BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **15**, No 1

Sofia • 2015

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2015-0001

α-Nearness Ant Colony System with Adaptive Strategies and Performance Analysis

Jinqiu Lv[1], *Xiaoming You*[1], *Sheng Liu*[2]

[1]*College of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai, 201620, China*
[2]*School of Management, Shanghai University of Engineering Science, Shanghai, 201620, China*
Emails: weiganglvjinqiu@outlook.com yxm6301@163.com ls6601@163.com

Abstract: *This paper proposes an improved ant colony system with adaptive strategies, called α-AACS and considers its performance. First of all, we introduce α-nearness based on the minimum 1-tree for the disadvantage of the Ant Colony System* (*ACS*)*, which better reflects the chances of a given link, being a member of an optimal tour. Next, we utilize the adaptive operator to balance the population diversity and the convergence speed and propose other optimizations for ACS. Finally, we present an account of the experiments and the statistic-based analysis, which clearly shows that α-AACS has a better global searching ability in finding the best solutions and better performance in solution variation.*

Keywords: *Ant colony system, α-nearness; minimum 1-tree, lower bound, adaptive strategy.*

1. Introduction

The most important member of prototypical optimization problems is undoubtedly Travelling Salesman Problem (TSP) [1], which has achieved great improvements. Formally, TSP can be represented by asking for the circuit of minimum total weight in a weighted, complete, undirected graph that visits each vertex exactly once and returns to its starting point.

Ant Colony System (ACS) [2] is a novel kind of an intelligent algorithm, which can be applied to TSP in a straightforward way. Besides, it is the successful

heuristic algorithm for TSP. Ants are able to find good solutions for the shortest path between the food source and the ant cave. They communicate via pheromone to mark their trails in variable quantities. Artificial ants imitate the behaviour of ant colonies to some extent. Although the first Ant Colony Optimization (ACO) algorithm, Ant System (proposed by M. Dorigo in 1992, 1996), was found to be inferior with respect to the state-of-the-art algorithms for TSP, it has provided inspiration for a number of extensions that significantly improved performance. These extensions include elitist AS, rank-based AS, MAX–MIN AS and ACS [2]. A great deal of scholars has devoted their researches and efforts in ACO. For example, Ying Zhang and Lijie Li have adopted the Dual Nearest Insertion Procedure to initialize the pheromone, integrated reinforcement learning through computing the low bound by 1-minimum spanning tree, and combined Lin Kerninghan local search [see 2]. G a n g H u et al. [3] have presented the binary ant colony algorithm with controllable search bias which had a good search ability and a high convergence speed. A new directed pheromone for representing the global information of searching is defined by X i a n g p i n g M e n g et al. [4]. In [6] the authors have presented an improved ant colony algorithm based on natural selection, which employed the evolution strategy of survival of the fittest in natural selection to enhance pheromones in paths whose random evolution factor was bigger than the threshold of the evolution drift factor in each process of iteration. Tianjun Liao and Thomas Stutzle proposed UACOR, a unified ACO algorithm for continuous optimization, which allows the usage of automatic algorithm configuration techniques to automatically derive new ACO algorithms [6].

This study presents a new ant colony optimization (α-AACS) which is incorporated with 3-opt local search to improve the solution quality. The paper is organized as follows. Section 2 provides a description of the proposed algorithm with a new technique. Section 3 reports experimental comparisons and Section 4 comes to a conclusion.

2. Algorithm description

2.1. α-AACS framework

The framework of α-AACS is given as follows in a pseudo code.
Input: A TSP data
Set parameters;
Initialize Pheromones trails;
Compute a lower bound ;
while (termination condition not met) **do**
  Compute Heuristic information by the minimum 1-tree
Construct the solution by adaptive strategies
  Local search by 3-opt
  Update pheromones
end

2.2. Computing the heuristic information by the minimum 1-tree

In the original ACS, when building a tour, ant k at the current position of city i chooses the next city j to move according to the so-called pseudorandom proportional rule, given by [2]

$$j = \begin{cases} \arg\max \left\{ \tau_{il} \left[\eta_{il} \right]^{\beta} \right\} l \in N_i^k \text{ if } q \le q_0, \\ J \qquad\qquad\qquad \text{otherwise,} \end{cases} \tag{1}$$

where q is a random variable uniformly distributed in [0, 1], q_0 ($0 \le q_0 \le 1$) is a parameter, and J is a random variable selected according to the probability distribution (with $\alpha = 1$)

$$p_{ij}^k = \frac{\left[\tau_{ij} \right]^{\alpha} \left[\eta_{ij} \right]^{\beta}}{\sum_{l \in N_i^k} \left[\tau_{il} \right]^{\alpha} \left[\eta_{il} \right]^{\beta}} \text{ if } j \in N_i^k, \tag{2}$$

where $\eta_{ij} = 1/d_{ij}$ is a heuristic value that is available apriori, α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and N_i^k is the feasible neighbourhood of ant k when being at city i, that is, the set of cities that ant k has not visited yet.

However, there is a certain risk that the application of this rule may prevent the optimal solution from being found. If an optimal solution contains one link, which is not connected to the several nearest neighbours of its two end cities, then the algorithm will have difficulties in obtaining the optimum. Thus, we introduce the concept of α-nearness [8] which better reflects the chances of a given link being a member of an optimal tour.

Definition 1. A 1-tree for a graph $G = (N, E)$ is a spanning tree on the node set $N\backslash\{1\}$ combined with two edges from E incident to node 1. And a minimum 1-tree is a 1-tree of minimum length.

Definition 2. Let T be a minimum 1-tree of length $L(T)$ and let $T^+(i, j)$ denote a minimum 1-tree required to contain the edge (i, j). Then the α-nearness of an edge (i, j) is defined as the quantity

$$\alpha(i, j) = L(T^+(i, j)) - L(T).$$

Let $\beta(i, j)$ denotes the length of the edge to be removed from the minimum 1-tree when edge (i, j) is added. Thus $\alpha(i, j) = c(i, j) - \beta(i, j)$. Then, as shown in Fig. 1, if (j_1, j_2) is an edge of the minimum 1-tree, i is one of the remaining nodes and j_1 is on that cycle that arises by adding the edge (i, j_2) to the tree, then $\beta(i, j_2)$ may be computed as the maximum of $\beta(i, j_1)$ and $c(j_1, j_2)$.

We use the algorithm as follows to compute α-values. Here, b and "mark" are two one-dimensional auxiliary arrays, where array b corresponds to the β-matrix but only contains β-values for a given node i, that is $b[j] = \beta(i, j)$. Array "mark" is used to indicate that $b[j]$ has been computed for node i. The determination of $b[j]$ is done in two phases. First, $b[j]$ is computed for all nodes j on the path from node i to the root of the tree. These nodes are marked with i. Next, a forward pass is used to compute the remaining b-values. The α-values are available in the inner loop.

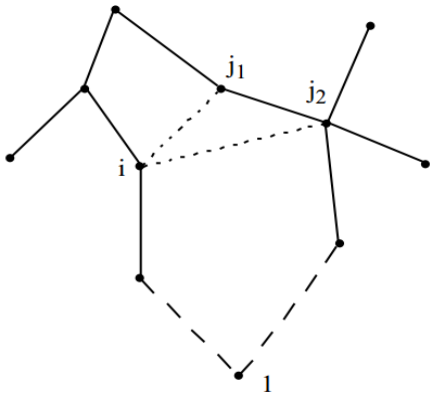

Fig. 1. $\beta(i, j_2)$ may be computed from $\beta(i, j_1)$

```
for i=2:n do
  mark[i]=0;
end do
for i=2:n do
  b[i]=–∞;
  k=i;
   while(k!=2) do
      j=dad[k] //dad[k] denotes the father node of k.
      b[j]=max(b[k], c(k, j));
      mark[j]=i;
      k=j;
   end do
   for j=2:n do
      if j!=i
        if mark[i]!=i
          b[j]=max(b[dad[j]], c(j, dad[j]));
        end if
        α(i, j)=c(i, j)–b[j];
      end if
    end do
end do
```

Next, the procedure of estimating the heuristic value by minimum 1-tree is described in the following steps. Here φ is a positive constant parameter.

Step 1. Compute a minimum spanning tree for $G'=(N\backslash\{1\}, E)$ with the help of Prim's algorithm.

Step 2. Find a minimum 1-tree for G by adding the two shortest edges incident to node 1 to the minimum spanning tree.

Step 3. Compute the nearness $\alpha(i, j)$ for all edges (i, j).

Step 4. $\eta_{ij} = 1/[\alpha(i, j) + \varphi]$.

2.3. Computing of the lower bound to improve α-nearness

In the previous subsection, the α-values provide a good estimate of the edges' probability of belonging to an optimal tour. The computational tests have shown that the α-measure provides a better estimate of the likelihood of an edge being optimal than the usual c-measure [8]. However, the α-measure can be improved substantially by making a simple transformation of the original cost matrix. We use the transformation based on the following equation [9]:

(3) $$d_{ij} = c_{ij} + \pi_i + \pi_j,$$

where the vector $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. The cost matrix $C = (c_{ij})$ is transformed to $D = (d_{ij})$, i.e., an optimal tour for D is also an optimal tour for C. The length of every tour is increased by $2\sum\pi_i$. Let T_π be a minimum 1-tree with respect to D, then its length $L(T_\pi)$ is the lower bound of the length of an optimal tour for D. Therefore, $w(\pi) = L(T_\pi) - 2\sum\pi_i$ is the lower bound on the length of an optimal tour for C. Now the problem is to find a vector $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ which maximizes the lower bound $w(\pi) = L(T_\pi) - 2\sum\pi_i$. When $w(\pi) > w(0)$, the α-values computed from D are better estimates of the edges being optimal than the α-values computed from C.

We use an iterative method of subgradient optimization [10] to maximize $w(\pi)$. The iterative equation is $\pi^{k+1} = \pi^k + t^k(0.7v^k + 0.3v^{k-1})$, where v^k is a subgradient vector in which $v^{-1} = v^0$, and t^k is a positive scalar, called the step size. The subgradient vector is computed by $v^k = d_k - 2$, where d_k is a vector whose elements are the degrees of the nodes in the current minimum 1-tree. This method makes the algorithm strive towards obtaining minimum 1-trees with node degrees equal to 2, i.e., minimum 1-trees transform tours. Fig. 2 shows the steps of a subgradient algorithm for computing the maximum of $w(\pi)$.

1. Let $k = 0, \pi^0 = 0$ and $W = -\infty$.
2. Find a minimum 1-tree, T_π^K.
3. Compute $w(\pi^k) = L(T_\pi^K) - 2\sum\pi_i$.
4. Let $W = \max(W, w(\pi^k))$.
5. Let $v^k = d_k - 2$, where d^k contains the degrees of nodes in T_π^K.
6. If $v^k = 0$ (T_π^K is an optimal tour), or a stop criterion is satisfied, then stop.
7. Choose a step size, $t^k (t^0 = 1)$.
8. Let $\pi^{k+1} = \pi^k + t^k(0.7v^k + 0.3v^{k-1})$, where $v^{-1} = v^0$.
9. Let $k = k + 1$ and go to Step 2.

Fig. 2. Subgradient optimization algorithm

It has been proven [11] that w will always converge to the maximum of $w(\pi)$, if $t^k \to 0$ for $k \to 0$ and $\sum t^k = \infty$.

2.4. An adaptive operator

In (1), the parameter q_0 determines whether the ants to make the best possible move or to explore other paths by roulette selection. In other words, tuning the parameter q_0 allows modulation of the degree of exploration and the choice whether to concentrate on the search of the system around the best-so-far solution or explore other tours. Thus, we use the next equation to calculate the value of q_0 in order to prevent the algorithm from falling into a local optimum:

$$q_0(n+1) = q_0(n) + c, \tag{4}$$

where c is a positive constant parameter and the initial value of q_0 is expressed as $q_0(1)$.

Since q_0 is an incremental value, at the early evolution it is a small value which can increase the diversity of population, while at the later stage of evolution it becomes a greater value in order to accelerate the convergence. The performance of this operator will be demonstrated by the experiments in the next section.

2.5. 3-opt local search

The 3-opt neighbourhood consists of those tours that can be obtained from a tour by replacing at most three of its arcs, making the lengths of the new tours shorter than before. The candidate set is determined by the α-nearness in this paper and letting the length of the candidate set be 5.

The removal of three arcs results in three partial tours that can be recombined into a full tour in four different ways, as shown in Fig. 3.

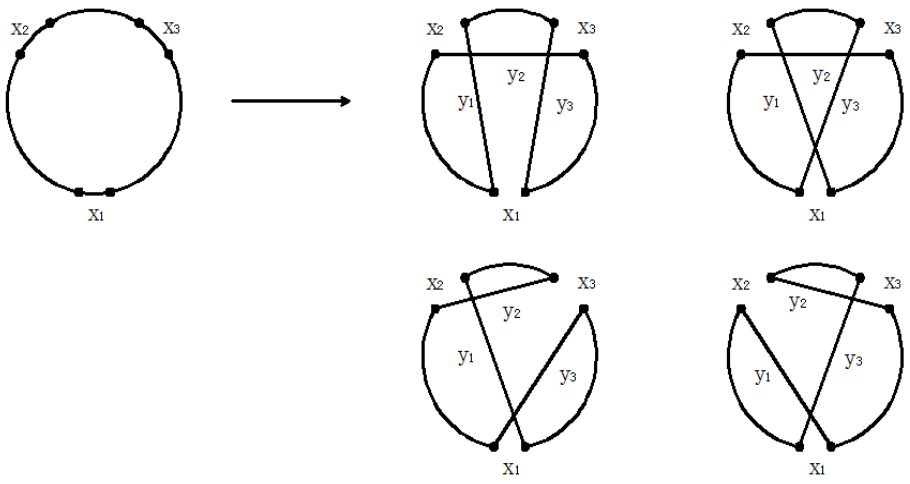

Fig. 3. Four ways of 3-opt

3. Experiments comparisons and statistics-based performance analysis

3.1. Experiments comparisons

To demonstrate the performance of the algorithm proposed, we conduct some computer simulations on a collection of benchmark problems from TSPLIB [12] and compare them with the known optimal solutions of other algorithms. For each benchmark and each algorithm, the experiments are executed 30 times.

Table 1 compares the proposed α-AACS (with 3-opt), ACS with 3-opt and ACS without 3-opt, using various TSP benchmark problems. Table 1 shows that: in Eil51 and Kroa150 problems, α-AACS can obtain the optimal solution; in Kroa100 Kroa200 and Pr264 problems, α-AACS can obtain the solution which is very similar to the optimal solution and the error can be approximated to 0; in the large-scale problems Lin318, the error of α-AACS is 0.37%, which has been reduced by about 7% compared to ACS.

Fig. 4 compares α-AACS (with 3-opt), ACS with 3-opt and ACS without 3-opt using Kroa150 benchmark problem. The solutions generated by α-AACS and ACS with 3-opt are very close, moreover Table 2 compares the time in which both algorithms achieve similar quality solutions by running 500 iterations. Obviously, the time that α-AACS spends is a little more than the time ACS with 3-opt spends. Hence, the advantage of α-AACS is not obvious for small scale problems and it could be even said that it has no advantage.

However, as shown in Fig. 5, the situation is different. It can be observed that α-AACS can obtain the optimum and outperform the two other algorithms. Thus, the results demonstrate the global searching ability of our algorithm in finding the best solutions for large scale problems.

Table 1. Comparison of α-AACS (with 3-opt) with other algorithms

Benchmark problem	Optimum	Algorithm	Near-optimum	Error (%)
Eil51	426	α-AACS+3opt	426.21	0
		ACS+3opt	431.17	1.21
		ACS	438.74	2.99
Kroa100	21282	α-AACS+3opt	21285.44	0.016
		ACS+3opt	21316.37	0.16
		ACS	22384.64	5.18
Kroa150	26524	α-AACS+3opt	26524.86	0
		ACS+3opt	26748.56	0.85
		ACS	28155.86	6.15
Kroa200	29368	α-AACS+3opt	29369.41	0.0048
		ACS+3opt	29834.05	1.59
		ACS	30855.32	5.06
Pr264	49135	α-AACS+3opt	49139.68	0.0095
		ACS+3opt	49729.52	1.21
		ACS	52562.55	6.97
Lin318	42029	α-AACS+3opt	42185.91	0.37
		ACS+3opt	42600.72	1.36
		ACS	45164.35	7.46

Table 2. Time comparison of α-AACS (with 3-opt) and ACS with 3-opt by running 500 iterations

Benchmark problem	α-AACS/s	ACS with 3-opt/s
Eil51	12.657	3.534
Kroa200	53.751	15.456
Lin318	99.125	34.203

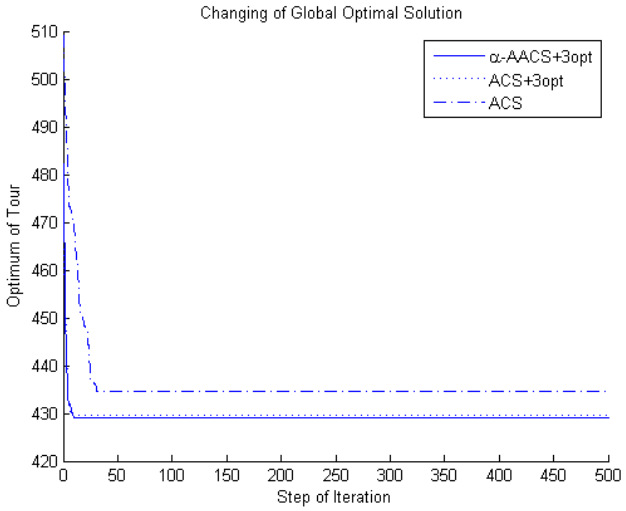

Fig. 4. Comparison of α-AACS (with 3-opt) with other algorithms using Eil51 benchmark problem

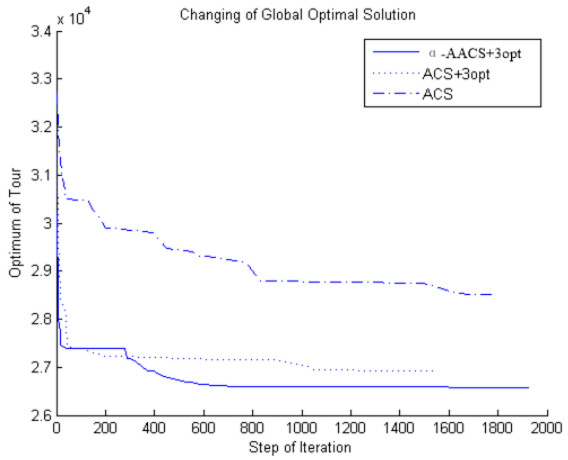

Fig. 5. Comparison of α-AACS (with 3-opt) with other algorithms, using Kroa150 benchmark problem

Table 3 compares the average length of the tours of the algorithm proposed in the paper using various $q_0(1)$ applied to different benchmark problems. As it can be seen, when α-AACS uses the first $q_0(1)$, all the average lengths are better than in the other two situations for three benchmark problems. According to it, setting $q_0(1) = 0.5$ will yield better solutions with respect to the TSP. Moreover, setting $q_0(1) = 0.5$ also has the stability in finding the optimum. Fig. 6 shows an example

of the comparison of α-AACS using different $q_0(1)$ for Kora200 benchmark problem. Hence, the use of $q_0(1) = 0.5$ in our algorithm is recommended to solve the TSP.

Table 3. Comparison of the average tours of α-AACS using different q_0

Benchmark problem	Set1($q_0(1)$=0.5)	Set2($q_0(1)$=0.4)	Set3($q_0(1)$=0.3)
Eil51	429.16	431.25	431.25
Kroa100	21294.73	21342.56	21621.66
Kroa200	29377.54	29834.05	31273.36

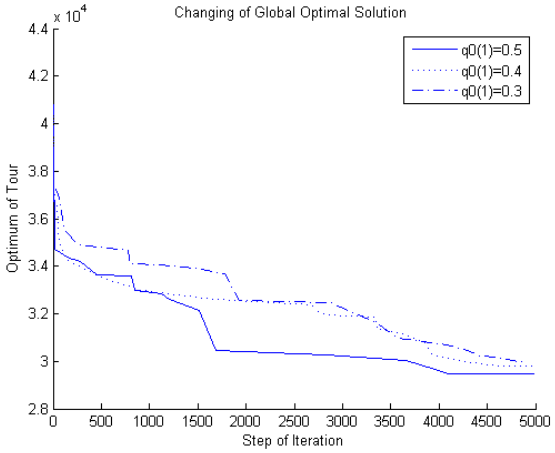

Fig. 6. Comparison of α-AACS (with 3-opt) using different $q_0(1)$ in Kroa200 benchmark problem

3.2. Statistics-based performance analysis

In order to compare the difference between α-AACS (with 3-opt) and ACS+3opt further, we have adopted the statistical index and performed some statistical tests. The statistical index is the standard deviation [13]σ, which is defined as follows:

$$\sigma = \sqrt{\frac{1}{m}\sum_{k=1}^{m}(L_k - \mu)^2}\ , \tag{5}$$

where L_k denotes the tour length generated by the k-th ant after the two algorithms complete each iteration; $\mu$is the mean value calculated by equation (6).

$$\mu = \frac{\sum_{k=1}^{m} L_k}{m}, \tag{6}$$

σ reflects the convergence speed of an algorithm. The less the σ value is, the better the centrality of the solutions generated by all ants is. Fig. 7 shows the convergence tendency of σ of the two algorithms. As it can be seen, the standard deviation σ of α-AACS is always smaller than that of ACS+3opt, whereas σ of ACS+3opt oscillates violently from the beginning to the end, which indicates that the path length generated by α-AACS at each iteration has always better centrality compared with ACS+3opt. After about 50 iterations, the standard deviation σ of α-AACS

becomes small and stable. This is because the value of the parameter q_0 is small at the early stage which makes the ants explore new tours, and with q_0 becoming bigger the ants begin to concentrate on the best-so-far solution.

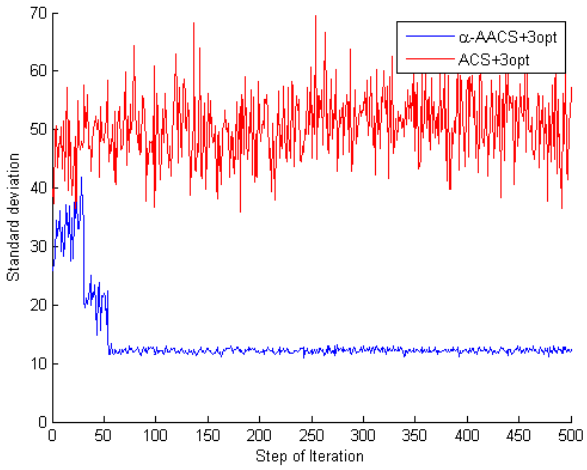

Fig. 7. Comparison of the convergence tendency of the standard deviationσ

4. Conclusion

This paper presents a new ant colony optimization called α-AACS algorithm for solving the TSP. We introduce the minimum 1-tree's concept, a method for computing the lower bound to improve α-nearness and an adaptive operator which can improve the solution quality. The experimental results show that the proposed algorithm can yield a global minimum or a near global minimum to the traveling salesman problem. According to the statistical tests, α-AACS has better performance in solution variation and centrality of solutions. Hence, it is an efficient algorithm for the TSP.

References

1. Karapetyan, D. Lin-Kernighan Heuristic Adaptations for the Generalized Travelling Salesman Problem. – European Journal of Operational Research, Vol. **208**, 2011, pp. 221-232.
2. Zhang, Y., L. Li. MST Ant Colony Optimization with Lin-Kerninghan Local Search for the Traveling Salesman Problem. – ISCID, Vol. **166**, 2008, pp. 344-347.
3. Hu, G., et al. Binary Ant Colony Algorithm with Controllable Search Bias – Control Theory & Applications, Vol. **28**, 2011, No 8, pp. 1071-1080.
4. Meng, Xiangping, et al. Ant Algorithm Based on Direction-Coordinating. – Control and Decision, Vol. **28**, 2013, No 5, pp. 782-786.
5. Wu, Hua-feng, et al. Improved Ant Colony Algorithm Based on Natural Selection Strategy for Solving TSP Problem. – Journal on Communications, Vol. **34**, 2013, No 4, pp. 165-170.
6. Liao, T., T. Stutzle, M. A. Montes de Oca, M. Dorigo. A Unified Ant Colony Optimization Algorithm for Continuous Optimization. – European Journal of Operational Research, Vol. **234**, 2014, pp. 597-609.

7. H e l s g a u n, K. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. – European Journal of Operational Research, Vol. **126**, 2000, No 1, pp. 106-130.
8. H e l d, M., R. M. K a r p. The Traveling-Salesman Problem and Minimum Spanning Trees – Oper. Res., Vol. **18**, 1970, pp. 1138-1162.
9. H e l d, M., R. M. K a r p. The Traveling-Salesman Problem and Minimum Spanning Trees. Part II. – Math. Programming, Vol. **1**, 1971, pp. 16-25.
10. P o l j a k, B. T. A General Method of Solving Extremum Problems. – Soviet Math. Dokl., Vol. **8**, 1967, pp. 593-597.
11. University of Heidelberg. TSPLIB website [EB/OL].
http://www. iwr. Uni-heidelberg. de/groups/comopt/software/TSPLIB95/tsp
12. T a n, G., D. M a m a d y. Real-Time Global Optimal Path Planning of Mobile Robots Based on Modified Ant System Algorithm [C]. ICNC'2006, pp. 204-214.
13. S t u t z l e, T., H. H o o s. MAX-MIN Ant System and Local Search for the Traveling Problem – In: Proc. of IEEE International Conference on Evolutionary Computation, 1997, pp. 309-315.