An Algorithm for Finding Shortest Path Tree Using Ant Colony Optimization Metaheuristic

Mariusz Głąbowski, Bartosz Musznicki,
Przemysław Nowak, and Piotr Zwierzykowski

Poznań University of Technology, Faculty of Electronics and Telecommunications
Chair of Communication and Computer Networks, Poznań, Poland
bartosz@musznicki.com, przemyslaw.nowak@inbox.com

Summary. This paper introduces the ShortestPathTreeACO algorithm designed for finding near-optimal and optimal solutions for the shortest path tree problem. The algorithm is based on Ant Colony Optimization metaheuristic, and therefore it is of significant importance to choose proper operation parameters that guarantee the results of required quality. The operation of the algorithm is explained in relation to the pseudocode introduced in the paper. An exemplary execution of the algorithm is depicted and discussed on a step-by-step basis. The experiments carried out within the custom-made framework of the experiment are the source of suggestions concerning the parameter values. The influence of the choice of the number of ants and the pheromone evaporation speed is investigated. The quality of generated solutions is addressed, as well as the issues of execution time.

1 Introduction

This article aims at presenting available possibilities of application of the Ant Colony Optimization (ACO) technique in finding the Shortest Path Tree (SPT) as a novel technique and alternative to the methods based on conventional algorithms for finding the shortest path such as the Dijkstra's or Bellman-Ford algorithm [1]. It should be stressed that the Ant Colony Optimization metaheuristic has been constructed to seek solutions of $\mathcal{NP}$-hard problems [2]. As such, there is thus no guarantee that the most optimum solution will be always found. Therefore, the obtained results may be both optimal (accurate) and approximations that depend on the degree of fitness of the algorithm itself for each individual problem to be solved. Therefore, it is crucial to first analyse a given task and to properly select the operations running parameters to be executed and to perform their optimization. Having carried out many research studies and tests, the authors have eventually

R.S. Choraś (ed.), *Image Processing and Communications Challenges 5,*
Advances in Intelligent Systems and Computing 233,
DOI: 10.1007/978-3-319-01622-1_36,

established and chosen appropriate parameters, methods and ways that prove to be the most effective in solving a given specific problem.

The two following subsections of the article present the base algorithm and define the problem of the shortest path tree. Section 2 provides a presentation of the proposed algorithm, discusses its pseudocode, as well as discusses its operation based on an example of a selected graph. Then, in Section 3, the authors present the results of the simulation study of the operation of the algorithm. The first subsection is focused at the percentage of correct solutions, while the second subsection discusses the duration of operation of the algorithm. Final remarks and conclusions are presented in the summary.

1.1 ShortestPathACO Algorithm

The *ShortestPathACO* algorithm is a method and a set of recommendations to ensure that the Ant Colony Optimization metaheuristic for solving the shortest path problem is properly applied. A detailed introduction to the methodology, discussion on the context and methods for finding paths, as well as a discussion on the classes of parameters and the methods for updating pheromones, are introduced by the authors in [3]. The subsequent paper [4] presents a thorough analysis of the *ShortestPathACO* based strategy to find the shortest path between two nodes.

1.2 Shortest Path Tree Problem

It can be proved [5, 6] that the shortest paths from one vertex of a graph to all of the remaining vertices create a shortest paths tree. A characteristic feature of this tree is the fact that its root is formed from the initial (source) vertex, all of its edges are directed in the direction opposite to the vertex, and each path that can be created from the initial vertex to any other vertex is the shortest path to this vertex. By having the vector $d = \{d_i : i \in \mathcal{N}\}$, whose each element d_i is called *vertex label* and takes on numerical values or is equal to infinity and denotes the shortest distance between a given vertex and the initial vertex, we are in position to create this tree. All labels, however, have to satisfy the following conditions called *Bellman's Equation* [7]:

$$d_s = 0 \tag{1a}$$

and

$$d_j = \min_{(i,j)\in\mathcal{A}} \{d_i + a_{ij}\}, \qquad \forall\, j \neq s \tag{1b}$$

The solution for the problem of finding the shortest path tree in a graph finds most of its recent applications in different demanding applications that are based upon multicast routing in communication networks [8, 9, 10, 11].

2 The Application of the ShortestPathACO Algorithm for Finding the Shortest Path Tree

Using the *ShortestPathACO* algorithm, the *ShortestPathTreeACO* algorithm has been developed – an algorithm that aims at constructing the shortest path tree. In this way, the Single-Source Shortest Paths problem (SSSP), related to finding the shortest paths from a single initial node to all other nodes in a weighted graph [5], is solved at the same time. Thanks to a construction of the shortest path tree it is not necessary to find the shortest paths from the source to all of the nodes one by one, but only to such nodes that have not yet been included in the tree. By taking this kind of approach to a solution of the single-source problem, we obtain full and extensive information on the shortest paths but with a lower number of operational counts and initiations of the *ShortestPathACO* algorithm.

Data: $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ – graph, a – edge cost vector, s – initial node, t – end node, m – the number of ants, α – the parameter that defines the influence of pheromones on the choice of the next node, β – parameter that determines the influence of remaining data on the choice of the next node, ρ – parameter that determines the speed at which evaporation of the pheromone trail occurs; takes on values from the interval $\langle 0, 1\rangle$, τ_0 – initial level of pheromones on the edges, τ_{min} – the minimum acceptable level of pheromones on edges, τ_{max} – maximum acceptable level of pheromones on edges, $iteration\_limit$ – the limit of ShortestPathACO iterations
Result: d – vector of labels, $pred$ – vector of predecessors

```
foreach i ∈ 𝒩 do
    d_i ← +∞;
    pred_i ← 0;
end
d_s ← 0;
remaining ← N − 1;
t ← N;
while remaining > 0 do
    while d_t ≠ +∞ and t > 1 do
        t ← t − 1;
    end
    if t > 1 then
        path, length ←
        ShortestPathACO(𝒢, a, s, t, m, α, β, ρ, τ_0, τ_min, τ_max, iteration_limit);
        current_length ← 0;
        foreach (i, j) ∈ path do
            current_length ← current_length + a_ij;
            if d_j > current_length then
                if d_j = +∞ then
                    remaining ← remaining − 1;
                end
                d_j ← current_length;
                pred_j ← i;
            end
        end
    end
end
```

Algorithm 1. ShortestPathTreeACO Algorithm

2.1 Pseudo-code and a Discussion on the Algorithm

The operation of the *ShortestPathTreeACO* algorithm is based on the iterative implementation of the *ShortestPathACO* algorithm with a variable value of the parameter t. At the beginning, the variables d and *pred* are initiated. The variables are responsible for respectively storing the vector for the labels of vertices (the length of the shortest path from the initial node to a given node) and for memorization of the vector of vertices that have been previously in the path to the initial node. Then, the label of the initial vertex d_s is set to 0, the variable *remaining* to $N-1$ (this is the number of labels of vertices that are still to be calculated), whereas the variable t is set to the last vertex in the graph.

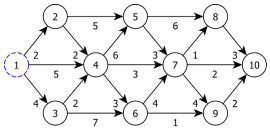
The remaining operations are performed in the loop for as long as there are any vertices whose labels have not yet been ultimately calculated (the variable *remaining* is greater than 0) available. Initially, the label d_t of vertex t is checked whether it is already set and whether t is greater than 1. Then, the variable t is decreased by 1 until both of the conditions are satisfied. If t is still greater than 1, the next action the algorithm has to perform is to initiate the *ShortestPathACO* algorithm for the current value of the variable t, which results in obtaining a calculated *path* between nodes s and t, as well as its *length*.

The next step is to set an auxiliary variable *current_length* for a given path to 0 and to follow thus obtained path to update the variables d and *pred*. For each edge of the path *path*, the length of this edge is added to the variable *current_length*. Later, only when the label of a vertex in which this edge terminates is greater than the variable mentioned earlier further action is performed. If the label of the vertex in which the current edge terminates has not been set earlier, the variable *remaining* is decreased by 1. Then, this label is set to the current value of the variable *current_length*, while the variable *pred* for this vertex is being set to the vertex from which the considered edge starts.

2.2 Illustration of the Operation of the Algorithm

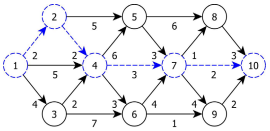
Fig. 1 shows subsequent steps in an exemplary process of building up the shortest path tree with the help of the *ShortestPathTreeACO* algorithm in the hand-made graph constructed from 10 vertices that were joined together by 19 edges. At each of the stages a partial tree is shown, as well as it is shown what values are taken on by particular parameters before the *ShortestPathACO* algorithm for the operative value t is activated. The edges marked with broken line indicate the elements that belong to SPT.

Fig. 1a shows the state of the graph after the initial stage, i.e., after all actions from the line 1-7 of the 1 algorithm have been executed. All labels of the vertices except the initial vertex *1* are not set, which means that there are 9 vertices to be calculated left. The first vertex to be considered is the



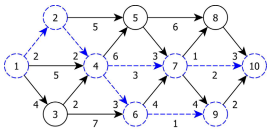
$d = [0\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty\ \infty]$
$pred = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
$remaining = 9$
$t = 10$

(a) Step 1



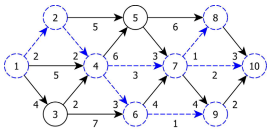
$d = [0\ 2\ \infty\ 4\ \infty\ \infty\ 7\ \infty\ \infty\ 9]$
$pred = [0\ 1\ 0\ 2\ 0\ 0\ 4\ 0\ 0\ 7]$
$remaining = 5$
$t = 9$

(b) Step 2



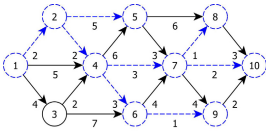
$d = [0\ 2\ \infty\ 4\ \infty\ 7\ 7\ \infty\ 8\ 9]$
$pred = [0\ 1\ 0\ 2\ 0\ 4\ 4\ 0\ 6\ 7]$
$remaining = 3$
$t = 8$

(c) Step 3



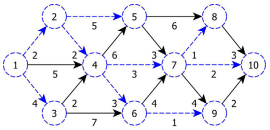
$d = [0\ 2\ \infty\ 4\ \infty\ 7\ 7\ 8\ 8\ 9]$
$pred = [0\ 1\ 0\ 2\ 0\ 4\ 4\ 7\ 6\ 7]$
$remaining = 2$
$t = 5$

(d) Step 4



$d = [0\ 2\ \infty\ 4\ 7\ 7\ 7\ 8\ 8\ 9]$
$pred = [0\ 1\ 0\ 2\ 2\ 4\ 4\ 7\ 6\ 7]$
$remaining = 1$
$t = 3$

(e) Step 5



$d = [0\ 2\ 4\ 4\ 7\ 7\ 7\ 8\ 8\ 9]$
$pred = [0\ 1\ 1\ 2\ 2\ 4\ 4\ 7\ 6\ 7]$
$remaining = 0$
$t = 0$

(f) Step 6

Fig. 1. Consecutive steps in the operation of the *ShortestPathTreeACO* algorithm

vertex *10*. After the execution of the first iteration of the main loop of the algorithm, its state is presented in Fig. 1b. The shortest path to vertex *10* has been found and thus to all vertices that are included in it, i.e., vertices *2*, *4* and *7*. There are now 5 vertices without labels left, while the next end vertex will be vertex *9*. The next iteration complements the list of labels d

by the next 2 values related to the end vertex *9* and vertex *6*, which is shown in Fig. 1c. Now there are only 3 vertices left that still have to have their labels established. The next end vertex is vertex *8*. In the next 3 iterations the paths to vertices *8*, *5* and *3* are found and the elements of the vector of labels d are complemented, while the value of the variable *remaining* is being decreased to 0, which terminates the operation of the algorithm.

In the discussed example, to construct the shortest path tree, and in this way to solve the single-pair shortest path problem, the *ShortestPathACO* algorithm had to be activated 5 times.

3 Study on the Operation of the Algorithm

Exactly as in the study on the general application of the *ShortestPathACO* [3] algorithm, a number of tests was performed to find out whether the solutions given by the *ShortestPathTreeACO* algorithm are correct, as well as to check how the selection of parameters influences the operation of the method in different kinds of graphs. For the discussion in this paper, the hand-constructed graph that is presented in Fig. 1 has been chosen. All the experiments were conducted in a simulation environment prepared using programming language C#. To obtain reliable results, each test was performed 100 times. To diminish the influence of the simulation environment, extreme results were rejected, and then the average values for the remaining results were calculated.

The values of the parameters of the algorithm were the same as during the conducted simulation tests of the *ShortestPathACO* algorithm to find the shortest path between two nodes [4]. Therefore, the values of the parameter α was determined as 1. The value β was adopted here as the one equal to 0.6, since for lower values the algorithm was not capable of finding accurately an optimum solution to the SPT. The parameters τ_0, τ_{min} and τ_{max} were set to 0, 0 and 200, respectively, while the parameter $iteration\_limit$ was set to 25000. It should be reminded that due to the heuristic nature of ACO-based methods, the values of these parameters are extremely important and in the case of different types of graphs these values should be verified.

In the following subsections we discuss the experiments aimed at verifying the influence of the choice of the number of ants m and of the parameter ρ on the quality of generated solutions and on the execution time of the algorithm. The methodology for the calculation of the quality of a solution was modified. In the case of the *ShortestPathACO* algorithm it was equal to the number of optimal lengths of paths divided by the number of tests performed. This time, taking into consideration the fact that a greater number of paths included in a tree was to be analysed, the quality of solution takes additionally into account the number of properly constructed labels of vertices in relation to the number of vertices. This value, taken from all the tests, was then added up and divided by the number of tests, which means that it is the average of

the results of one test. For example, if the algorithm properly calculates the shortest path for 9 out of 10 vertices, then the quality of solution will amount to 90%. If in the second attempt the number of correct paths decreases to 7, then for the two tests the average of the quality of solution will be 80%, with the standard deviation equal to 10%.

3.1 Percentage of Correct Solutions

The percentage of optimal solutions in relation to the parameters m and ρ can be retraced in Graph 2. In the case of the considered graph, it is already with a small number of ants that the percentage of correct solutions ranges within the boundaries of 70%, while the results for different values of ρ are similar. It should not be forgotten, however, that this value means that for 9 of sought-after paths, for nearly 7 of them the found path was the shortest. In addition, the result is also influenced by the fact that the resulting value is the average for 100 measurements. By watching closely the standard deviation we can observe that with a small number of ants it exceeds even 15% for $\rho = 0.001$. This results from a large differentiation in the obtained results between individual measurements and is tantamount to the fact that during some of the tests the quality of solution dropped below 50%, while on other occasions was well above 90%. Along with the increase in the number of ants the percentage number of correct solutions increases, while the standard deviation decreases, which translates into a decrease in the width of confidence intervals in the obtained results. With 14 ants, or more, all paths that have been found are the shortest paths, whereas the standard deviation is equal to 0, which means that in all attempts the results were the same.

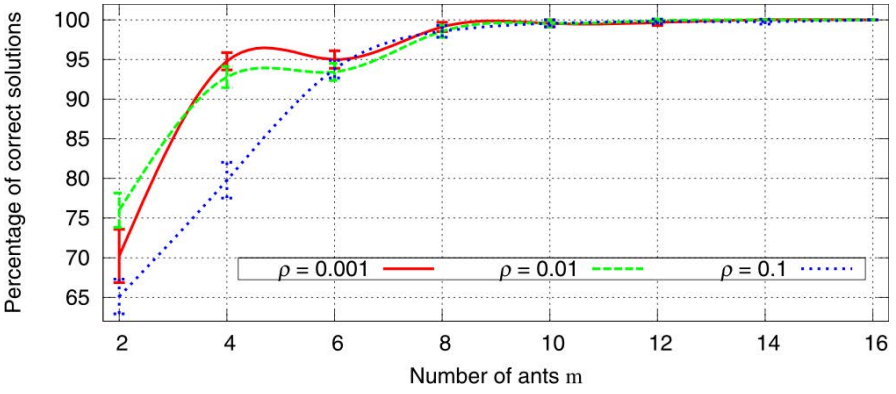


Fig. 2. The quality of generated solutions provided by the *ShortestPathTreeACO* algorithm for the graph from Figure 1 depending on the parameters m and ρ

By analysing the influence of the parameter ρ we can observe that for its higher values, with a small number of ants, the standard deviation takes on higher values, whereas from the number of at least 6 ants the obtained results are remarkably similar for different values of ρ. It is clearly visible that despite the worse results for $\rho = 0.1$ with a small number of ants, after an increase in the number of ants to 6, all three graphs are very similar. With 8 ants, the algorithm approaches 99%, and from the number of 14 ants all found paths are optimal regardless of the number of tests. The data presented in the graph are non-decreasing functions, whereas from a certain threshold, in fact, solid functions.

3.2 Duration of Operation

While analysing the operation times of the algorithm in relation to the parameters m and ρ shown in Graph 3 some interesting dependencies are clearly observable. With a small number of ants, the operation time of the algorithm is very long — considerably longer than that with a substantially higher number of ants. The reason for the above might be the fact that with a small number of ants the time necessary to check as large number of edges as possible prolongs. A colony composed of just 2 ants is not capable of effectively checking available paths, hence the convergence is reached very late. Starting from 4 ants and with $\rho = 0.001$ and $\rho = 0.01$, the operating time of the algorithm stabilizes and increases proportionally to the number of ants. For $\rho = 0.1$, the situation looks quite differently. Despite very similar results for different ρ with 2 ants, from the number of 4 ants, the operation time of the algorithm doubles and drops to an acceptable level with as many as 10 ants. Such results are derived from the fact that the exploration features of

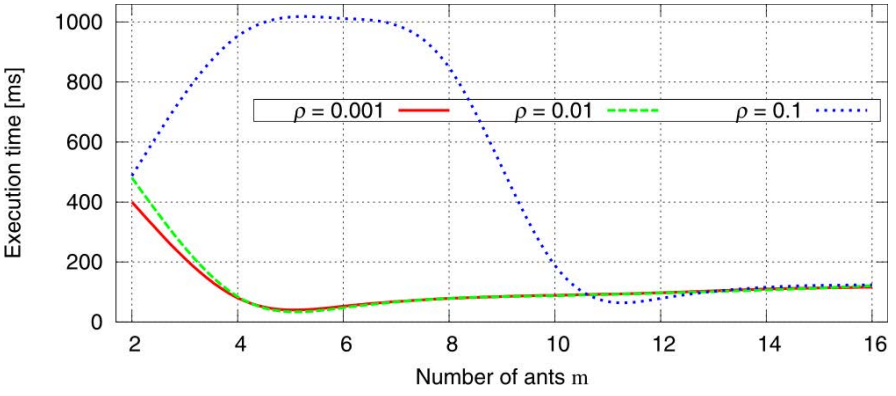


Fig. 3. Execution time of the *ShortestPathTreeACO* algorithm for the graph from Figure 1 depending on the parameters m and ρ

ants become too high. This in turn causes pheromones to be deposited on a great number of edges, while the paths obtained in successive iterations do not repeat, which makes it difficult for the algorithm to reach convergence. From the number of 10 ants, regardless of the value of the parameter ρ, the operation time of the algorithm improves gradually along with the increase in the number of ants.

One can conclude unequivocally from graph 3 that an excessive increase in the value of the value of the parameter ρ may lead to a prolongation of the operation time of the algorithm that ultimately does not improve at all obtained results — as referred to Graph 2.

Taking into consideration both the analysis of the obtained results and the operation time of the algorithm, it is possible to determine that for $\rho = 0.001$ and $\rho = 0.01$ and from the number of about 8 ants, the algorithm performs very well and generates good quality solutions with relatively short time of its operation.

4 Conclusion

The study described in the present article proves that the proposed *ShortestPathTreeACO* algorithm can be successfully used for constructing the shortest path tree based on the method that has been derived from Ant Colony Optimization metaheuristic. It results from the the simulation tests that, for the graph under consideration, it is possible to establish the minimum value of the number of ants that makes the most optimum solutions obtainable. Any further increase would make no sense because it would only translate into an increase in the operation time of the algorithm and a further demand for calculation resources. The number of ants used cannot though be too small because this leads to an increase in time required for the solution to be found. A similar dependence is observable in the case of the parameter that defines the speed of the evaporation of pheromones.

There is a clear need for a careful selection of appropriate parameters for the operation of the algorithm to a specific application, in this particular case to the type of the structure of the graph. This necessity can have its detrimental effect on the possibility of a wide application of the algorithm for any graphs in practice. The initial analysis of the performance of the algorithm in other types of graphs carried out by the authors clearly indicates that in order to obtain acceptable results in more complex and complicated structures, for example in multi-stage graphs, it may be necessary to carry on with additional studies aimed at developing methods for even better improvement in obtainable results.

References

1. Wu, B.Y., Chao, K.-M.: Spanning Trees and Optimization Problems. Chapman & Hall/CRC Press, USA (2004)
2. Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press, Cambridge (2004)
3. Głąbowski, M., Musznicki, B., Nowak, P., Zwierzykowski, P.: Shortest Path Problem Solving Based on Ant Colony Optimization Metaheuristic. International Journal of Image Processing & Communications 17(1-2), 7–17 (2012)
4. Głąbowski, M., Musznicki, B., Nowak, P., Zwierzykowski, P.: ShortestPathACO based strategy to find the Shortest Path between two nodes. In: Proceedings of 2013 IEICE Information and Communication Technology Forum (ICTF), Sarajevo, Bosnia and Herzegovina, pp. 29–31 (May 2013)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge (2009)
6. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
7. Bertsekas, D.P.: Network Optimization: Continuos and Discrete Models. Athena Scientific, Belmont (1998)
8. Maxemchuk, N.F., Shur, D.H.: An Internet multicast system for the stock market. ACM Transactions on Computer Systems 19(3), 384–412 (2001)
9. Pragyansmita, P., Raghavan, S.V.: Survey of Multicast Routing Algorithms and Protocols. In: Proceedings of ICCC 2002, the Fifteenth International Conference on Computer Communication, Washington, DC, USA, pp. 902–926 (2002)
10. Piechowiak, M., Zwierzykowski, P.: The Evaluation of Unconstrained Multicast Routing Algorithms in Ad-Hoc Networks. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2012. CCIS, vol. 291, pp. 344–351. Springer, Heidelberg (2012)
11. Musznicki, B., Tomczak, M., Zwierzykowski, P.: Dijkstra-based Localized Multicast Routing in Wireless Sensor Networks. In: Proceedings of CSNDSP 2012, 8th IEEE, IET International Symposium on Communication Systems, Networks and Digital Signal Processing, Poznań, Poland, pp. 18–20 (July 2012)