Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 142194, 7 pages
http://dx.doi.org/10.1155/2014/142194

Research Article

Using the ACS Approach to Solve Continuous Mathematical Problems in Engineering

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

[1] *Department of Computer Science and Engineering, National Sun Yat-sen University, 70 Lienhai Road, Kaohsiung 80424, Taiwan*
[2] *Department of Computer Science and Information Engineering, National University of Kaohsiung, 700 Kaohsiung University Road, Kaohsiung 81148, Taiwan*

Correspondence should be addressed to Chung-Nan Lee; cnlee@cse.nsysu.edu.tw

Received 15 February 2014; Revised 16 May 2014; Accepted 6 June 2014; Published 24 June 2014

Academic Editor: Jianquan Lu

Ant colony system (ACS) has been widely applied for solving discrete domain problems in recent years. In particular, they are efficient and effective in finding nearly optimal solutions to discrete search spaces. Because of the restriction of ant-based algorithms, when the solution space of a problem to be solved is continuous, it is not so appropriate to use the original ACS to solve it. However, engineering mathematics in the real applications are always applied in the continuous domain. This paper thus proposes an extended ACS approach based on binary-coding to provide a standard process for solving problems with continuous variables. It first encodes solution space for continuous domain into a discrete binary-coding space (searching map), and a modified ACS can be applied to find the solution. Each selected edge in a complete path represents a part of a candidate solution. Different from the previous ant-based algorithms for continuous domain, the proposed binary coding ACS (BCACS) could retain the original operators and keep the benefits and characteristics of the traditional ACS. Besides, the proposed approach is easy to implement and could be applied in different kinds of problems in addition to mathematical problems. Several constrained functions are also evaluated to demonstrate the performance of the proposed algorithm.

1. Introduction

Ant colony systems (ACS) have been shown to have good performance in finding near-optimal solutions for NP-hard problems. An ACS adopts distributed computation and uses a constructive greedy strategy [1] with positive feedback to search for solutions. It is a powerful approach inspired by the behavior of ants, which 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 [2–6]. They are also adopted to solve algebraic equations in mathematics [7].

Normally, ant-based algorithms are used in discrete solution space. Thus, they usually resolve the problems which need to find a trip as a solution due to the restriction of the encoding space. This kind of problems such as TSP (travelling salesman problem) provides a searching map to ACS for artificial ants to search for good trips. In other words, if the solution space of a problem can be transformed as a searching map, ACS would then be easily applied to it.

In the past, ACS was seldom used in continuous domain problems since the search space is hard to transfer to trips. Several ant-based methods were then proposed to support continuous solution space [8–12]. However, these methods did not completely utilize the traditional ant process in continuous domains. Thus, they preserved few characteristics of the original ACS algorithm, losing some of its advantages. Hong et al. also designed an ACS algorithm with a specific binary coding to find appropriate membership functions in fuzzy data mining. Their experimental results showed the good performance of ACS for handling continuous values. The paper thus proposes a generalized version and a standard procedure of the ACS algorithm in order to easily apply it to other applications. The proposed method retains the process of the original ACS algorithm to preserve its good characteristics and recommends a simple encoding process for continuous variables. It encodes the continuous variables

into a binary string and easily transforms the problem to be solved into a multistage searching map. After that, each ant selects an edge in each stage with an edge representing a part of the binary string for a continuous variable. Experiments on finding solutions to several constrained functions are also made, and the results show that the proposed approach works well in solving these continuous variables problems.

2. Review of Existing ACS Algorithms for Continuous Solution Space

The ant system was first introduced by Chang and Lin and Dorigo et al. [2, 13] and then extended to the ant colony system (ACS) [14]. ACS is based on observations of real ant colonies searching for food according to the pheromone density to solve complex problems. It uses the state transition rule, global updating rule, and local updating rule to choose nodes and change pheromone density. The state transition rule is used by an ant to probabilistically decide its next state according to the pheromone on the edge and the inverse of the distance (visibility). After all the ants have completed their tours, the pheromone density of the best tour passed is then updated by the global updating rule. There are two kinds of global updates. 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. The other kind takes the best tour among the ones passed by the ants in each individual iteration.

Besides, when an ant chooses an edge between nodes j and s, it immediately updates the pheromone density of the edge to avoid local optimum by the local updating rule.

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.

Traditionally, ant-based 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 ACS algorithm is that it is a constructive algorithm, which is different from genetic algorithms (GAs). When ant-based algorithms select an edge, they will refer to those which are selected before. The ants then choose better edges using heuristic functions designed according to the given problem. ACS algorithms seldom produce poor solutions and are thus 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-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 [9–11]. The solution space is defined as a plane (2-dimentional 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.

Pourtakdoust and Nobahari then proposed an extended ACS algorithm to solve the math equation [7]. 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.

Besides, Socha and Dorigo proposed a complicated ACO algorithm for continuous domains [12]. Their method maintained a set of PDF (probability density functions) to record the information of pheromone. It defined some specific operators in order to apply ACO in continuous solution space. Karimi et al. proposed another well-defined continuous ACS algorithm with Tabu search [8]. It could avoid bad regions and guide the population toward a better solution area. Their experimental results showed that the ant-based algorithm could also get good performance in the continuous domains.

Hong et al. proposed two methods based on the traditional ACS with multistages in order to find nearly optimal membership functions in fuzzy mining [15, 16]. Each membership function contains several parameters to adjust, and each parameter is encoded into a multistage code with two binary bits in each stage [15]. Each stage thus contains four candidate nodes which represent $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$, respectively.

A simple example for its searching map is shown in Figure 1. The trip for an ant from the start point to the destination represents a possible complete binary string for a solution. Hong et al. then improved it by hierarchical processing to increase the solution accuracy [3]. In this paper, we proposed the generalized version and a standard procedure of the algorithm in order to apply it in more continuous domain problems. The proposed method in this paper has the following advantages. First, it is easy to implement since it follows the traditional ACS architecture. Second, it does not need to store extra information from traditional ACS; there is no need to design specific data structure. Finally, this algorithm is complied with traditional ACS algorithm, including the operators and the process. It means that it retains all of the characteristics and strong points from the original ACS algorithm. The details of the algorithm are described below.

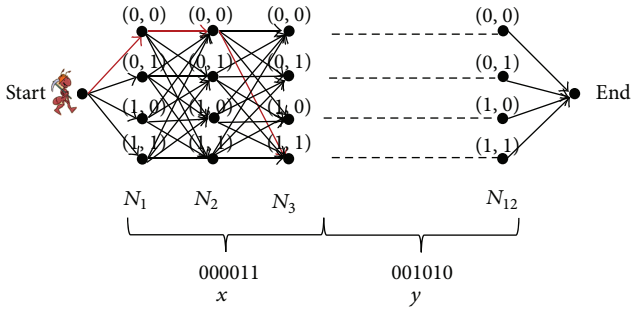

Figure 1: The searching map with 12 stages from previous work.

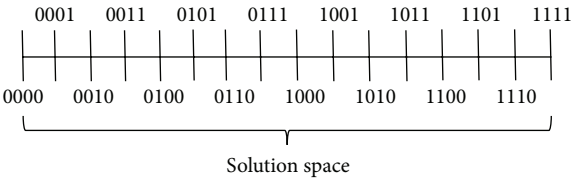

Figure 2: A simple encoding example for a continuous variable with 4 bits.

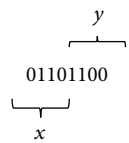

Figure 3: An example of encoding two variables.

3. Generalized Binary-Coding ACS Algorithm

This section describes the proposed generalized ACS-based algorithm for continuous solution space. It is called the binary-coding ant colony system (BCACS) and preserves all of the operators in the original ACS, including the state transition rule, the local updating process, and the global updating process. The details of the proposed algorithm are described below.

3.1. Encoding and Searching Map for BCACS. In the traditional ACS, the resolved problems always provide a map for the population of ants to search. For example, in the TSP problem, the map for salesman to visit is exactly the one for the ACS approach. When an ant has finished a trip in the map, it produces a suitable solution for this problem. However, there is no such map which the ACS algorithm needs in a continuous domain problem. Thus, the proposed generalized ACS approach has to produce a multistage searching map for a continuous domain first in order to resolve this kind of problems. In BCACS, each continuous variable will be encoded into a binary string with length n. If all of the bits in this string are 0, it means that the decoding value for this variable is the minimal value in the solution space. In other words, BCACS decodes the value for a continuous variable by the following formula:

$$V_{\text{decoding}} = V_{\min} + (V_{\max} - V_{\min}) \times \frac{V_{\text{encoding}}}{2^n - 1}, \quad (1)$$

where V_{decoding} is the decoding value of the variable V, $V_{\min}$ is the minimal possible value of V, $V_{\max}$ is the maximal value of V, V_{encoding} is the decimal value of the current binary string of V, and n is the length of V. A simple example is shown in Figure 2.

In Figure 2, a variable is represented by four bits. Assume that the range of the continuous variable is from 0 to 1; then 1101 represents the value of $0 + (1 - 0) * 13/15$, which is 13/15. Thus, a continuous variable is represented by a discrete number of candidates, and the length of its representation will affect the final precision. Assume that a problem includes m variables and the length of each variable is n; then BCACS will encode the whole solution into a binary string with $m \times n$ bits. An example with two variables is shown in Figure 3.

BCACS then generates a multistage searching map in order to obtain a binary string as a possible good result. The length of each variable is further divided into several pieces with the same sublength. Let the sublength be l. The number of nodes in each stage is, thus, 2^l, and a node includes l bits as a substring of a solution. An ant then selects edges for its route to the destination, and all the bits in the nodes passed by the ant are concatenated together to form a solution. The transformed multistage searching map is shown in Figure 4.

By Figure 4, when an ant finishes its route from a starting point to a destination, it also produces an encoding discrete binary string for every continuous variable at the same time. After that, the binary strings can then be decoded to real values in the solution space according to formula (1).

In the ant-based algorithm, the phase number (not the fitness value) of the trip is an important factor to affect the performance. The ants hardly find appropriate solutions if the phase number of the trip is too big. Therefore, setting longer substrings for nodes can reduce the stage number of BCACS. It can also provide more information to the heuristic function because the encoding process can store more data in longer substrings. On the other hand, shorter substrings can help detailed search through pheromone. Thus, appropriate length for substrings should be set to provide a good trade-off between stored information and detailed search, thus guiding the population to move to better solutions.

3.2. Proposed BCACS Algorithm. The proposed ACS is similar to the original ACS. They have similar characteristics and operators such as local update, global update, and edge selection. The details of the proposed BCACS algorithm are described as follows.

3.2.1. The BCACS Algorithm

Input. The input is as follows: a problem (of both continuous and discrete domain) to be solved, a number q of ants, an initial pheromone density τ_0, a node bit number l in each stage, m stages for each continuous variable (it can also work

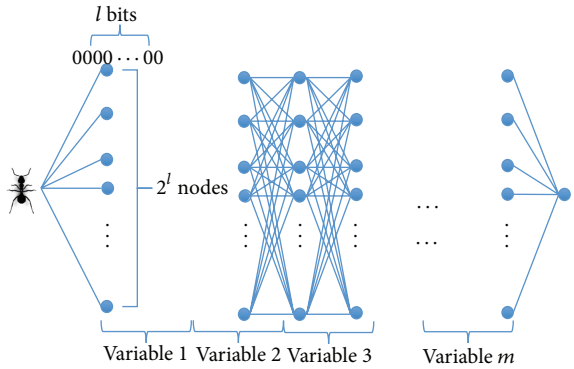

FIGURE 4: A transformed searching map.

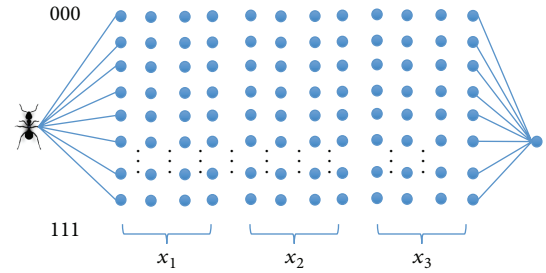

FIGURE 5: A simple searching map for three variables.

for discrete variables), a maximum number G of iterations, the local updating ratio r_l, and the global updating radio r_g.

Output. The output is as follows: a nearly optimal solution to the problem.

Step 1. Generate the multistage searching map for the continuous solution space according to the principle in Section 3.1.

Step 2. Define an appropriate fitness function for evaluating paths passed by ants.

Step 3. Initially set the pheromone density τ_0 on each edge in the map and the current best solution S_c as empty.

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. Put the artificial ant at the starting point on the map.

Step 5.2. Select an edge from the 2^l candidate edges to the next stage according to the pseudorandom proportional rule. In this step, we can design a suitable heuristic function for the problem to guide the population.

Step 5.3. Update the pheromone amount on the selected edge by the traditional local updating process to avoid the population visiting the same edge.

Step 5.4. If the next stage is not the destination, go to Step 5.2.

Step 6. Concatenate the bits on the nodes passed by each artificial ant to construct a solution, decode all the continuous variables, and evaluate the fitness value of the solution 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, replace S_c with the current best solution.

Step 7. Update the pheromone amount on the trip of S_c by the traditional global updating process.

Step 8. If $g < G$, then set the generation number $g = g + 1$ and repeat Steps 5 to 7.

Step 9. Find the one with the highest fitness value amount the q ants, and get the values of the variables for the best ant.

3.3. An Example. In this section, an example is given to illustrate the proposed BCACS for solving a continuous mathematical problem. Assume that there is a formula $x_1^2 + x_2^2 + x_3^2$ with each variable from −5.12 to 5.12. The BCACS is used to find the minimum value for the formula. The execution process is derived as follows.

Step 1. The multistage searching map for the three variables is first generated (Figure 5). Assume that the length of each substring is set as 3 and the coding length for a variable is 12. Each variable thus consists of four phases. The searching map is shown in Figure 6.

Step 2. The formula $x_1^2 + x_2^2 + x_3^2$ is set as the fitness function for each artificial ant because it is to be minimized.

Step 3. The pheromone density τ_0 is initially set as 1.0 and the current best solution S_c is set as empty.

Step 4. The initial generation number g is set as 1 and the maximum number G of iterations is assumed to be 100.

Step 5. In this step, the ants construct their trips with 12 stages (4×3) through the node selection process. An ant selects a node in each stage and each node represents a 3-bit substring in a whole encoding string. For example, a selected trip (encoded string) may be {100 000 000 010 011 111 111 100 011 111 110 111}. In addition, BCACS will perform local updating process after selecting a node.

Step 6. BCACS will decode the encoded strings to get the values of the three variables. Take the encoded string

Table 1: Comparison of BCACS and the three other methods.

Method	$\min(f_1)$	$\min(f_2)$	$\min(f_3)$	$\min(f_4)$	$\min(f_5)$	$\min(f_6)$	$\min(f_7)$
BCACS	$4.68e-6$	0.01969	0.00404	$1.17e-6$	$3.13e-5$	0.00741	0.16546
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

in Step 5 as an example. It is {100 000 000 010 011 111 111 100 011 111 110 111}. {100 000 000 010}, {011 111 111 100}, and {011 111 110 111} represent the encoded string of x_1, x_2, and x_3, respectively. According to formula (1), x_1 is calculated first. $V_{\min}$ is −5.12, $V_{\max}$ is 5.12, and V_{encoding} is 2050 ($2050_{10} = 100000000010_2$). Thus, $x_1 = -5.12 + (5.12 - (-5.12)) \times (2050/4095) = 0.00625$. Similarly, BCACS can obtain $x_2 = -0.00875$ and $x_3 = -0.02626$. The fitness value can then be calculated as $(0.00625)^2 + (-0.00875)^2 + (-0.02626)^2$, which is 0.000805. If BCACS gets a better result than that stored in S_c in this iteration, it will replace the original one in S_c.

Step 7. The pheromone amount on the best trip of the current iteration is updated by the global updating process.

Step 8. If $g < 100$, then the generation number g is increased by 1 and Steps 5 to 7 are repeated.

Step 9. After 100 generations, the solution with the highest fitness value can be obtained, and the values of the three variables can be encoded. In the example, x_1 is 0.000012, x_2 is 0.000003, and x_3 is 0.000011, very close to 0.

4. Experimental Results

Experiments were made to show the performance of the proposed BCACS. The experiments were implemented in Java on a personal computer with an Intel Core 2 Quad 6600 CPU and 4 GB of RAM. BCACS was compared to some existing approaches including API [11], GA, and CACS [7]. CACS was a special-purpose ACS for resolving mathematical functions. It used some complex designs to approach the optimal solution to a function. For example, it applied a normal probability distribution function to calculate the content of pheromone and adjusted the parameters by using the current best fitness value. The GA approach adopted here was the traditional genetic algorithm and it decoded the solution space as binary strings. 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 tour previous experiments [15, 16] were used. The seven test functions are listed as follows:

(1) $x_1^2 + x_2^2 + x_3^2$, $x_i \in \{-5.12, 5.12\}$,

(2) $100(x_1^2 - x_2)^2 + (1 - x_1)^2$, $x_i \in \{-2.05, 2.05\}$,

(3) $50 + \sum_{i=1}^{5}(x_i^2 - 10\cos(2\pi x_i))$, $x_i \in \{-5.12, 5.12\}$,

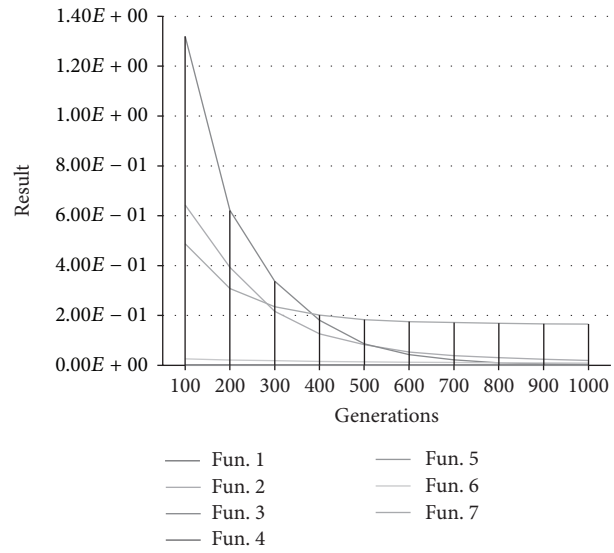

Figure 6: The average results along with the number of generations.

(4) $1 + \sum_{i=1}^{2}(x_i^2/4000) - \prod_{i=1}^{2}\cos(x_i/\sqrt{i})$, $x_i \in \{-5.12, 5.12\}$,

(5) $1 + \sum_{i=1}^{5}(x_i^2/4000) - \prod_{i=1}^{5}\cos(x_i/\sqrt{i})$, $x_i \in \{-5.12, 5.12\}$,

(6) $0.5 + (\sin^2(x_1^2 + x_2^2)^{1/2} - 0.5)/(1 + 0.001(x_1^2 + x_2^2))$, $x_i \in \{-100, 100\}$,

(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. Getting a function value closer to zero thus meant a better performance. The number of candidate edges in each stage was 2 ($l = 1$). The experimental results by the different approaches are summarized in Table 1.

From Table 1, it could be observed that BCACS 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 discrete in GA and API, GA and API could obtain the minimum evaluation value (0) because of the optimal solutions being encoded in the encoding spaces. The encoding space in BCACS was also discrete. It used formula (1) as the decoding function, and there were no special parameters designed to guarantee the optimal solutions in the encoding spaces. Therefore, BCACS could not get the minimum

Table 2: The average results for different node bit numbers (l).

l	Fun. 1	Fun. 2	Fun. 3	Fun. 4	Fun. 5	Fun. 6	Fun. 7
1	$4.68E-06$	0.01969	0.00404	$1.17E-06$	$3.13E-05$	$7.41E-03$	$1.65E-01$
2	$4.68E-06$	0.02514	0.07208	$7.51E-05$	$1.20E-04$	$6.71E-03$	$1.55E-01$
3	$4.90E-06$	0.01913	0.02955	$8.57E-06$	$3.17E-05$	$6.99E-03$	$1.61E-01$
4	$8.84E-06$	0.04885	0.24192	$3.09E-05$	$6.46E-05$	$8.24E-03$	$2.11E-01$

Table 3: The execution time (ms) for different node bit numbers (l) for the seven test functions.

l	Fun. 1	Fun. 2	Fun. 3	Fun. 4	Fun. 5	Fun. 6	Fun. 7	Total
1	191	74	225	112	194	77	82	957
2	83	62	207	85	212	85	98	836
3	137	95	245	96	166	67	72	886
4	139	94	324	138	338	93	99	1241

evaluation value (0) in the first two functions. For the other complex functions, BCACS could obtain good and stable performance. In function (3), only BCACS can get a good result which is near the minimum value 0. CACS had shown its good performance in resolving function optimization problems [7]. In contrast, BCACS was a general continuous ACS not only for the function optimization problems but also for other continuous domain problems in the same process. It not only was thus as flexible as the traditional ACS but could also achieve good performance for problems with continuous variables.

Experiments were then made to evaluate the effects of different node bit numbers (l) when the same encoding length is set on the solutions. The results averaged in 1000 runs for the seven test functions are shown in Table 2.

In Table 2, the results in each column were very similar but the performance for $l = 1$ was the best among the different bit numbers. If l is bigger, more information could be provided to heuristic functions to guide an ant in selecting the next edge and more appropriate solutions could be obtained. But, in the application of computing the minimum value of a function, we did not design any heuristic function to guide the ants to approach better results. Thus, we should choose a smaller number of l in order to do more precise searching. It is very important for BCACS to define an effective heuristic function and set an appropriate l value in real implementation. The execution times for different node bit numbers (l) under the same total encoding length, averaged in 1000 runs, for the seven test functions are shown in Table 3.

If the number of candidate edges was smaller (meaning the bit number was less) and the length of the whole encoded string is fixed, the distance from the start point to the destination would become longer. It would increase the computation time for an ant to finish its trip. However, if the number of candidate edges was bigger, BCACS would spend more computation time in the selection process. In the experiments, BCACS had the smallest computation time when the number was set as 4.

Figure 6 then shows the average results by BCACS along with different generations for the seven test functions with $l = 1$. In the early 300 generations, the populations of BCACS converged quickly to obtain good results. BCACS then shows its ability of seeking better results in the solution space in the next generations. BCACS thus has the same characteristics and benefits (like converging quickly and having good exploring ability) as the traditional ACS because it has the standard process of the traditional ACS.

In the past, we designed a specific version [15] of the proposed algorithm to find appropriate membership functions for the fuzzy data-mining problems. In that paper, the approach was compared with GA and could get good results from the experiments. In this paper, we propose a general version of the algorithm and conduct experiments for mathematical functions. The experimental results showed that it also obtained good enough results in this application.

5. Conclusion and Future Work

In this paper, a simple and standard ACS approach for solving problems in continuous domains is proposed. The algorithm adopts the traditional operators and process of ACS and thus retains its benefits and characteristics. It just encodes a solution space into a binary encoding space and transforms a problem into a map-searching problem. The ACS process can then be easily applied to it. The experimental results show that the proposed BCACS is very competitive to the existing ACS and some other evolutionary algorithms.

Even though the performance of BCACS is good enough and the process is very similar to the traditional ACS algorithm, there are still some issues to be further explored. First, there are more parameters in BCACS than in the traditional ACS. We will thus study how to set up these parameters in the future. Second, we will apply the BCACS algorithm to more applications with continuous solution space. The experimental results in [12] were not directly provided. They set the results from the best algorithm as 1 and from the other algorithms as its multiples, which caused the difficulties in

the comparison of the experimental results. We thus make the detailed experimental comparison as our future work.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] W.-J. Jiang, Y.-H. Xu, and Y.-S. Xu, "A novel data mining algorithm based on ant colony system," in *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC '05)*, vol. 3, pp. 1919–1923, Guangzhou, China, August 2005.

[2] M. S. Chang and H. Y. Lin, "An immunized ant colony system algorithm to solve unequal area facility layout problems using flexible bay structure," in *Proceedings of the Institute of Industrial Engineers Asian Conference*, pp. 9–17, 2013.

[3] L. M. Gambardella, R. Montemanni, and D. Weyland, "Coupling ant colony systems with strong local searches," *European Journal of Operational Research*, vol. 220, no. 3, pp. 831–843, 2012.

[4] A. Madureira, D. Falcao, and I. Pereira, "Ant colony system based approach to single machine scheduling problems: weighted tardiness scheduling problem," in *Proceedings of the 4th World Congress on Nature and Biologically Inspired Computing (NaBIC '12)*, pp. 86–91, November 2012.

[5] O. P. Verma, P. Kumar, M. Hanmandlu, and S. Chhabra, "High dynamic range optimal fuzzy color image enhancement using artificial ant colony system," *Applied Soft Computing*, vol. 12, no. 1, pp. 394–404, 2012.

[6] S. Yan and Y.-L. Shih, "An ant colony system-based hybrid algorithm for an emergency roadway repair time-space network flow problem," *Transportmetrica*, vol. 8, no. 5, pp. 361–386, 2012.

[7] S. H. Pourtakdoust and H. Nobahari, "An extension of ant colony system to continuous optimization problems," in *Ant Colony Optimization and Swarm Intelligence*, vol. 3172 of *Lecture Notes in Computer Science*, pp. 294–301, Springer, Berlin, Germany, 2004.

[8] A. Karimi, H. Nobahari, and P. Siarry, "Continuous ant colony system and tabu search algorithms hybridized for global minimization of continuous multi-minima functions," *Computational Optimization and Applications*, vol. 45, no. 3, Article ID MR2600899, pp. 639–661, 2010.

[9] L. Kuhn, *Ant colony optimization for continuous space [M.S. thesis]*, The Department of Information Technology and Electrical Engineering of The University of Queensland, 2002.

[10] L. Hong and X. Shibo, "On ant colony algorithm for solving continuous optimization problem," in *Proceedings of the 4th International Conference on Intelligent Information Hiding and Multiedia Signal Processing (IIH-MSP '08)*, pp. 1450–1453, August 2008.

[11] N. Monmarché, G. Venturini, and M. Slimane, "On how Pachycondyla apicalis ants suggest a new search algorithm," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 937–946, 2000.

[12] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1155–1173, 2008.

[13] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[14] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[15] T.-P. Hong, Y.-F. Tung, S.-L. Wang, M.-T. Wu, and Y.-L. Wu, "An ACS-based framework for fuzzy data mining," *Expert Systems with Applications*, vol. 36, no. 9, pp. 11844–11852, 2009.

[16] T.-P. Hong, Y.-F. Tung, S.-L. Wang, Y.-L. Wu, and M.-T. Wu, "A multi-level ant-colony mining algorithm for membership functions," *Information Sciences*, vol. 182, no. 1, pp. 3–14, 2012.