating rule increases the pheromone value of the best tour and decreases those of the others. 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. **2**.

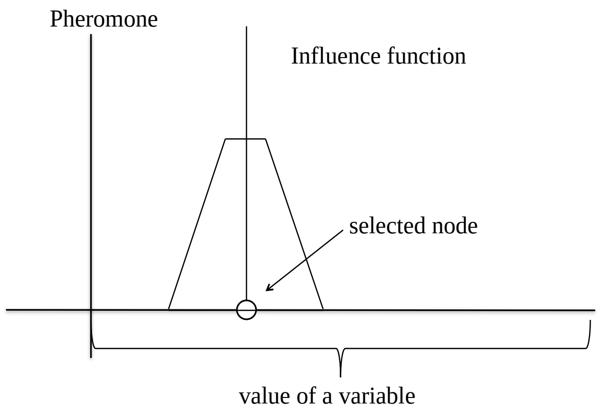

Fig. 2 Example of a trapezoid influence function

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. **1**.

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:

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

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.

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. 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. 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.

3.5 Proposed DEACS Algorithm

The proposed algorithm is proposed below.

INPUT: A problem to be solved, a number q of ants, an initial pheromone density τ_0, a number d of dynamic edges, a maximum number G of iterations, a base pheromone density local updating ratio and a 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 and an appropriate fitness function for evaluating paths.

STEP 2: Initially set the pheromone distribution function of each variable as the given initial pheromone density τ_0 and the current best solution S_c as empty.

STEP 3: Set the initial generation number g = 1.

STEP 4: Build a complete route for each artificial ant by the following substeps.

> **STEP 4.1**: Set s = 1, where s is used to keep the identity number of the current variable (stage) to be processed in the graph.
>
> **STEP 4.2**: Get the corresponding pheromone distribution function of the s-th variable.
>
> **STEP 4.3**: Produce d dynamic edges according to the method in Section 3.4.
>
> **STEP 4.4**: Select a path from the d dynamic edges according to the pseudo-random proportional rule.
>
> **STEP 4.5**: Initialize the table of influence functions for the s-th variable if the table isn't exist.
>
> **STEP 4.6**: 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 Section 3.3.
>
> **STEP 4.7**: Set $s = s + 1$.
>
> **STEP 4.8**: 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 4.2.

STEP 5: 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 empty (the first generation) or the best solution in the iteration is better than S_c, set S_c as the current best solution.

STEP 6: Find the one with the highest fitness value among the q ants, and get the values of the variables for the best ant.

STEP 7: Generate the corresponding influence functions for the variable values found in STEP 6 and then update the distribution functions (the tables of influence functions) of the variables according to the global updating process mentioned in Section 3.2.

STEP 8: If $g = G$, output S_c; otherwise, $g = g + 1$ and go to STEP 4.

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.

DEACS was compared to some existing approaches including API [7], GA, and CACS [8]. CACS was a special-purpose ACS for resolving mathematical functions. It used some complex designs to approach the optimal solution to a function. The GA approach adopted here was the traditional genetic algorithm and it decoded the solution space as a binary string. API was inspired by a primitive ant's recruitment behavior. The recruitment technique made the population proceed towards the optimum solution. The same parameter settings from the previous experiments were used. The seven test functions are listed in Table 1.

TABLE 1
SEVEN TEST FUNCTIONS FOR COMPARISON

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

All of the above test functions had a minimum value of 0, which was to be found. Thus, getting a function value closer to zero meant a better performance. The experimental results by the different approaches are summarized in Table 2.

From Table 2, it could be observed that DEACS and CACS could obtain satisfactory solutions for these test functions when compared to the others. For Functions 1 and 2, since the functions were simple and the encoding spaces were dispersed in GA and API, GA and API could obtain the minimum value 0 as the solutions. DEACS and CACS could just get the values very close to 0. In the other complex functions, DEACS and CACS could obtain better performances than the other two methods.

TABLE 2
Comparisons of DEACS and The Three Other Methods

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
DEACS	3.1e-7	5.3e-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	7.476	0.004	0.250	0.006	0.093
GA	0.0	0.0	2.124	0.030	0.139	0.073	0.133

5. Conclusion

This work proposed an extended ACS algorithm for solving continuous variables problems. The proposed algorithm is different from the existing ant-based 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 experimental results show that DEACS is very competitive to the existing ACS and some other evolutionary algorithms. In the future, the DEACS algorithm will be applied to more problems in addition to the constrained functions.

Acknowledgment

This research was supported by the National Science Council of the Republic of China under contract NSC 100-2221-E-390-026-MY3.

References

[1] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," *European Conference on Artificial Life*, pp. 134–142, Paris, France. Elsevier, 1991.

[2] M. Dorig and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, 1997.

[3] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of coop- erating agents," *IEEE Transactions on Systems, Man, and Cybernetics Part B*, vol. 26, pp. 29–41, 1996.

[4] W. Jiang, Y. H. Xu, and Y.S. Xu, "A novel data mining algorithm based on ant colony system," *International Conference on Machine Learning and Cybernetics*, vol. 3, pp. 1919– 1923, Guangzhou, 2005.

[5] L. Kuhn, *Ant Colony Optimization for Continuous Space*, Master's Thesis, The Department of Information Technology and Electrical Engineering of The University of Queensland, 2002.

[6] H. Li and S. Xiong, "On ant colony algorithm for solving continuous optimization problem," *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 1450–1453, 2008.

[7] N. Monmarche, G. Venturini, and M. Slimane, "On how pachycondyla apicalis ants suggest a new search algorithm," *Future Generation Computer Systems*, vol. 16, pp. 937-946, Jun. 2000.

[8] S. H. Pourtakdoust and H. Nobahari, "An extension of ant colony system to continuous optimization problems," *Lecture Notes in Computer Science*, vol. 3172, pp. 158–173, 2004.