ShortestPathACO based strategy to find the Shortest Path between two nodes

Mariusz Głąbowski, Bartosz Musznicki, Przemysław Nowak and Piotr Zwierzykowski
Poznań University of Technology, Faculty of Electronics and Telecommunications,
Chair of Communication and Computer Networks, Polanka 3, 60-965 Poznań, Poland

***Abstract*—The paper presents the means to use ShortestPathACO algorithm to find the shortest path between a pair of nodes in a directed graph. This Ant Colony Optimization metaheuristic based algorithm allows to influence the quality of generated solutions, through the application of an approach dissimilar from the one typically used to solve a single-pair shortest path problem. The operation of the algorithm is discussed in relation to the pseudocode introduced in the paper. The attention paid to the parameters that influence the results is accompanied by the motivation of usage scenarios. Experiments carried out within the custom made framework of the experiment are the source of suggestions concerning the selection of values and computation methods optimal for particular applications. The influence of the choice of number of ants and the pheromone evaporation speed is investigated. The quality of solutions generated by ShortestPathACO algorithm is also addressed. The issues of execution time and convergence achievement are considered as well.**

I. Introduction

The ShortestPathACO algorithm is a method and a set of recommendations to ensure that the Ant Colony Optimization (ACO) metaheuristic for solving the shortest path problem is properly applied. A detailed introduction to the methodology, discussion on the context and the methods for finding paths, as well as a discussion on the classes of parameters and the methods for updating pheromones are presented by the authors in [1]. The following sections of this publication are meant to present the analysed problem more thoroughly and go deeper into the purpose of investigating some of the considerations mentioned and indicated in the previous paper, i.e., the application of the ShortestPathACO-based strategy in finding the Shortest Path between two nodes.

A. Problem of the shortest path between a pair of nodes

For the directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where N is the set of nodes (vertices) and A is the set of arcs (edges), we assign the length a_{ij} to each of its edges $(i, j) \in \mathcal{A}$ (alternatively, this length can be also called the cost). For the resulting path $(n_1, n_2, \ldots, n_k)$, its length can be expressed by Formula (1).

$$a_{ij} = \sum_{i=1}^{k-1} a_{n_i n_{i+1}} \tag{1}$$

A path is called the shortest path if it has the shortest length from among all paths that begin and terminate in given vertices. The shortest path problem involves finding paths with the shortest lengths between selected pairs of nodes. The initial node will be designated as s, while the end node as t.

A number of basic variants of the shortest path problem can be distinguished [2]:

- finding the shortest path between a pair of nodes,
- finding the shortest paths with single initial node,
- finding the shortest paths with single end node,
- finding the shortest paths between all pairs of nodes.

These matters find their application in different areas such as routing in communication networks [3]–[5], pipeline transport or satellite navigation. Therefore, this paper focuses on discussing a method for solving the problem of the shortest path between a pair of nodes in a directed graph, often called a single-pair shortest path problem [6].

B. ShortestPathACO algorithm

Before executing the ShortestPathACO algorithm, one should have a full awareness that the Ant Colony Optimization metaheuristics has been constructed to seek solutions of $\mathcal{NP}$-hard problems [7]. As such, it does not always guarantee finding the most optimum solution. 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. Moreover, in particular situations a solution may not even be found. Because of this particular feature, it is extremely important 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 for the ShortestPathACO algorithm, appropriate parameters, methods and ways that prove to be the most effective in solving a given problem have been eventually chosen and established by the authors. For this reason, the further part of the paper discusses the authors' proposal of the parameters that can be used in solving a single-pair shortest path problem, presents a pseudo-code of the method and gives a detailed discussion on the operation of the algorithm. Then, we proceed to present the relevant simulation study carried out within the framework of the experiment and analyse the influence of the choice of parameters on the quality of generated solutions, execution time and on the process of reaching convergence for the algorithm. Final remarks and conclusions are presented in the summary.

II. Types of parameters and their values

The ShortestPathACO algorithm uses the following parameters:

m – the number of ants,
α – the parameter that defines the influence of pheromones on the choice of the next vertex,
β – parameter that determines the influence of remaining data on the choice of the next vertex,
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,
s – initial node,
t – end node.

In solving a single-pair shortest path problem, the parameters m, α, β and ρ have to be accommodated to a specific problem, whereas the value τ_0 is set to 0 to provide the opportunity of the use of the transition stage during which the calculation of the edge coefficients involves one of the adopted equations [1]. The parameter τ_{min} is set to 0, while τ_{max}, through the application of a very high value, is not taken into consideration at all.

Finding paths is executed according to a time list that keeps records of times during which an ant reaches a given vertex. What follows is that the quality of the paths generated by individual ants is enhanced more efficiently. To ensure checking of all possible edges, Formula (2) is used in the process of selection of the next vertex, while in the transition stage Formula (3) is used. The use of the second stage during the process of selection of next edges is deactivated (the edge coefficient equal to 0, which would result in a random-type selection of edges), because this can be compensated by a greater number of ants, thus optimizing the efficiency of the algorithm.

$$q_{ij} = \begin{cases} \tau_{ij}^{\alpha}(1+\beta)^2 & \text{for} \quad \begin{cases} vi\_nodes_j = false \\ vi\_edges_{ij} = false \end{cases} \\ \tau_{ij}^{\alpha}(1+\beta) & \text{for} \quad \begin{cases} vi\_nodes_j = true \\ vi\_edges_{ij} = false \end{cases} \\ \tau_{ij}^{\alpha}(1+\beta) & \text{for} \quad \begin{cases} vi\_nodes_j = false \\ vi\_edges_{ij} = true \end{cases} \\ \tau_{ij}^{\alpha} & \text{elsewhere} \end{cases} \tag{2}$$

$$q_{ij}^{'} = \begin{cases} \left(\frac{1}{a_{ij}}\right)^{\alpha}(1+\beta)^2 & \text{for} \quad \begin{cases} vi\_nodes_j = false \\ vi\_edges_{ij} = false \end{cases} \\ \left(\frac{1}{a_{ij}}\right)^{\alpha}(1+\beta) & \text{for} \quad \begin{cases} vi\_nodes_j = true \\ vi\_edges_{ij} = false \end{cases} \\ \left(\frac{1}{a_{ij}}\right)^{\alpha}(1+\beta) & \text{for} \quad \begin{cases} vi\_nodes_j = false \\ vi\_edges_{ij} = true \end{cases} \\ \left(\frac{1}{a_{ij}}\right)^{\alpha} & \text{elsewhere} \end{cases} \tag{3}$$

$$\Delta\tau = \frac{1}{a_P} \tag{4}$$

The levels of pheromones are updated in steps during the return of ants to the initial vertex, which makes it possible to take account of the quality of a solution in the determination of $\Delta\tau$, and at the same time enhances short paths. The $\Delta\tau$ itself is calculated by Formula (4) that takes into consideration the length of a path because, in the case of the considered problem, it is the only necessary element.

In the initial runs of the algorithm (the first three routes of each of the ants), $\Delta\tau$ can be set to 0 to introduce more randomness and increase variability in the choices of the ants, which is particularly helpful in finding new paths, particularly in graphs with complex structure. However, this strategy was discarded in the experiments discussed in this paper because it was proved that the result of its introduction was a longer time for the algorithm to achieve convergence.

III. Pseudo-code and a detailed discussion on the algorithm

Pseudo-code 1 presents the ShortestPathACO algorithm. As in the majority of algorithms, its operation is based on the loop (line 2) that is executed $iterattion\_limit$ times at the maximum. In the majority of cases, with large values of this limit, the algorithm will achieve convergence much earlier and will terminate its operation. The mentioned limit is introduced to prevent the algorithm from being infinitely executed in the case of a failure in finding a solution. In each iteration, an ant that is assigned to the lowest time value in the list is retrieved from the time list. If a greater number of ants has been assigned the same time, then they are retrieved one by one in the subsequent iterations. Immediately after the retrieval of an ant from the list, this ant is removed form the list. Each ant has been assigned to the vertex in which it is currently located. This vertex is then compared with the target vertex in line 8.

If the current ant reached the end vertex, a number of operations is initiated. The ant's mode is realigned to reverse — this means that in next iterations the ant will be returning towards the initial vertex along the path that has been previously found by it. The next operation is to calculate $\Delta\tau$ for this ant, which is executed by Function ***ComputeDTau(k,length,m)*** in line 10. If the ant embarks on one of the first 3 routes, $\Delta\tau$ takes on the value 0 for the ant — the pheromone trail on this path is not further enhanced, which has been explained and reasoned earlier. If, however, the ant has already covered as many as 3 routes, $\Delta\tau$ is calculated in a regular way as the inverse of the length of the path found by the ant. If the ant's path is the shortest path from among the number of all paths that have been found so far, the value $\Delta\tau$ is multiplied for the ant by the number of ants m. This operation allows us to considerably shorten the execution time of the whole of the algorithm.

The next step is to compare again the length of the path found by the ant with the length of the shortest path that has been found so far. This operation provides the opportunity to update the value of the length of the path and the path itself that are stored globally. Then, it is checked whether the ant's path has the length that is equal to the length of the path found by the ant that reached the end vertex earlier as the preceding ant. The reason for this operation is the need for establishing whether the pheromone trail on the paths of the graph has been stabilized enough to assume that the next iterations and the paths to be found by ants in the process will not introduce any improvement to the currently best solution found so far. Hence, if all ants, one by one, find a path with the same length, the algorithm terminates its operation. If, however, the

Algorithm 1: ShortestPathACO

Data: graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, edge weights vector a, initial vertex s, end vertex t, number of ants m, pheromone influence factor α, influence factor for other related data β, pheromone evaporation speed ρ, initial value of pheromones τ_0, minimum value of pheromones τ_{min}, maximum value of pheromones τ_{max}, iteration limit $iteration\_limit$

Result: path found $path$, length of found path $length$

```
Initialize(G, s);
for iteration ← 1 to iteration_limit do
    k ← GetRoot(list);                                   // Get an ant from the beginning of list
    time ← GetRootKey(list);                             // Get time from the beginning of list
    RemoveRoot(list);                                    // Remove ant from the beginning of list
    i ← GetNode(k);                                      // Get current vertex of ant k
    skip ← false;
    if i = t then
        SetMode(k, true);                                // Set the k ant's mode to reverse
        ComputeDTau(k, length, m);                       // Calculate Δτ of ant k
        if GetLength(k) < length then                    // Get length of path of ant k
            path ← GetPath(k);                           // Get path of ant k
            length ← GetLength(k);                       // Get length of path of ant k
        end
        if GetLength(k) = last_length then               // Get length of path of ant k
            convergence ← convergence + 1;
            if convergence ≥ m then break;
        else
            convergence ← 0;
        end
        last_length ← GetLength(k);                      // Get length of path of ant k
        (e, f) ← GetLastArc(k);                          // Get the last edge from the path of ant k
        SetNode(k, e);                                   // Set the current vertex of ant k
        Add(list, time + a_ef, k);                       // Add ant k to list with given time
        skip ← true;
    end
    if GetMode(k) = false then                           // Get mode of ant k
        arc ← SelectNextArc(k, i);                       // Select next vertex for ant k
        if arc = NULL then
            (e, f) ← GetLastArc(k);                      // Get the last edge from the path of ant k
            if e = s then
                NextRun(G, k, s);                        // Prepare ant k for next route
                Add(list, time, k);                      // Add ant k to list with given time
            else
                RemoveLastArc(k);                        // Remove the last edge from the path of ant k
                SubstractFromLength(k, a_ef);            // Subtract a_ef from length of path of ant k
                SetNode(k, e);                           // Set the current vertex of ant k
                Add(list, time + a_ef, k);               // Add ant k to list with given time
            end
        else
            (i, j) ← arc;
            SetNode(k, j);                               // Set the current vertex of ant k
            SetVisited(k, j);                            // set vertex j as "visited" by ant k
            vi_nodes_j ← true;
            vi_arc_ij ← true;
            AddArc(k, (i, j));                           // Add edge (i, j) to path of ant k
            AddToLength(k, a_ij);                        // Add a_ij to the length of path of ant k
            Add(list, time + a_ij, k);                   // Add ant k to list with given time
        end
    else if skip = false then
        AddToTau(k);                                     // Add Δτ of ant k to the last edge of its path
        if i = s then
            NextRun(G, k, s);                            // Prepare ant k for next route
            Add(list, time, k);                          // Add ant k to list with given time
        else
            RemoveLastArc(k);                            // Remove the last edge from the path of ant k
            (e, f) ← GetLastArc(k);                      // Get the last edge from the path of ant k
            SetNode(k, e);                               // Set current vertex of ant k
            Add(list, time + a_ef, k);                   // Add ant k to list with given time
        end
    end
    for time to GetRootKey(list) −1 do                   // Get time from the beginning of list
        Evaporate();                                     // Evaporate pheromones on all edges
    end
end
```

preceding ant finds a path with a different length, the auxiliary counter for the evaluation of the convergence of the algorithm $convergence$ is reset.

It is worthwhile to emphasise at this point that a comparison of the lengths of subsequent paths found in the iterations is not fully reliable because it may turn out that different paths can have exactly the same length. However, making a comparison of the whole of the paths, edge by edge, would be too costly computationally. On the other hand, even if different paths have the same length, it is of no major significance because no matter which path will be chosen, we will get the optimum solution in the end.

After these operations have been completed, the length of a path of a given ant is recorded and the last edge of this path is mapped to the list. The edge is necessary to determine the preceding vertex which was visited by the ant. This vertex is then set as the current vertex for the given ant, while the ant itself is added to the time list with the time equal to the current time extended by the length of the edge excerpted a moment ago. In addition, the auxiliary variable $skip$ is set to $true$, which prevents the ant from passing along the last edge of its path in the current iteration.

Depending on what mode a given ant is in, the algorithm executes different operations. In the progressing mode (finding the end vertex) in line 28 the function *SelectNextArc(**k,i**)* is executed. The function aims at selecting the next edge for the ant currently under consideration to follow. This function calculates for each of the edges coming out of the vertex in which the ant is currently located the probability for the ant to pass along this edge (on the basis of the edge coefficient q_{ij}). The edge that has the highest probability is then chosen as the next to be followed by the ant. In the case of these probabilities being equivalent (the same), the edge is chosen randomly. If, however, there are no edges coming out of the considered vertex or all vertices are already in the current path of the ant, $NULL$ is returned. When this is the case, the last edge in the ant's path is retrieved and its vertices are recorded and remembered. If the vertex from which the edge leaves is the initial vertex, the function *NextRun(**G,k,s**)* is executed. The function's task is to clear all data gathered during the ant's route that has just been terminated and to prepare the ant for the next route. After these operations have been completed, the counter of the ant's routes is extended, while the initial vertex, which in turn is labelled as the one already visited by the current ant, is set as the current vertex. After all these operations are completed, the ant is added to the time list with the current time for the ant to be ready to start the path searching process anew. All the described operations are necessary because an ant that would be returned to the initial vertex, most probably would stay there until the end of the execution of the algorithm, which, taking into account a low number of ants, would decidedly worsen the quality of generated solutions.

In a situation where the function *SelectNextArc(**k,i**)* returns $NULL$ and the ant does not have to be drawn back to the initial vertex, the last edge in its path is removed. The length of the ant's path is diminished by the length of this edge. The preceding vertex in this path (the vertex from which the edge that has just been removed leaves) is set as a new vertex which the ant visits at the moment, while the ant itself is added to the time list with the time equal to the current time extended by the length of the edge. The above strategy provides an opportunity to avoid a situation when the ant stops in one place and cannot move any further.

If the function *SelectNextArc(**k,i**)* returns the edge, the vertex in which this edge terminates is set as the current vertex for the ant. In addition, this vertex is labelled as one that has been visited by this ant. This is, in fact, part of the mechanism that prevents the creation of loops in the paths generated by ants. Additionally, the vertex that has just been set is labelled in the global table as "visited". The similar situation is with the edge — it is labelled as "visited" in the global table. These tables are then useful in the process of determining the edge coefficients in the selection of the next vertex. The next step is to add the selected edge to the ant's path and to extend the length of this path by the length of this edge. Then, the ant is added to the time list, as usual with the time equal to the current time extended by the length of the edge.

For an ant that is in the reverse mode (return to the initial vertex), the same operations are executed. In the function *AddToTau(**k**)* (line 51), the last edge along the ant's path (the edge that has just been covered by the ant) is retrieved and its vertices are remembered. At this point, an increase in the value of pheromones by $\Delta\tau$ ensues (set earlier for this ant), and, should the need arise, its possible decrease to τ_{max}, if this value has been exceeded. Next operations depend on whether the vertex from which the edge that has just been removed extends is the initial vertex.

If such a situation occurs, the function *NextRun(**G,k,s**)*, described earlier, is executed. As previously (in the previous place of its application), the ant is added to the time list with the current time. This is the second situation when the ant is added to the time list without extending the current time by the length of the edge. The reason for this is the fact that the ant reaching the initial vertex immediately starts its next route.

In the case where the ant has not yet reached the initial vertex and is in the reverse mode, the last edge in the path of the ant (the edge that has just been covered by the ant) will be removed from the ant's path. Thanks to this, in the subsequent steps, until the initial vertex is reached, it is the last edge that is still along the path of a given ant that will always be retrieved. Then, the next one to be retrieved from the path is the edge that is currently the last one, while the vertex from which it extends is set as the current vertex which the ant visits at the moment. What is to be done now is just to add the ant to the time list, with the time composed of the current time plus the length of the currently last edge in the ant's path.

The last operation to be performed in the algorithm is to diminish the value of pheromones on all edges by the fraction determined on the basis of the parameter ρ with the application of the function *Evaporate()*. At the same time, the value of pheromones is maintained at the minimum level τ_{min} [8]. In order to best map the duration of time (that we use to additionally enhance short paths), this operation is executed for the current time value and for all values lower than the subsequent time value appearing in the time list. This simulates evaporation of pheromones for each total time value.

IV. Study on the operation of the algorithm

In order to verify the operation of the algorithm and to analyse its performance, a number of tests with different graphs and values of individual parameters were performed. For a presentation in this paper, the hand-constructed graph that is presented in Figure 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 path to be found was between the first and the last vertex.

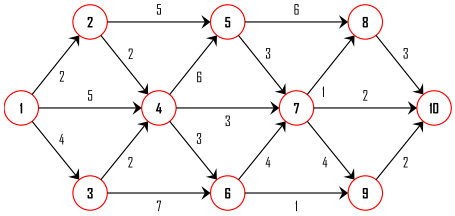

Figure 1. The graph in which the study on the operation of the ShortestPathACO algorithm was performed

The values of the parameter α were experimentally determined as 1, while those of the parameter β as 0.5. It should not be forgotten that the values of these parameters are extremely important and in the case of another type of a graph or another type of the shortest path problem, these values should be verified. 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 is worthwhile to emphasize again that both the use of the second stage during the selection of next edges, in which the edge coefficient is set to 0, and resetting $\Delta\tau$ during the first three routes of ants was deactivated.

The experiments carried out in the study focused on three aspects. The influence of the choice of the number of ants m and of the parameter ρ on the quality of generated solutions and the execution time of the algorithm were studied. Another thing to be checked during the study was to evaluate how the algorithm reaches its convergence.

A. Percentage of correct solutions

Figure 2 shows a chart presenting how the algorithm behaves in the studied graph depending on a change in the number of ants m for different values of the speed ρ at which pheromones evaporate. With a small number of ants, the algorithm is not capable of finding a correct solution at all. This does not mean, however, that the solutions generated by the algorithm are far from being close to the optimal solution. In certain applications, however, these results can be sufficient, and it is then this type of compromise between the demanded resources and the accuracy of results is worth being considered. When the number of ants is increased to 6, we observe an increase in the percentage of the number of correct solutions to about 75% depending on the parameter ρ. With 8 ants, this value approaches 95%, whereas when the number of ants is higher or equal to 12, the algorithm is virtually always capable of finding optimum solutions. This illustrates well the behaviour of the algorithm. With a small number of ants for the graph with a far higher number of edges, the algorithm is not capable of checking all paths, hence the shortest path cannot be found. A gradual increase in the number of ants causes ants to find the path more and more often until an increase in their number introduces no substantial changes. For the graph from Figure 1, $m = 12$ is the threshold that, in practice, guarantees finding the optimal solution. An increase in the parameter ρ is not followed by any noticeable changes. The difference in the percentage number of correct solutions for a given value m can be regarded as just a insignificant fluctuation. The chart clearly shows that, no matter what the value of the parameter ρ is, the quality of the generated solutions is approximately a logarithmic function of the number of ants m.

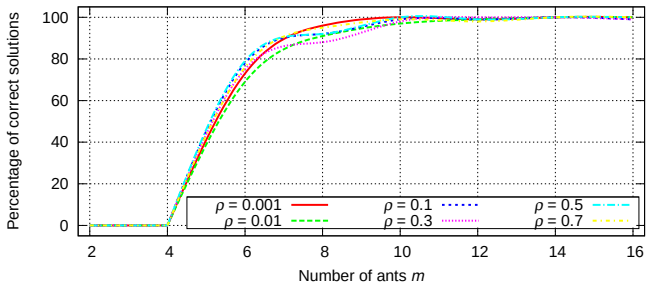

Figure 2. Graph (chart) of the quality of generated solutions provided by the ShortestPathACO algorithm for the graph from Figure 1 depending on the parameters m and ρ

B. Duration of operation

The graph from Figure 3 illustrates the average operation times in relation to the parameters m and ρ. Elongation of the operation (execution) time of the algorithm in relation to the increase in the number of ants is clearly visible. Two factors contribute here. Firstly, a greater number of ants is followed by a greater number of paths that will be eventually found, which is undoubtedly time-consuming and increases the amount of necessary computational work. On the other hand, however, achieving convergence is far more difficult with a great number of ants. The process of stabilization of the pheromone trail is decidedly longer with a higher number of found paths because the random nature of the operation of ants may cause, at one point, that one of the ants can choose a path that is completely different from all the others. Taking into account earlier considerations on the issue of the quality of generated solutions, a determination of a threshold that would guarantee finding the optimal solution with the probability approaching 100% is extremely important because it prevents the algorithm from being trapped in the application of parameters that make its execution longer with the same results obtained in the process.

The influence of the parameter ρ is not so unequivocal though. By making a generalisation, it can be said that its increase is equivalent to elongation of the execution time of the algorithm, though there are some exceptions to this rule. An increase in ρ above 0.1 causes a significant increase in the operation time of the algorithm, but with a greater number of ants, for the value equal to 0.7, a slight decrease as compared to lower values is observable. The parameter ρ, responsible for the evaporation rate of the pheromone trail, prolongs the

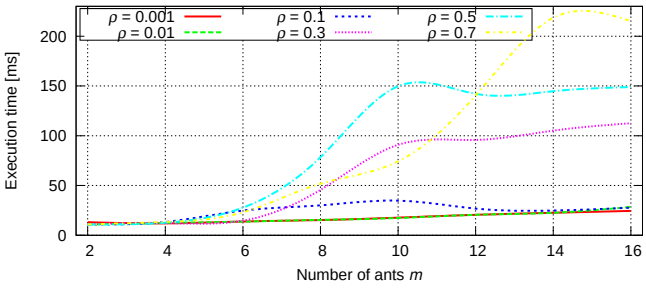

Figure 3. Execution time of the ShortestPathACO algorithm for the graph from Figure 1 depending on the parameters m and ρ

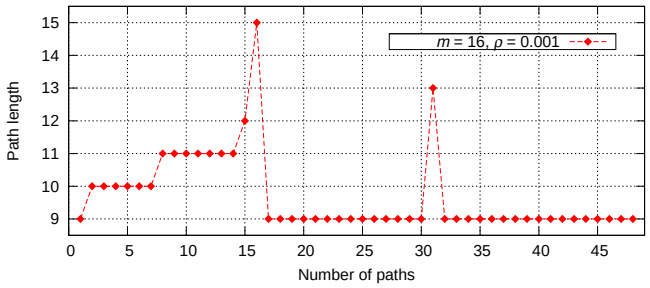

Figure 4. An exemplary process of achieving convergence in the ShortestPathACO algorithm for the graph from Fig. 1 for $m = 16$ and $\rho = 0.1$

operation time of the algorithm because it regularly decreases the pheromone level on the edges, which translates into a decrease in the differences between them. One can say that a too high increase in ρ causes a decrease in the influence of short paths on ants. Based on the obtained results the conclusion is that, even with high values of ρ, the optimal solution can be found, though it takes decidedly much more time. Yet another conclusion can be drawn - since the parameter ρ does not influence significantly the quality of generated solutions, it should not be exceedingly increased because this will solely translate into elongation of the execution time of the algorithm.

C. Convergence achievement

The next issue under consideration is the way the algorithm reaches convergence. Figure 4 shows the most frequent behaviour of the algorithm. The paths found by all 16 ants are arranged in the increasing order, since shorter paths are found earlier. When the ants begin their next tour, two situations can occur. It is either the ants achieve convergence to the best solution found in the previous route, or they will continue their search. If the pheromone trail stabilizes in the next tours sufficiently for all the ants to choose one path, the algorithm terminates its operation. There may be, of course, certain departures to this, which is to be seen in the discussed figure. One of the ants chose a different path than the rest, hence the process of convergence achievement was prolonged. If, however, the condition for the termination of the execution of the algorithm was modified in such a way that it would be required for $\frac{2m}{3}$ or even $\frac{m}{2}$ ants from the whole of the ant colony to choose the same path, convergence would be achieved earlier. Otherwise, ants will keep on finding different paths (sometimes paths that have already been found earlier) until the algorithm reaches the limit $iteration\_limit$ and will terminate its operation without achieving convergence. Such a situation is disadvantageous because of two reasons. Firstly, the execution time of the algorithm will be prolonged considerably and, secondly, the quality of thus obtained solution is questionable. If such an event occurs, this probably means that the parameters of the algorithm have been inaccurately selected, or that the algorithm is not capable of finding a solution for this particular graph.

V. Conclusions

The method for solving the single-pair shortest path problem that applies the ShortestPathACO algorithm presented in the paper is strongly related to the operations stemming from the Ant Colony Optimization metaheuristics. In general, these is no hundred per cent guarantee that the obtained solutions will be optimal. There is, however, a possibility to efficiently improve the quality of obtained paths through an appropriate adjustment of the mode of operation and the selection of the parameters of the algorithm depending on a specific context of the task to be solved. The results of the study prove that through the optimization of the use of the algorithm and the establishment of thresholds appropriate for a given class of problems, it is possible to guarantee to obtain optimal results with the probability of nearly 100%. At the same time, it is still possible to shorten the execution time of the algorithm and to limit the number of operations performed in each of the cycles. Moreover, with certain applications, a need for reaching a compromise between the required computational resources and the accuracy of results can be justified. When this is the case, the shortest path between two nodes can be found at a lesser expense, with the expected level of the quality of result (the cost of a path) retained.

The considerations presented in the paper indicate that an analysis and a presentation of the behaviour of the algorithm in other types of graphs, such as multi-stage graphs, is the appropriate area for further research. ShortestPathACO seems to present a substantial potential for is application also in the area of solving single-source shortest path problem.

References

[1] M. Głąbowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "Shortest Path Problem Solving Based on Ant Colony Optimization Metaheuristic," *International Journal of Image Processing & Communications*, vol. 17, no. 1–2, pp. 7–17, 2012.

[2] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, N.J.: Prentice Hall, 1993.

[3] K. Stachowiak, J. Weissenberg, and P. Zwierzykowski, "Lagrangian relaxation in the multicriterial routing," in *IEEE AFRICON*, Livingstone, Zambia, September 2011, pp. 1–6.

[4] M. Piechowiak, M. Stasiak, and P. Zwierzykowski, "The Application of K–Shortest Path Algorithm in Multicast Routing," *Theoretical and Applied Informatics*, vol. 21, no. 2, pp. 69–82, 2009.

[5] B. Musznicki, M. Tomczak, and P. Zwierzykowski, "Dijkstra-based Localized Multicast Routing in Wireless Sensor Networks," in *Proceedings of International Symposium on Communication Systems, Networks and Digital Signal Processing*, Poznań, Poland, 18–20 July 2012.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*. Cambridge, Massachusetts: The MIT Press, 2009.

[7] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, Massachusetts: The MIT Press, 2004.

[8] T. Stützle and H. H. Hoos, "$\mathcal{MAX}$-$\mathcal{MIN}$ ant system," *Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000.