Swarm and Evolutionary Computation 32 (2017) 140–149

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Effective heuristics for ant colony optimization to handle large-scale problems

Hassan Ismkhan

Department of Computer Engineering, University of Bonab, Bonab, East Azerbaijan, Iran

ARTICLE INFO

Article history:
Received 23 May 2015
Received in revised form
24 May 2016
Accepted 20 June 2016
Available online 22 June 2016

Keywords:
Large-scale optimization
Ant colony optimization
ACO
Heuristics
Traveling salesman problem

ABSTRACT

Although ant colony optimization (ACO) has successfully been applied to a wide range of optimization problems, its high time- and space-complexity prevent it to be applied to the large-scale instances. Furthermore, local search, used in ACO to increase its performance, is applied without using heuristic information stored in pheromone values. To overcome these problems, this paper proposes new strategies including effective representation and heuristics, which speed up ACO and enable it to be applied to large-scale instances. Results show that in performed experiments, proposed ACO has better performance than other versions in terms of accuracy and speed.

1. Introduction

ACO algorithms have successfully been applied to many optimization problems. These problems include combinatorial optimizations such as versions of scheduling problems [1,2], finding edge-disjoint paths problem [3], types of quadratic assignment problem [4,5], QoS-based service selection problems [6], types of vehicle routing problem [7–10], types of traveling salesman problem [11–16] and knapsack problem [17,18]. The ACO has also applications in mobile networks [19], community mining in social networks [20,21], solving clustering problems [22,23], finger print matching [24] and many other problems [25,26]. ACO algorithms also have successfully been applied to continuous optimization problems. A previous work [27] proposes a type of ACO that is applied to continues function optimization by archiving solutions and utilizing a type of local search algorithm in its baseline algorithm. Paper [28] suggests a niche ACO-based on the fitness sharing principle to solve continuous optimization problems. Continuous optimization problem have been solved by types of ACO in two other works [29,30].

Ant algorithms are easy to implement and cover wide range of applications [31], but their performance dramatically decreases in dealing with large-scale problems. The proposed algorithm in [32] is a hybrid method, which combines ACO with swarm algorithm and a local search. The largest instance in the experiments of this paper is Kroa200 with size of 200 (as number of nodes). Reference [33] proposes a version of ACO based on memetic algorithm. The size of largest instance used in the experiments of this paper is also 200. The largest instance used in [34], which is based on combing ACO with a type of gradient search, is also KroA200. The sizes of the largest instances in [35–41], which propose versions of ACO, is 250 or even less. The word "large-scale" is in title of [41], however, similar to [42,43], it has not been applied to instances with size of larger than 1400. The size of the largest instance in [44], which proposes an algorithm based on combination of ACO with Artificial Bee Colony (ABC) algorithm, is 724. The parallel algorithm proposed in [45] tries to overcome the premature convergence, which is one of the major problems of ACO and leads to trap in local optimum. The largest instance considered in this paper is pr2392 with size of 2392. Where in almost all references in this paper, the versions of ACO have been applied to instances with the maximum size of less than 2300. The proposed parallel strategy in [46] has been applied to Pcb3038, which is a TSP instance with size of 3038.

Some drawbacks of current versions of ACO are as follows: (1) Space complexity of holding pheromone in main memory is high and the most famous current solution to solve this problem is using candidate pheromone values which is too inflexible in many situations. (2) Current versions of ACO use the local search algorithms in their baseline algorithm, but they are not able to use pheromone information to operate more efficiently. (3) The time complexity of selecting next move is high. As the case 1, the current available solution to solve this problem is using candidates-sets, which reduces performance of ACO. This paper, states these

E-mail addresses: H.Ismkhan@bonabu.ac.ir, Esmkhan@gmail.com

http://dx.doi.org/10.1016/j.swevo.2016.06.006
2210-6502/© 2016 Elsevier B.V. All rights reserved.

drawbacks with more details. This paper also suggests some new and effective strategies to eliminate these drawbacks. These strategies increase performance of ACO and enable it to apply to large-scale problems.

Therefore, the rest of this paper has been organized as follows: This paper reviews ACO in Section 2 and it points out some ACO drawbacks in Section 3. Section 4 proposes and designs some new strategies to eliminate current drawbacks of ACO algorithms. Section 5 presents experimental results and finally Section 6 summarizes this paper.

2. Related works

ACO has been inspired by the behavior of real ants that enables them to find the shortest paths between food sources and their nest. In the process of finding the shortest path, the ants are guided by exploiting pheromone, which is deposited on the ground by other ants.

Earlier versions of ACO were applied to traveling salesman problem (TSP) [1]. To solve TSP by applying the ACO, some agents (ants) locate on different positions begin to build a solution. Each ant selects next move according to heuristic information and pheromone values. This is called *transition rule*. After all agents completed their solutions, pheromone values are updated. This process is repeated until some condition is met. A simple version of ACO can be implemented according to pseudo-codes in Algorithm 1.

Algorithm 1. ACO general pseudo codes.

```
1)  while (!Some condition)
2)      for $k = 1$ to $N( =$ number of nodes)
3)          while ($solution_k$ is incomplete)
4)                  $agent_k$ selects $k$'th move;
5)          end while
6)          update pheromone globally;
7)      end for
8)  end while
```

Ant system (AS), the earlier version of ACO, was designed to solve TSP. TSP consists of finding the minimum Hamilton tour in weighted graph. In AS, transition rule is defined as (1):

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]^\alpha . [\eta(r,s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r,u)]^\alpha . [\eta(r,s)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Where $p_k(r, s)$ is probability of choosing node s after node r by ant k, $J_k(r)$ is the set of unvisited nodes of ant k, $\eta(r, s)=1/distance(r, s)$ and $\tau(r, u)$ is pheromone amount of ru. Parameters β and α show the importance of edge weight and pheromone amount respectively. After all ants tours are completed, pheromone update rule is executed as (2).

$$\tau(r,s) \leftarrow (1-\rho). \tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k(r,s) \tag{2}$$

Where $0<\rho<1$ is a parameter, $\Delta\tau_k(r, s)= \begin{cases} \frac{1}{\text{tour length of ant } k}, & \text{if edge}(r,s) \text{is in ant } k \text{ tour} \\ 0, & \& \text{ otherwise} \end{cases}$, and m is the number of ants.

[1] There are two types of TSP including symmetric TSP and asymmetric TSP. In this paper, "TSP" refers to symmetric TSP.

In AS, after some iterations, some pheromone values increase with too high rate, then AS traps in local optima. To solve these problems, Max-Min ant system (MMAS) [47] limits pheromone values between τ_{min} and τ_{max} (τ_{min} and τ_{max} are parameters) and updates pheromone belonging to the global best tour only.

Real ants deposit pheromone when they are moving from one location to another. To simulate this behavior, in Ant Colony System (ACS) [12], *local updating rule* is defined as (3).

$$\tau(r,s) \leftarrow (1-\rho). \tau(r,s) + \rho. \tau_0 \tag{3}$$

Where $0<\rho<1$ is a parameter and τ_0 is an initial value of pheromones. In addition, transition rule and global updating rule are performed by Eqs. (4) and (5) respectively.

$$S = \begin{cases} \underset{u \in J_k(r)}{\text{argmax}} \left\{ [\tau(r,u)]. [\eta(r,u)]^\beta \right\}, & \text{if } q \le q_0 \\ \text{use}(1), & \text{otherwise} \end{cases} \tag{4}$$

$$\tau(r,s) \leftarrow (1-\alpha). \tau(r,s) + \alpha. (\textit{the length of global best tour})^{-1} \tag{5}$$

Where q is a random number uniformly distributed in [0 … 1] and q_0 $(0 \le q_0 \le 1)$ is a parameter. For Eq. (5), $0<\alpha<1$ is a parameter.

There are some research suggesting self-organizing ACO that adapts parameters while it is running [48,49]. Paper [50] suggests three bounds for pheromone trails and uses occasional pheromone contractions, instead of regular evaporations. Paper [51] proposes two-stage updating pheromone ACO and tries to enhance performance of ACO by defining two-level pheromone update rule where increasing search capability is the aim of the first stage and ACO convergence speed-up is goal of second stage.

The choices for an ant to select next node in final steps are less than primary steps [52]. An ant in primary steps can freely choose next node because there are many nodes that have not been visited yet, but in final steps, there are just a few nodes that have not been visited yet, so a bad choice in primary parts of a solution may cause successive bad selections and finally bad solution. To solve this problem, reference [52] supposes two strategies that both of them have been based on changing *local updating rule*. In their algorithms, some parameters like number of visited nodes and length of recent completed path have been participated in *local updating rule*.

Ant colony population size is another challenge in ACO algorithms. When the stopping criteria is defined by a fixed number of iterations, increasing ant colony population increases algorithm time complexity, in other hand small population size of ant colony leads to low quality solutions. To solve this problem, reference [53] decreases colony population when it is running. Although this idea may be effective, how to control decreasing colony population is another challenge.

In some research, ACO takes advantage of other meta-heuristics. Combining ACO with GA is very usual. In paper [54] ant operates as crossover operator and constructs solution according to two solutions taken from population. Paper [11] uses Inver-over operator, one of famous operators in evolutionary algorithm (EA). Reference [5] combines GA with ACO in two separate phases.

ACO also can solve continuous optimization problem. Pheromone representation is a challenge when ACO is applied to continuous optimization problem [29,30].

To speed up ACO, researchers suggest some versions [55–58]. In order to speedup ACO, paper [58] limits the length of solutions, which are built by ants. In addition, in order to deal with large graphs, it suggests representing pheromone trails by a type of hash table. Paper [59] suggests interesting parallel ACO based on message passing interface (MPI) but it has critical problem in communication and it has to transfer pheromone values between master and slaves workstations at the end of each step. There is another type of parallel ACO by MPI that each workstation executes ACO serially. At the end of each step, they transfer data (may be the best solution so far or only updated data) among themselves [59]. However all MPI based parallel ACO algorithm have bottleneck in communication level. This problem may be solved by using shared memory model [60], however these models have not been applied to TSP instances larger than 2000 nodes. Even, in research-work with title “Large-Scale”, ACO has never been applied to problem with size more than 2500 [6,41].

3. ACO drawbacks

Although many shortcomings of ACO have been found and solved by researchers, there are many shortages in recent ACO versions that decrease its performance and applicability. All reviewed papers are interesting, but why these methods can not be applied to larger instances? Some drawbacks of current ACO algorithm can be introduced as following:

1) For large problem instances, consuming high space of memory is the typical shortage of the current versions of ACO algorithms. ACO algorithms need pheromone values during the run of the algorithm. Unfortunately in many applications, ACO algorithms need $\Omega(n^2)$ space complexity to hold pheromone values in main memory. Usually, pheromones are implemented by two-dimensional arrays that are not satisfactory for large instances. This implementation also leads a problem when we search pheromone value with some conditions. For instance, suppose we want to design a new heuristic that requires maximum pheromone value from each position (Our first strategy which is stated in Section 4, is such a heuristic that needs moves with high level of pheromone value). In this case, for TSP and some other problems, for each position such as i, we have to consider τ_{ij} for all $j\ (\neq i)$ that takes $O(n)$ time complexity for moves from each position, and takes $O(n^2)$ for all positions. Using candidates-set concept is an easy way to avoid the two-dimensional arrays implementation of pheromone values. Although utilizing some choices as candidates for each position may solve space complexity problem, using static choice, can cause local optima trap. In other hand, updating candidates dynamically needs to justify some questions of when and how algorithm updates the candidates. Which moves should be removed from the candidates? Which moves should be added to the candidates? In fact, we should design a new representation and implementation with efficient heuristic.
2) For TSP, in each iteration, each ant should complete its solution in N (= problem size) steps, and in each step, in average, it has to probe $N\,/\,2$ nodes by calling transition rules. It means that each ant completes its solution in $O(n^2)$ time so the total time complexity becomes $O(n^2 \times \textit{number of iterations})$ that is too high for applying the ACO to large instances. To solve this problem, researchers usually use candidates list for each position [12,47]. This simple solution may aid ACO to increase its speed but as it was said above, using fix candidates list can cause local optima trap.
3) Combining ACO algorithms with other meta-heuristics can enhance their performance. In recent years, local searches have been utilized in ACO algorithms to increase quality of solutions. Although this idea is very effective, enabling local searches to utilize stored knowledge in pheromone trails can improve the performance of ACO, even more.

4. New strategies for ACO

In this section, we try to eliminate the stated drawbacks by suggesting three new strategies. These strategies, including new representation, implementation, and designing efficient heuristics, not only help ACO to apply to large-scale instances of TSP, but also increase its performance including speed and produced solutions quality.

4.1. The first strategy, using pheromone information in local search

In previous versions of ACO, local searches did not use pheromone information but it is logical that local search should be able to use existing information when pheromone values are ready to use. We can use pheromone values directly in local searches. For example, if we use 2-opt as local search then we can use *transition rule value* of edges instead of cost, but this approach increases calculation only without any improvement, so we suggest that it is better to use pheromone information in calculating candidates-sets that used by local search. By this approach, local search can use dynamic candidates-set of best possible nodes. Fig. 1 shows a better view about this approach. Based on this approach, after each step, ACO should update candidates according to pheromone information.

To apply this strategy, in each step, candidates-set of each node

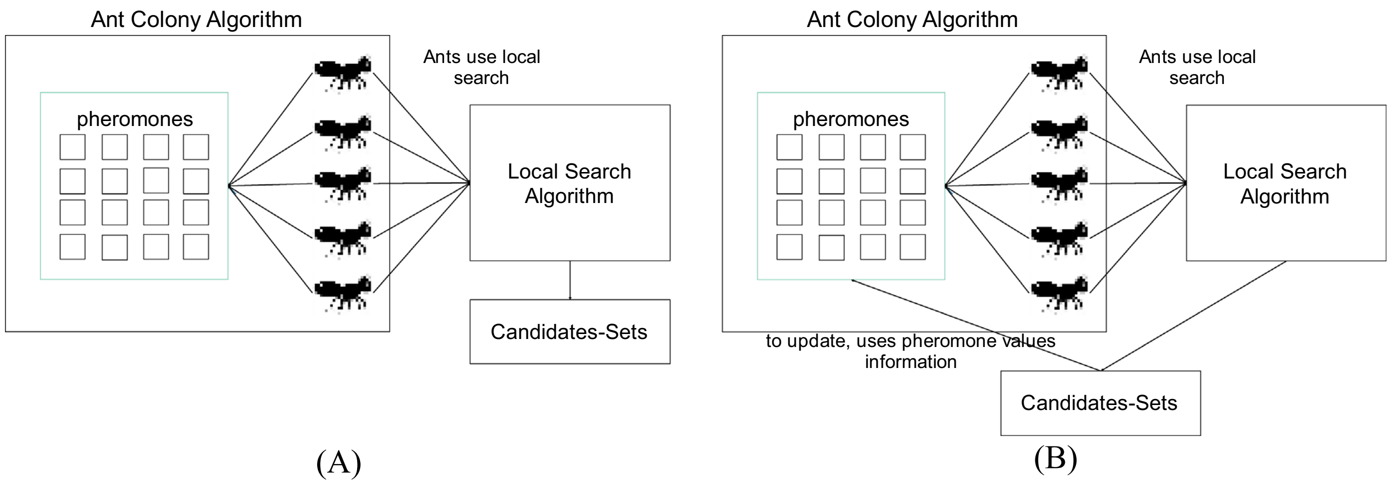

Fig. 1. (A) Shows classical view, in which, local search does not uses pheromone values. (B) Proposed strategy suggests to use pheromone values.

is updated based on Algorithm 2. At the first step, candidates-set of each node is initialized using nearest-neighbors (For a node, its first nearest-neighbor is located in the first place of candidates-set, its second nearest-neighbor is located in the second place of candidates-set and so on). Then after each other step, for a node *n*, its corresponding nodes with maximum value of pheromone is inserted into the first place of candidates-set of *n*. In addition, successor node of *n* in the best tour is inserted into the first place of candidates-set of *n*. Before each insertion, a node in the last place of candidates-set of *n* should be removed (structure of the candidates-set is similar a queue) and repetition is not allowed in candidates-sets.

Algorithm 2. Candidates-set updating algorithm.

```
for each $\underline{n}$ in instance
    $\underline{best} = I$ where $\forall J \neq \underline{n}$ then $\tau[\underline{n}][J] \leq \tau[\underline{n}][I]$ //variable best is argmax { $\tau[\underline{n}][0], \tau[\underline{n}][1], \ldots \tau[\underline{n}][size - 1]$}
    Remove the last node of candidates-set of node $\underline{n}$.
    Insert $\underline{best}$ into the first place of candidates-set of node $\underline{n}$.
    Remove the last node of candidates-set of node $\underline{n}$.
    Insert $\underline{successor\ of\ n}$ in best tour into the first of candidates-set of node $\underline{n}$.
end for
```

4.2. Second strategy, new representation for pheromone values

Previous versions of ACO were holding pheromone information entirely in memory but it may be not possible for large instances. In addition, we cope with a problem when we search pheromone value with some conditions. For example, please assume a problem with four nodes. If we use classical method to holding pheromone values like Fig. 2, to find the maximum (or minimum) corresponding pheromone value from the first node, we have to consider τ_{12}, τ_{13} and τ_{14}. When we have *n* nodes, then the time complexity of $O(n)$ is needed. If we want to find the maximum corresponding pheromone value from each node, then we need time complexity of $O(n^2)$.

To solve these problems, we should attend to the pheromone values. Fortunately, majority of pheromone values are initiated τ_0 and remained fix until the end of algorithm so it can be easily concluded that we are faced with sparse matrix so we represent pheromone values as (6).

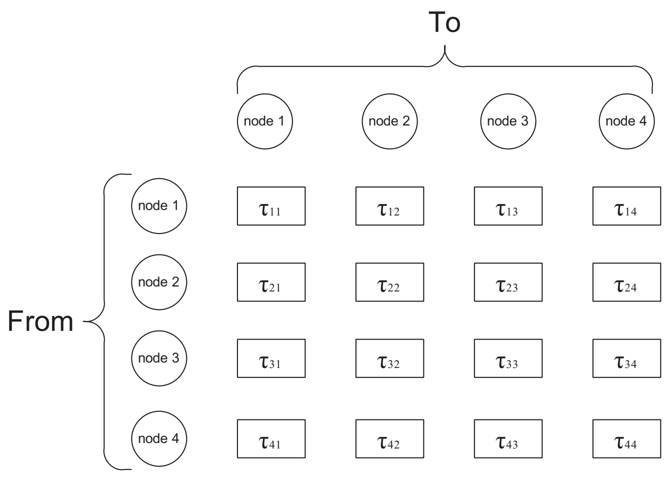

Fig. 2. Holding pheromone values for a problem of size four, using classical method.

some instances and in all of them the number of inserted items to spars matrix were less than $50 \times n$, however to confide, we limit the size of each node list to *Max-Size* (as a parameter). During the algorithm, if for a list of pheromone trails of a node, the number of pheromone trails that have value different from initial state becomes larger than *Max-Size*, then we cut the element with lower pheromone value.

4.3. Third strategy, ACO speedup

Time complexity of selecting next move is a bottleneck of ACO in some applications. In recent research, we try to solve this problem by using genetic crossover operator in body of ACO [54] but we could not extend this approach to large instances. To speed up this process, researchers also use candidate lists as Algorithm 3.

Algorithm 3. Usual algorithm to speed up transition rule.

```
Instructions for $Ant_k$, to select next node of node $r$:
1) Probe Eq. (4) for candidates of $r$ at first and if choosing them is impossible go to next step.
2) Probe Eq. (4) for all not visited nodes.
```

$$pheromones(i,j) = \begin{cases} \tau_0 \left(= \dfrac{1}{cost\ of\ the\ best\ solution\ in\ initial\ phase} \right), & pheromones(i,j)\ has\ not\ been\ updated \\ corresponding\ value\ of\ node\ j\ ini's\ list, & pheromones(i,j)\ has\ been\ updated \end{cases} \tag{6}$$

Fig. 3 shows some parts of pheromone values of burma14, instance from TSPLIB [61], after 300 iterations. For this instance, we have τ_0=1.71 × 10^{-5}. For $\tau(3,4)$ we have $\tau(3,4)$=5.65×10^{-5} because this value has been updated along the algorithm and inserted to spars matrix. For $\tau(1,8)$, we have $\tau(1,8) = \tau_0$=1. 71×10^{-5} because its value has not been updated along algorithm and it has been remained in its initial value. To find the maximum (or minimum) corresponding pheromone value from the first node, we need to consider only two values, instead thirteen.

Space complexity of stated method to implement pheromone shows linear behavior. We probed this method in dealing with

To operate more heuristically, in selection process, we rationalize that if a node, say *j*, has been selected as the next node of *i* in the previous steps so it has high chance to be selected as the next node of *i* in the current step. According to this fact, to select next node of *i*, algorithm should not only probe candidate list of node *i* but also probe all nodes in corresponding list of node *i* in pheromone spars matrix. For all nodes *j* in corresponding list of node *i* in pheromone spars matrix, we have $\tau(i, j) \neq \tau_0$. It means that $\tau(i, j)$ is updated along ACO then it can be concluded that node *j* have been selected as the next node of node *i* in the previous steps and it is logical to consider as the next node of *i* in the current step (Algorithm 4).

Algorithm 4. Proposed algorithm to speed up transition rule.

```
Instructions for Ant$_k$, to select next node of node r:
Hit domain:{    1) Probe Eq. (4) for candidates of $r$ at first and if
                   choosing of them are impossible, go to next step.
                2) Probe Eq. (4) for all nodes in corresponding list of
                   node $r$ in pheromone spars matrix and if choosing
                   of them are impossible, go to the next step.
Fault domain:{  3) Probe Eq. (4) for all not visited nodes.
```

4.4. Proposed ACO

We define ACO as Algorithm 5 according to stated strategies in previous section. We use the proposed new effective strategies in ACO so we name it ESACO (Effective Strategies + ACO). ESACO differs with other ACO versions in three: 1) its pheromone space complexity is linear. 2) It uses pheromone information in its local search (line 11). 3) It uses new method to next node selection (line 5).

Algorithm 5. Suggested ACO (ESACO) to solve large TSP instance.

```
1-  Apply initialization;
2-  for step = 1 to Max-Iterations
3-      for i = 1 to number-of-nodes
4-          for k = 1 to number-of-ants
5-              ant$_k$ selects next edge according to Algorithm 4;
6-              ant$_k$ update corresponding pheromone value of recent selected move by (3);
7-              if i = number-of-nodes
8-                  ant$_k$ improve its tour by 2-opt local search;
9-              end if
10-         end for
11-     end for
12-     update global best tour;
10-     update pheromone globally belong to the global best tour;
11-     update candidates-sets by Algorithm 2;
12- end for
```

5. Experimental results

In this section, we report experimental results of several experiments that we set up to evaluate ESACO performance. We selected TSP as a case study and picked up 22 instances from TSPLIB [61]. We implemented ESACO by C++ language and used. NET 2010; we ran all experiments on 2.0 GHZ Intel CPU with 1.0 G RAM, under Microsoft Windows 7 system.

Subsection 5.1 reports the results of the experiment that had been designed to estimate better possible values for α and ρ. We evaluate performance of each proposed strategy separately. Subsections 5.2 and 5.3 conclude efficiency of second and third strategies respectively. Subsection 5.4 compares ESACO with other versions of ACO. Subsection 5.5 puts forward statistical results for ESACO and finally subsection 5.6 shows efficiency of first strategy.

5.1. Parameters setting

For ESACO, similar to other meta-heuristics, the parameters should be set with appropriate values. For ESACO, number of nearest neighbors (of each node) = 4, *Max-Size* = 10. There is no concern about these parameters, because ESACO dynamically updates its candidates-sets using Algorithm 2. In order to have appropriate run-time for large-scale instances, we set *Max-Iterations*

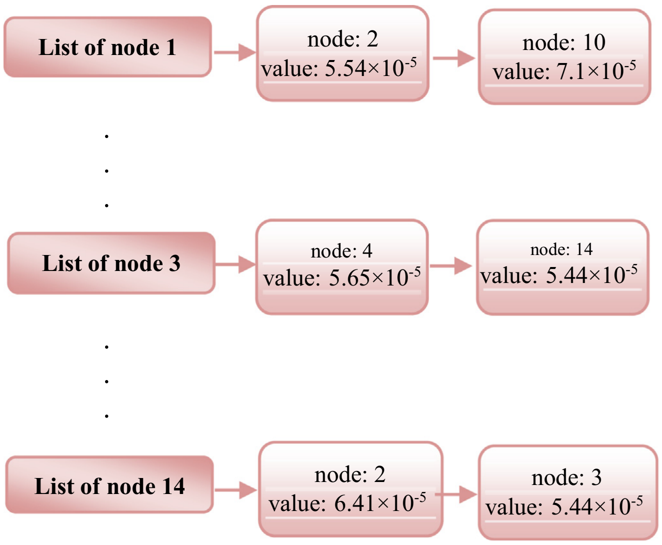

Fig. 3. Some parts of spars matrix for pheromone values of burma14 after 300 iterations.

to 300, and number of ants is set to 10. For greater number of ants, convergence of ACO becomes too time-consuming. Effects of the parameter β on the performance of ESACO are predictable, because high values for β increase the effects of distances between nodes, so it causes to decrease effects of pheromone values. Therefore, we set β to 2 and used an experiment to explore appropriate value for α and ρ. So in this experiment, we explored appropriate values for α and ρ using Algorithm 6. We applied this algorithm to lin318, att532, pr1002 and pr2392. We set $\Delta_1 = \Delta_2$=0.1. In this experiment, we ran this algorithm for 10 times. Figs. 4 and 5 show the average cost and running time of these 10 runs for each corresponding α and ρ by 3-dimensional surface diagram. In all diagrams of both figures, violet color surface shows low cost and running time. According to these diagrams deeply, we can easily found that in all diagrams, the violet surface covers some coordinates specially ($\alpha = 0.9, \rho = 0.6$). According to the results of this experiments we set $\alpha = 0.9$ and $\rho = 0.6$. In addition, after setting α and ρ, we manually consider some values including 1.5, 2.0 and 2.5 for β. Among these values, ESACO had better results with β=2.0.

Algorithm 6. Algorithm to find best α and ρ.

```
for each test instance $I$
    for $\alpha = 0; \alpha \le 1; \alpha = \alpha + \Delta_1$
        for $\rho=0; \rho\le1; \rho = \rho + \Delta_2$
            Set $\alpha$and $\rho$as corresponding parameters of ESACO;
            Solve $I$ by ESACO for a number of iterations;
            Record average cost for current $\alpha$ and $\rho$;
        end for
    end for
    Report $\alpha$ and $\rho$ for with the best average cost;
    Report $\alpha$ and $\rho$ for with the minimum average run-time;
end for
```

5.2. Second strategy analysis

Without using new representation for pheromone values it is not possible to apply ESACO to the large-scale instances. Furthermore the space complexity of the above-mentioned method to implement pheromone values is $O(n)$, while holding whole of pheromone in memory, which is common in the current versions

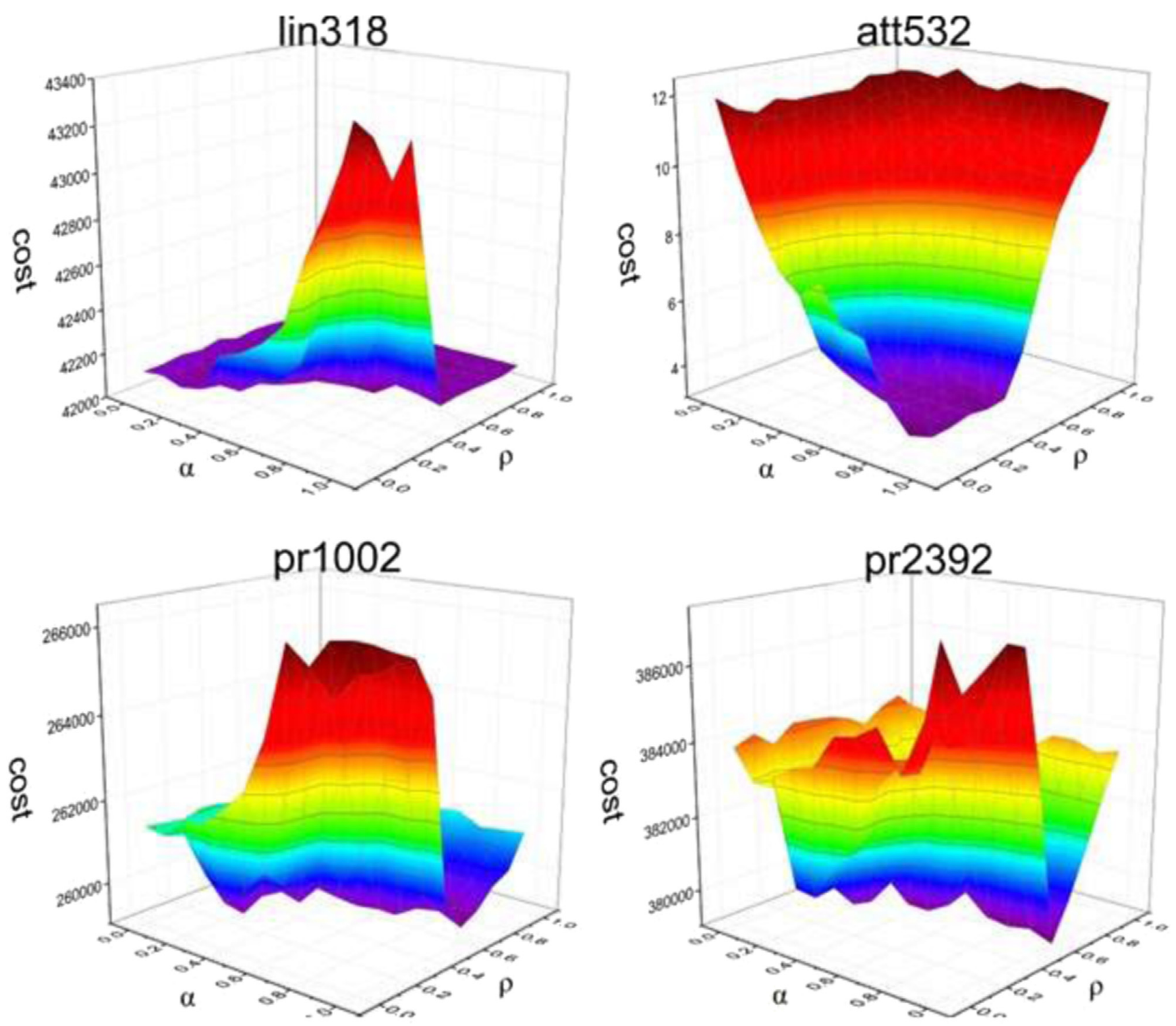

Fig. 4. Average cost for corresponding (α, ρ).

of ACO requires $O(n^2)$ space complexity. To examine the space complexity of new representation of pheromone values, we applied ESACO to some large instances including fnl4461, pla7397, and brd14051, then we observed that for each instance, the number of inserted items into spars matrix was less than $50 \times n$. This space complexity seems to be satisfactory. However, to confide strongly, we limited the size of each node list to *Max-Size* (= 10, as a parameter) it means that we limit space complexity to O (*Max-Size* $\times$ n).

5.3. Third strategy analysis

To analyze the third strategy, we calculated *hit rate* by using Eq. (7) for some instances from TSPLIB including Pr1002, fnl4461, pla7397 and brd14051. For these instances, *hit rate* values were greater than 0.97, which means that in only 3% of calling, all not visited nodes have been probed and in 97%, next node has been selected by probing only two Cl_r and P sets that have been determined in Algorithm 4.

$$\textit{hit domain rate} = \frac{\textit{number of calling next cityfunction in hit domoin}}{\textit{number calling second selection function}} \qquad (7)$$

5.4. Comparing ESACO with Recent ACO Versions

We have compared ESACO with recent ACO versions including TSIACO [51], MMAS [47] and ACS [12]. In this experiment, instances with size of less than 2000 have been considered because the other versions of ACO were not applicable for larger sizes. Table 1 gives some information about the used instances. Each version has been applied to each instance for 30 times. Table 2 shows parameter setting for each version. Table 3 introduces the experimental results including average, standard deviation (STDV) and Wilcoxon signed ranks test result.

The three last columns in Table 3 have been specified for the results of Wilcoxon signed ranks test at confidence level of 5% (if the P-value is less than 0.05, then the difference is significant) between ESACO results and each one of other versions. In these columns the "+" sign show that ESACO is significantly better than other corresponding version of ACO and "~" shows that there is no significant difference between ESACO and other version of ACO. For none of instances, ESACO has been beaten.

For TSP, standard implementation of ACO algorithm has $O(n^2)$ time complexity, increment by the size of instance [62] but in the next sub section, based on performed experiment, we will show that ESACO has very low running time increment by size. We will

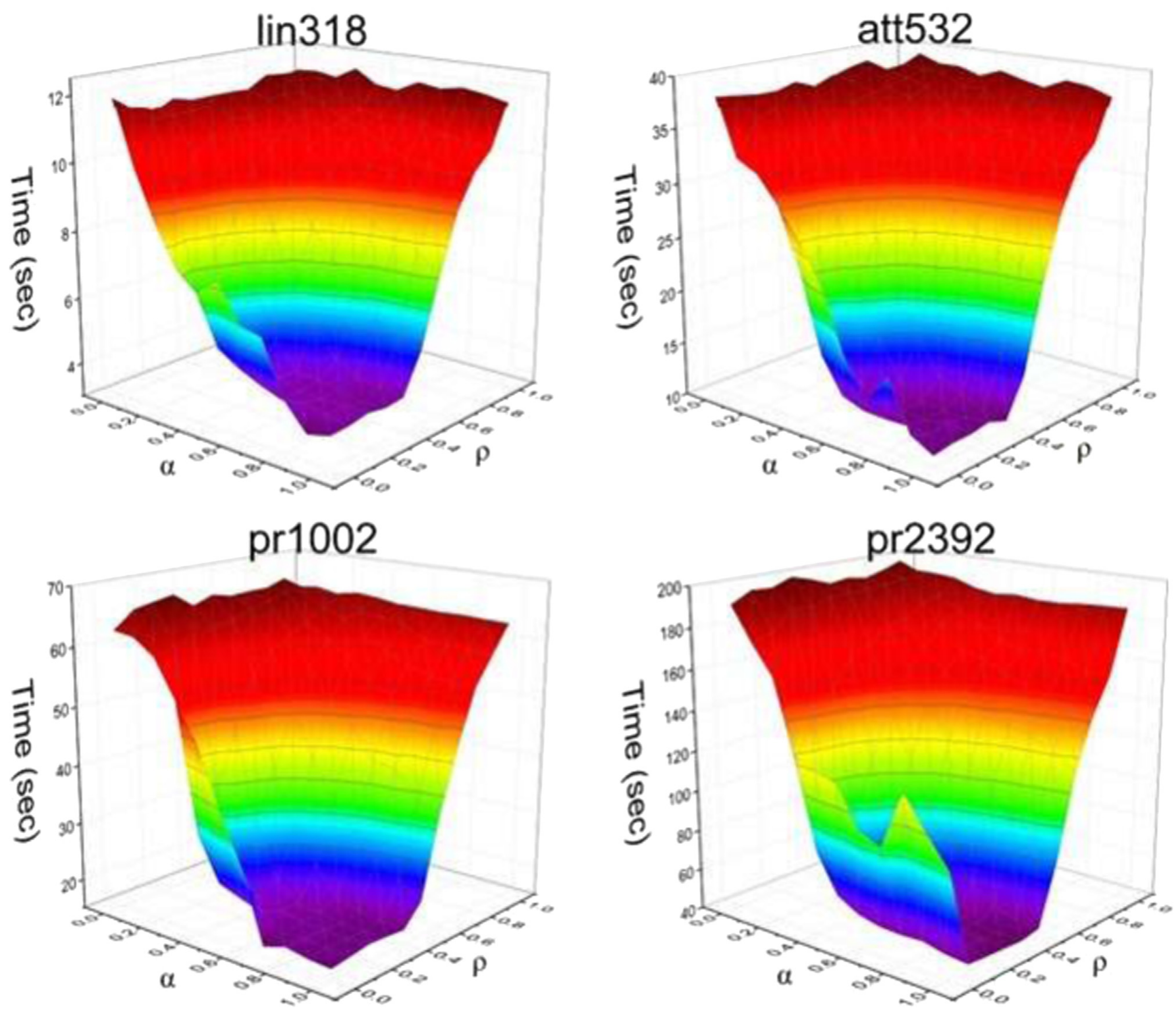

Fig. 5. Average running time for corresponding (α, ρ).

Table 1
12 selected instances form TSPLIB.

Name	Dimension	Optimum Cost
eil51	51	426
eil76	76	538
kroA100	100	21282
lin105	105	14379
d198	198	15780
lin318	318	42029
pcb442	442	50778
att532	532	27686
rat783	783	8806
pr1002	1002	259045
fl1577	1577	22249
pr2392	2392	378032

Table 2
Parameters Setting.

Parameter	ACS	MMAS	TSIACO	ESACO
α	0.1	1.0	1.0	0.9
β	2.0	2.0	2.0	2.0
ρ	0.1	0.5	0.5	0.6
Local search	2-opt	2-opt	2-opt	2-opt
Num-Cndid[a]	4	4	4	4
q_0	0.9			
Number of candidates for transition rule for ACS, MMAS and TSIACO is 20.				
For MMAS, τ_{min}, τ_{max}, τ_0 are set as in [47].				
m[b]	–	–	5	–
cl[b]			0.9	–
r[b]	–	–	5	–

[a] **Num-Cndid: Number of candidates used in 2-opt.**
[b] **These parameters are for TSIACO.**

show the speed of ESACO in dealing with very large instances in the next section. In addition, according to Fig. 6, that introduces the average running time of the considered versions of ACO in performed experiment, ESACO is quicker than the other versions. Diagram in Fig. 6 shows the speed of other versions are not competitive with ESACO for size more than 1000.

In our experiment, Max-Iterations is 300, while MMAS may have better accuracy for long runs. However, it should be noted that based on runtimes shown in Fig. 6, even with this number of iterations, MMAS is slower than other versions. Therefore, at least, we can conclude EASCO is quicker than MMAS.

In our experiments, we also used types of other meta-heuristics such as a type of Particle Swarm Optimization (PSO) proposed in

Table 3
Compared results on instances with size of less than 2000.

	ACS		MMAS		TSIACO		ESACO		Wilcoxon signed ranks test: ESACO vs		
	Average	**STDV**	**Average**	**STDV**	**Average**	**STDV**	**Average**	**STDV**	ACS	MMAS	TSIACO
eil51	426	0	426	0	426	0	426	0	~	~	~
eil76	538	0	538	0	538	0	538	0	~	~	~
kroA100	21282	0	21282	0	21282	0	21282	0	~	~	~
lin105	14411.63	43.62	14379	0	14379	0	14379	0	+	~	~
d198	15983.5	200.67	15882.63	84.4	15832.77	16.85	15780	0	+	+	+
lin318	42219.33	78.04	42278	98.33	42383.17	75.04	42058.2	45.86	+	+	+
pcb442	50906.17	61.78	51011.63	46.54	50973.97	47.73	50792.3	32.65	+	+	+
att532	27759.2	25.13	27815.77	72.48	27819.37	26.31	27698.97	10.09	+	+	+
rat783	8845.37	14.47	8890.3	5.29	8903.03	9.81	8809.77	4.77	+	+	+
pr1002	262150.97	420.5	262331.27	641.81	263334.1	519.06	259560.6	442.08	+	+	+
fl1577	22581.9	83.51	22666.23	283.66	22744.97	128.56	22279.57	32.85	+	+	+
pr2392	384325.83	641.22	384685.97	859.19	387170.03	800.01	379189.23	675.25	+	+	+

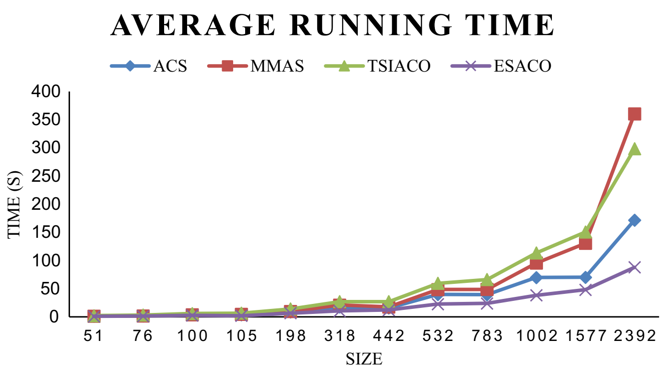

Fig. 6. Average run-times of the versions of ACO.

[63], named Improved Enhanced Self-Tentative PSO (IEST-PSO) and a type of Artificial Bee Colony (ABC) algorithm proposed in [44], which is a Hybrid Ant-Bee Colony (HBC) algorithm. The results of IEST-PSO are not convinced. While we tried to enhance its performance by combining 3-opt local search, the results of IEST-PSO+3-Opt were not convinced yet. While average error of ESACO (which only uses 2-opt local search) for the first five instances are zero, IEST-PSO+3-opt only could gain average zero error for the Eil51 (the first instance). The results of HBC+3-opt were better than IEST-PSO+3-opt, but except for the first two instances (eil51 and eil76), the result obtained by ESACO were better than HBC+3-opt for other remaining instances of this experiment.

5.5. *Large-scale instances and statistical results*

To show applicability of ESACO, we applied it to some large-scale instances. We ran this experiment for 20 times. We have reported all experimental results for ESACO in Table 4. Results in this table show that ESACO also have high accuracy in dealing with large-scale instances. According to "Err" sub-column of "Average" column, it can be easily seen that ESACO could solve all test

Table 4
Results of ESACO for all instances.

Problem name	Optimum	Concorde-LK	Best		Average		Worst		STDV[b]		Running Time (Sec)
			cost	Err[c]	cost	Err	cost	Err	cost	$\mathrm{Err}_{\mathrm{STDV}}$[d]	
eil51	426	**426**	**426**	0.000	**426**	0.000	426	0.000	0.000	0.000	1.11775
eil76	538	**538**	**538**	0.000	**538**	0.000	538	0.000	0.000	0.000	1.3884
kroA100	21282	**21282**	**21282**	0.000	**21282**	0.000	21282	0.000	0.000	0.000	2.6161
lin105	14379	14475	**14379**	0.000	**14379**	0.000	14379	0.000	0.000	0.000	1.9898
d198	15780	15878	**15780**	0.000	**15780**	0.000	15780	0.000	0.000	0.000	6.50835
kroA200	29368	**29368**	**29368**	0.000	**29368**	0.000	29368	0.000	0.000	0.000	4.65505
a280	2579	2594	**2579**	0.000	2579.1	0.000	2581	0.001	0.733	0.00028	4.47645
lin318	42029	43066	**42029**	0.000	42053.9	0.000	42155	0.003	42.834	0.00102	10.1751
pcb442	50778	51636	**50778**	0.000	50803.6	0.000	50954	0.004	64.215	0.00126	11.4644
att532	27686	-*	**27686**	0.000	27701.2	0.000	27712	0.001	8.101	0.00029	23.0849
rat783	8806	8833	**8806**	0.000	8809.75	0.000	8817	0.003	6.883	0.00078	22.5959
pr1002	259045	260759	**259045**	0.000	259509	0.001	260455	0.006	552.107	0.00213	35.8184
fl1577	22249	22870	22255	0.002	22292.8	0.001	22522	0.002	10.559	0.00047	46.439
fl3795	28772	29325	28787	0.001	28883.5	0.003	28994	0.007	51.142	0.00178	119.331
fnl4461	182566	183056	183254	0.004	183446	0.004	183637	0.006	122.266	0.00067	192.621
rl5915	565153	585203	567177	0.005	568935	0.006	570911	0.011	965.671	0.00171	216.926
pla7397	23260728	23633327	23345479	0.004	23389341	0.005	23476015	0.007	21694.114	0.00093	213.929
rl11849	923288	939937	928876	0.009	930338	0.007	932258	0.009	742.573	0.0008	575.806
usa13509	19982859	20043709	20172735	0.012	20195089	0.01	20221268	0.013	16453.184	0.00082	914.212
brd14051	469385	471170	473718	0.011	474087	0.01	474517	0.011	229.824	0.00049	682.49
d15112	1573084	1577204	1587150	0.009	1589288	0.01	1590989	0.012	1153.602	0.00073	776.726
d18512	645238	647471	652516	0.011	653154	0.012	653891	0.014	396.461	0.00061	684.436

* Concorde does not support ATT type.
[b] Standard deviation
[c] Err = (value - optimum) / optimum
[d] $\mathrm{Err}_{\mathrm{STDV}}$ (or STDV Err) = STDV / optimum

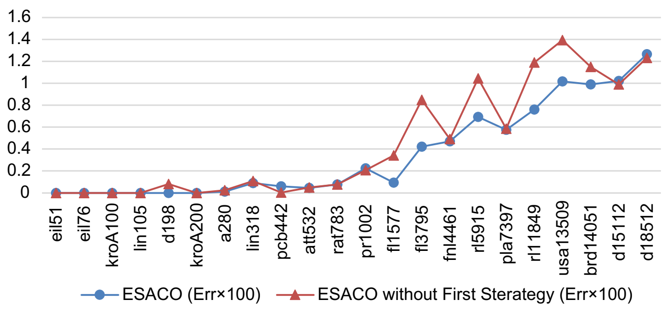

Fig. 7. Err of average cost: In performed experiment, for majority of instances, ESACO has better results than the case without using first strategy.

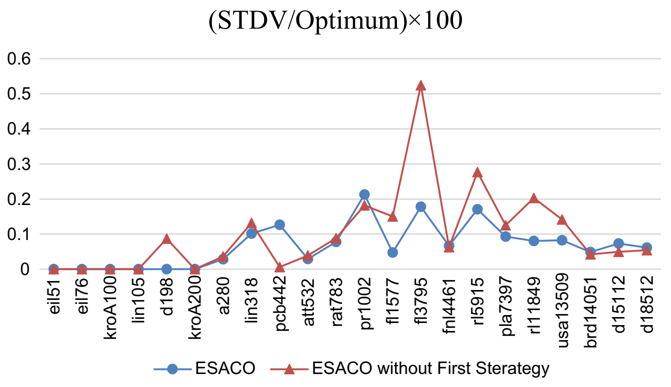

Fig. 8. Err of STDV × 100: This diagram shows that in performed experiments, ESACO is more stable than when it does not use first strategy. For majority of instances ESACO line is under ESACO without first strategy.

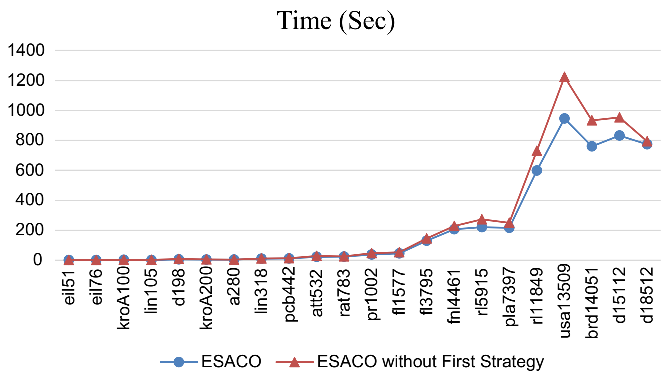

Fig. 9. Runtime of ESACO with and without the first strategy.

instances under 0.02 errors while other ACO versions rarely have been applied to instances with size larger than 1500.

In Table 4, “Err” sub-column of “average” column shows that ESACO solves all instance even very large ones by less than 0.02 errors. Similarly, “Err_{STDV}” sub-column of “STDV” column shows that ESACO is very stable.

The last column in Table 4 shows that ESACO speed is significant; it could solve all instances in less than 950 seconds. By having deep look at this column, another advantage of ESACO may be revealed. To explain in more details, for example, please consider the average running time of rat783 and fl1577; the size of fl1577 is twice greater than rat783 and its running time is about twice.

In addition, in Table 4, results of a run of LK of Concorde TSP solver software version 1.1[2] is included. It can be easily seen that in many cases, worst cost of ESACO is better than results gained by Concorde.

[2] Available at http://www.math.uwaterloo.ca/tsp/concorde/

5.6. *The effects of the first strategy*

To show the effects of the first strategy, we tested ESACO with and without the first strategy. We have introduced all results of this experiment by three diagrams in Figs. 7–9. All these diagrams validate advantage of the first strategy.

6. Conclusion

In this paper, we point out some critical drawbacks of current ACO algorithms, which prevent ACO to apply to large-scale problems. The current implementation of pheromone is inefficient, selecting next move by transition rule is very time consuming, and local search used in ACO operates without effective use of pheromone values. These drawbacks increase ACO time and space complexities, decrease its performance, and do not let it be applied to large-scale instances.

In this paper, we propose some strategies including the new implementation and effective heuristics for suggested ACO (ESACO).

In order to show efficiency of ESACO, we set up multiple experiments using several standard instances of TSP. Experimental results show that the proposed methodologies are effective. In conducted experiments, ESACO achieved better performance than other versions of ACO in terms of speed and accuracy, and when the size of instance is being increased, the ESACO significantly overcomes the other versions.

Results of applying ESACO to large instances show that ESACO is quick, accurate, and applicable to solve very large instances up to 20000 while other versions of ACO rarely have been applied to the instances with size of more than 1500.

Acknowledgment

This work is supported by University of Bonab (contract number: 94/D/AP/543).

References

[1] D. Merkle, M. Middendorf, Ant colony optimization with global pheromone evaluation for scheduling a single machine, Appl. Intell. 18 (1) (2003) 105–111.
[2] C. Blum, M. Sampels, An ant colony optimization algorithm for shop scheduling problems, J. Math. Model. Algorithms 3 (3) (2004) 285–308.
[3] M.J. Blesa, C. Blum, Finding edge-disjoint paths in networks: an ant colony optimization algorithm, J. Math. Model. Algorithm 6 (3) (2007) 361–391.
[4] L.M. Gambardella, É.D. Taillard, M. Dorigo, Ant colonies for the quadratic assignment problem, J. Oper. Res. Soc. 50 (1999) 167–176.
[5] L.Y. Tseng, S.C. Liang, A hybrid metaheuristic for the quadratic assignment problem, Comput. Optim. Appl. 34 (1) (2006) 85–113.
[6] C. Zhang, H. Yin, B. Zhang, A N ovel Ant Colony Optimization Algorithm for Large Scale QoS-Based Service Selection Problem, Discret. Dyn. Nat. Soc. 2013 (2013) 1–9.
[7] C.Y. Lee, Z.J. Lee, S.W. Lin, K.C. Ying, An enhanced ant colony optimization (eaco) applied to capacitated vehicle routing problem, Appl. Intell. 32 (1) (2010) 88–95.
[8] J.J.D.L. Cruz, C.D.P. Arboleda, V. Cantillo, J.R.M. Torres, A two-pheromone trail ant colony system—tabu search approach for the heterogeneous vehicle routing problem with time windows and multiple products, J. Heuristics 19 (2) (2013) 233–252.
[9] M. Reed, A. Yiannakou, R. Evering, An ant colony algorithm for the multi-compartment vehicle routing problem, Appl. Soft Comput. 15 (2014) 169–176.
[10] K.V. Narasimha, E. Kivelevitch, B. Sharma, M. Kumar, An ant colony optimization technique for solving min–max Multi-Depot Vehicle Routing Problem, Swarm Evolut. Comput. 13 (2013) 63–73.
[11] M. Mavrovouniotis, S. Yang, A memetic ant colony optimization algorithm for the dynamic travelling salesman problem, Soft Comput. 15 (7) (2011) 1405–1425.
[12] M. Dorigo, A.L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Trans. Evolut. Comput. 1 (1) (1997) 53–66.
[13] S. Ghafurian, N. Javadian, An ant colony algorithm for solving fixed destination multi-depot multiple, Appl. Soft Comput. 11 (1) (2011) 1256–1262.
[14] E. Lizárraga, O. Castillo, J. Soria, A method to solve the traveling salesman problem using ant colony optimization variants with ant set partitioning, Recent

Advances on Hybrid Intelligent Systems, Springer, Berlin Heidelberg, 2013.
[15] I. Ariyasingha, T. Fernando, Performance analysis of the multi-objective ant colony optimization algorithms for the traveling salesman problem, Swarm Evolut. Comput. 23 (2015) 11–26.
[16] H. Ismkhan, Accelerating the Ant Colony Optimization By Smart ANTs, Using Genetic, CoRR, vol. abs/1411.2897, 2014.
[17] L. Ke, Z. Feng, Z. Ren, X. Wei, An ant colony optimization approach for the multidimensional knapsack problem, J. Heuristics 16 (1) (2011) 65–83.
[18] Z.G. Rena, Z.R. Fengb, A.M. Zhanga, Fusing ant colony optimization with lagrangian relaxation for the multiple-choice multidimensional knapsack problem, Inf. Sci. 182 (1) (2012) 15–29.
[19] G. Singh, N. Kumar, A. Verma, OANTALG: An Orientation Based Ant Colony Algorithm for Mobile Ad Hoc Networks, Wirel. Personal. Commun. 77 (3) (2014) 1859–1884.
[20] L.B. Romdhane, Y. Chaabani, H. Zardi, A robust ant colony optimization-based algorithm for community mining in large scale oriented social graphs, Expert. Syst. Appl. 40 (14) (2013) 5709–5718.
[21] H. Dongxiao, L. Jie, L. Dayou, J. Di and J. Zhengxue, Ant colony optimization for community detection in large-scale complex networks, in Seventh International Conference on Natural Computation, Shanghai, 2011.
[22] A. Chowdhury, S. Das, Automatic shape independent clustering inspired by ant dynamics, Swarm Evolut. Comput. 3 (2012) 33–45.
[23] K.M. Salama, A.A. Freitas, Classification with cluster-based Bayesian multi-nets using Ant Colony Optimisation, Swarm Evolut. Comput. 18 (2014) 54–70.
[24] K. Cao, X. Yang, X. Chen, Y. Zang, J. Liang, J. Tian, A novel ant colony optimization algorithm for large-distorted fingerprint matching, Pattern Recognit. 45 (1) (2012) 151–161.
[25] M. Geetha, G. Nawaz, Fuzzy-ant based dynamic routing on large road networks, in International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME), Salem, Tamilnadu, 2012.
[26] R. Jovanovic, M. Tuba, An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem, Appl. Soft Comput. 11 (8) (2011) 5360–5366.
[27] T. Liao, M.A.M.D. Oca, D. Aydin, T. Stutzle, An incremental ant colony algorithm with local search for continuous optimization, Iridia-Tech. Report. Ser. (2011).
[28] X. Zhang, L. Wang, B. Huang, An improved niche ant colony algorithm for multi-modal function optimization, in international symposium on instrumentation & measurement, sensor network and automation, 2012.
[29] J. Xiao, L.P. Li, A hybrid ant colony optimization for continuous domains, Expert. Syst. Appl. 38 (9) (2011) 11072–11077.
[30] W. Tfaili, P. Siarry, A new charged ant colony algorithm for continuous dynamic optimization, Appl. Math. Comput. 197 (2) (2008) 604–613.
[31] R.J. Mullen, D.N. Monekosso, S.A. Barman, P. Remagnino, Review: a review of ant algorithms, Expert. Syst. Appl. 36 (6) (2009) 9608–9617.
[32] M. Mahi, Ö.K. Baykan, H. Kodaz, A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem, Appl. Soft Comput. 30 (2015) 484–490.
[33] M. Mavrovouniotis, F. M. Müller and S. Yang, An Ant Colony Optimization Based Memetic Algorithm for the Dynamic Travelling Salesman Problem, in the 2015 on Genetic and Evolutionary Computation Conference, 2015.
[34] T. Saenphon, S. Phimoltares, C. Lursinsap, Combining new Fast Opposite Gradient Search with Ant Colony Optimization for solving travelling salesman problem, Eng. Appl. Artif. Intell. 35 (2014) 324–334.
[35] L. Zhou, L. Ding, X. Qiang, A Multi-population Discrete Firefly Algorithm to Solve TSP, Bio-Inspired Computing - Theories and Applications, Springer, Berlin Heidelberg 2014, pp. 648–653.
[36] M. Mavrovouniotis, Y. Shengxiang, Y. Xin, Multi-colony ant algorithms for the dynamic travelling salesman problem, in IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments, 2014.
[37] W. D. Gang, P. X. qiang, G. Hong, Y. Z. Gui, A Hybrid Model for Solving TSP Based on Artifiial Immune and Ant Colony, in International Confrence on Computer Application and System Modeling (ICCASM 2010), Wuhan, 2009.
[38] J. Ouyang, G.-R. Yan, A multi-group ant colony system algorithm for TSP, in Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004.
[39] X. Song, B. Li, H. Yang, Improved Ant Colony Algorithm and its Applications in TSP, in Sixth International Conference on Intelligent Systems Design and Applications, 2006. ISDA''06, Jinan, 2006.
[40] L. Wang, Q. Zhu, An Efficient Approach for Solving TSP: the Rapidly Convergent Ant Colony Algorithm, in Fourth International Conference on Natural Computation, Jinan, 2008.
[41] X. Li, J. Liao, M. Cai, Ant Colony Algorithm for Large Scale TSP, in International Conference on Electrical and Control Engineering (ICECE), Yichang, 2011.
[42] D. Weyland, R. Montemanni, L.M. Gambardella, An Enhanced Ant Colony System for the Probabilistic Traveling Salesman Problem, Bio-Inspired Models of Network, Information, and Computing Systems, Springer International Publishing 2014, pp. 237–249.
[43] C. Qi, An Ant Colony System Hybridized with Randomized Algorithm for TSP, in Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Qingdao, 2007.
[44] A.S. Girsang, C.-W. Tsai, C.-S. Yang, A Hybrid Ant-Bee Colony Optimization for Solving Traveling Salesman Problem with Competitive Agents, Mobile, Ubiquitous, and Intelligent Computing, Springer, Berlin Heidelberg 2014, pp. 643–648.
[45] L. Chen, H.-Y. Sun, S. Wang, A parallel ant colony algorithm on massively parallel processors and its convergence analysis for the travelling salesman problem, Inf. Sci. v199 (2012) 31–42.
[46] A. Uchida, Y. Ito, K. Nakano, Accelerating ant colony optimisation for the travelling salesman problem on the GPU, Int. J. Parallel, Emergent Distrib. Syst. 29 (4) (2014) 401–420.
[47] T. Stützle, H.H. Hoos, Max-min ant system, Future Gener. Comput. Syst. Arch. 16 (9) (2000) 889–914.
[48] L. L. S. Ju, Y. Zhang, Improved ant colony optimization for the traveling salesman problem, in International conference on intelligent computation technology and automation, 2008.
[49] Z. Cai, H. Huang, Ant colony optimization algorithm based on adaptive weight and volatility parameters, in the 2008 second international symposium on intelligent information technology application, 2008.
[50] N. Ivkovic, M. Golub, A New Ant Colony Optimization Algorithm: Three Bound Ant System, Lect. Notes Comput. Sci. 8667 (2014) 280–281.
[51] Z. Zhang, Z. Feng, Two-stage updating pheromone for invariant ant colony optimization algorithm, Expert. Syst. Appl. 39 (1) (2012) 706–712.
[52] H.M. Naimi, N. Taherinejad, New robust and efficient ant colony algorithms: using new interpretation of local updating process, Expert. Syst. Appl. 36 (1) (2009) 481–488.
[53] Z. Wu, N. Zhao, G. Ren, T. Quan, Population declining ant colony optimization algorithm and its applications, Expert. Syst. Appl. 36 (3) (2009) 6276–6281.
[54] H. Ismkhan, K. Zamanifar, Using ants as a genetic crossover operator in gls to solve stsp, in International conference of soft computing and pattern recognition, 2010.
[55] T. Stützle, Parallelization strategies for ant colony optimization, Lect. notes Comput. Sci. 1498 (1998) 722–731.
[56] M. Craus, L. Rudeanu, Parallel framework for ant-like algorithms, in Third International Symposium on/Algorithms, Models and Parallel and Distributed Computing/Third International Workshop on Tools for Parallel Computing on Heterogeneous Networks, 2004.
[57] H. Koshimizu, T. Saito, Parallel ant colony optimizers with local and global ants, in international joint conference on neural networks, 2009.
[58] E. Alba, F. Chicano, ACOhg: Dealing with Huge Graphs, in Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO''07), London, England, United Kingdom, 2007.
[59] X. Jie, L. Caiyun, C. Zhong, A new parallel ant colony optimization algorithm based on message passing interface, in the 2008 ieee pacific-asia workshop on computational intelligence and industrial application, 2009.
[60] M. Pedemonte, S. Nesmachnow, H. Cancela, A survey on parallel ant colony optimization, Appl. Soft Comput. 11 (8) (2011) 5181–5197.
[61] G. Reinelt, Tsplib—a traveling salesman problem library, Orsa J. Comput. 3 (1991) 376–384.
[62] Y. Zhou, Runtime analysis of an ant colony optimization algorithm for tsp instances, IEEE Trans. Evolut. Comput. 13 (5) (2009) 1083–1092.
[63] J.-w. Zhang, W.-j. Si, Improved Enhanced Self-Tentative PSO Algorithm for TSP, in Sixth International Conference on Natural Computation, Yantai, Shandong, 2010.

Hassan Ismkhan He is instructor of Faculty of Engineering, University of Bonab since 2011. His teaching courses include Data Structures, Algorithm Design and Theory of Formal Languages and Automata. His research interests include Evolutionary Algorithms, Clustering Algorithms and Operational Research.