Mobile Ad-hoc Networks: an Experimentation System and Evaluation of Routing Algorithms

Maciej Foszczynski, Marek Adamczyk, Kamil Musial, Iwona Pozniak-Koszalka, Andrzej Kasprzak
Dept. of Systems and Computer Networks, Wroclaw University of Technology
Wroclaw, Poland
e-mail: iwona.pozniak-.koszalka@pwr.wroc.pl

***Abstract*—The paper concerns the problem of path finding in wireless ad-hoc networks. Several algorithms, including meta-heuristic algorithms, evolutionary algorithm and the created hybrid algorithm, are considered. Algorithms have been implemented into a designed experimentation system. The system allows making simulation experiments along with multistage experiment design. In the paper, the results of some experiments are discussed. Moreover, the comparative analysis of efficiency of algorithms is presented. It may be concluded that the proposed hybrid algorithm seems to be promising.**

Keywords-wireless network; ad-hoc network; path finding; meta-heuristic algorithms; hybrid algorithm, experimentation system, simulation, efficiency

I. INTRODUCTION

Mobile wireless ad-hoc networks are networks with a short period of life. An ad-hoc network is a wireless network to which mobile devices that can act both as client and access point are connected. The most characteristic feature of the ad-hoc network is the lack of any central control device, and also any device to supervise the operation of this information exchange system. Another important feature is the lack of fixed network infrastructure. Systems with this type of connection, therefore, are characterized by high variability and irregularity, which implies the problems absent, or present to a lesser extent in the standard fixed infrastructure networks, both wired, and wireless. Mobility of devices forming such structure is the cause of irregular construction and is a reason of frequent changes in the network structure. The consequence of these characteristics is high importance of algorithms to find not only the shortest path leading from source to destination node, but also to be able to find it fast, regardless of network structure changes. Performance of the algorithm that solves this problem with a large variation of the network structure is crucial, because the algorithm will have to be used after any change in the network structure.

This paper in its content aims to present and formulate the problem (Section II), and demonstrates the variety of its synthetic solutions (Section III). Major emphasis has been made to describe and present the experimentation system created (Section IV), and the results of testing of certain algorithms obtained with this system and using multistage experiment design ideas [1] (Section V). In the final part of the paper, the matter of prospects for the future is raised, including a summary (Section VI).

II. PROBLEM STATEMENT

To fully realize the problem of path finding in a graph of mobile ad-hoc network, one have to imagine a sample network, like the one shown in Fig. 1. It is clear to see, that from a mathematical point of view, this problem can be reduced to find the shortest path between two vertices of an undirected graph.

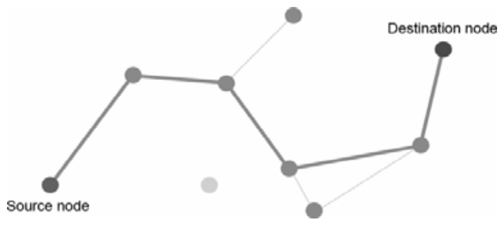

Figure 1. Sample structure of ad-hoc network.

Mathematical model symbolizing the entire analysed network is a non directed, weighted graph. Vertices in the graph represent individual devices in the network. Connections between the vertices are the physical representation of the wireless connections between devices. The weight of each of the edges in the form of a specific number, defines the quality of the connection. In order to simplify the mathematical analysis of the problem, it can be assumed that the larger the weight, the worse the connection quality. The final element which is necessary to build a full, abstract representation of the problem is to determine the conditions of existence of the connections between vertices.

In the proposed model, the possibility to connect two vertices in the graph is defined by their range, which is an abstract representation of the range of wireless devices in real ad-hoc networks. In the mathematical model, it will also be the number given in standardized units, to determine the radius of coverage of the given vertex. Based on the radius, it can be determined which of the neighboring vertices of a vertex can connect to it and, therefore, can be connected with an edge, what may represent a real connection.

III. THE ALGORITHMS

Two proactive algorithms and two author's reactive algorithms are under consideration, including implementations of Dijkstra and A-star algorithms, as well as ACO (Ant Colony Optimization) and Hybrid algorithms. Dijkstra's and A* algorithms' main purpose was to provide comparison to the reactive algorithm in a modified form of Ant Colony Optimization, and the proposed (by the authors of this paper) hybrid algorithm, which is a combination of modified versions of two of the selected algorithms.

A. Dijkstra and A-star Algorithms

Dijkstra's algorithm is an algorithm that always returns the optimal or close to the optimal route, although it is computationally greedy. In this case, the algorithm has been modified in such way, that after finding the path to the destination node it finishes the path finding process.

Necessary condition for the algorithm is to divide the vertices of a graph into two sets [2]. One set contains the vertices to which paths have been already counted, and the other contains all the nodes which have not yet been processed.

Determination of the path is made iteratively, e.g. [3]. As the first vertex, the initial, start vertex of the simulation is set. In the A* algorithm, like Dijkstra's algorithm, gives the optimal path between two vertices of the graph, but to calculate the path it uses heuristics [4].

The algorithm minimizes the function *f(x) = g(x) + h(x)* where *g(x)* is the distance from the start node to the vertex *x* and *h(x)* is the path predicted by the heuristic from the vertex *x* to the destination node. The values of *f(x)*, *g(x)* and *h(x)* are stored in three tables [5].

As heuristic functions, we have chosen the „Euclid" function (1), and „Manhattan" function (2).

$$h(x) = \sqrt{(x.X - end.X)^2 + (x.Y - end.Y)^2} \qquad (1)$$

$$h(x) = |x.X - end.X| + |x.Y - end.Y| \qquad (2)$$

Determination of the path is iterative, as in Dijkstra's algorithm e.g. [6].

B. Ant Colony Optimization Algorithm

The idea of the ant colony optimization is to base the algorithm's work on the behaviour of the colony of ants, seeking a route from their nest to food source and back again, e.g. [7].

Ants, as they move along the edges of the graph, leave their pheromone to indicate to the other ants that the edge has already been visited [8]. With time, the concentration of pheromone P_c on the edges of the graph is decreasing with concentration loss factor *l*, i.e. *new* $P_c = P_c \cdot l$.

Pheromone concentration loss process is continuous and occurs at the beginning of each run of the algorithm's iteration, e.g. [9].

The proposed modification of a classic ACO consists in dividing ants into two categories: forward and backward ants. Forward ants' main purpose is to explore the graph and to find the destination node. When forward ant reaches the destination, it sends back backward ant and dyes. Backward ants are much more likely to follow the pheromone, because their priority is to consolidate the route and get back to the source node quickly, from where they send forward ants again.

In a classic implementation of this algorithm, routing tables are used to locally memorize the results of the algorithm's work in the network. For the means of an abstract implementation, routing tables have been omitted, as assumed that the subject of the research was the path finding itself, rather than maintaining the route within a given instance of the problem.

Determination of path length in this algorithm is made in an iterative manner. The path which ant chooses for the next step is added to the total value for each ant. Final result is determined as the shortest path of all of the ants.

C. Hybrid Algorithm

Hybrid algorithm is an author's algorithm, which was developed in response to the need to reduce the cost of finding the path, regarding the implementation of the first n steps as quickly as possible, and then, after a quick advancement in path selection in the first stage, further optimization of the path made by using one of specialized algorithms e.g. [10].

To implement this algorithm, modified version of ACO was implemented in conjunction with Dijkstra's algorithm e.g. [11]. Modification has been made to limit the amount of ants and to modify the way the ant chooses its next vertex in the graph. Algorithm obtained in this way allows for a close to random, but relatively controlled first *n* steps, which will be made. After completing *n* steps, the ACO finishes and passes its current vertex as the starting vertex for the next algorithm.

After the calculation of the initial direction, Dijkstra's algorithm is run, which is aimed to find the path to the destination node if it has not been reached yet e.g. [12].

Path length in this algorithm is made in an iterative manner, as a sum of path values given by both of the algorithms.

IV. EXPERIMENTATION SYSTEM

A. Basic Characteristic

The Windows platform has been chosen as an implementation environment, on which an application in C# programming language has been created. To run the simulator, the workstation must be equipped with Windows 2000/XP/Vista/7 operating system and .NET Framework 3.5.

The simulator has an interface that allows the user to easily configure all the parameters of the application. Moreover, its construction allows to quickly and easily extending its capabilities, including possible addition of new algorithms.

B. Functiona Features of Application

After launching the simulator application, the application main window appears, as shown in Fig. 2. The main window is divided into clearly separated areas.

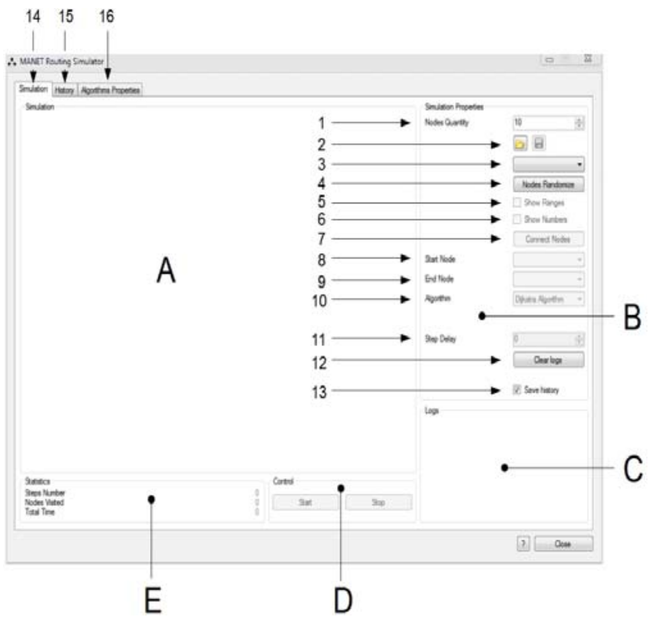

Figure 2. Main window of application

The largest area of the application window is the area of simulation (A). In this area the graph representing the specific problem and the effect of the algorithm will be shown. It is also possible to modify a specific instance of the problem before running the algorithm itself.

In the settings area, (B), we see the basic parameters that can be modified in the program. The first one is the parameter determining the number of vertices in the graph, which is to be generated (‘1’)

Next are two buttons, allowing to save the current graph to a file and load a saved graph to the program (‘2’).

A select form (‘3’) allows choosing a specific instance of the problem saved earlier. To add a graph to the list, save it in a subdirectory called „Graphs” in the root directory of the simulator. After adding the file and restarting the application, saved graph appears on the defined graphs selection list.

Under the selection of defined graphs, we see the graph draw button (‘4’), allowing to generate a random graph, consisting of the number of vertices determined by the parameter (‘1’).

Vertex positions are random according to normal distribution. If the arrangement of the vertices is not satisfying, it is possible to draw another instance by re-clicking on the „Randomize” button, or manually modifying the position of given nodes. Nodes in the simulation area can be moved using drag-and-drop method.

Below are two fields that allow interfering in the amount of information displayed in the simulation. „Show ranges” select (‘5’), displays the circle around each of the nodes, symbolizing node's range in relation to the other vertices. Selecting „Show numbers” parameter (‘6’) will cause a number to appear next to each node which enables its identification.

The number (‘7’) in the illustration has been assigned to a button that connects all vertices in the graph. Connections are made on the basis of nodes range. The connection between the two vertices *a* and *b* may occur if, and only if, the range *r* of the vertex with less value is less than or equal to the distance d_{ab} between the vertices (3).

$$C(a,b) = \begin{cases} 1 : \max(r_a, r_b) \geq d_{ab} \\ 0 : if\ else \end{cases} \tag{3}$$

The next two fields, (‘8’) and (‘9’), allow the selection of the source and destination node in the graph. Algorithms will find the shortest path between the initial and final vertex, using only the available connections. There is a possibility that it will be impossible to find any path between two selected vertices.

After selecting the initial and final vertex, an algorithm that will look for the shortest path between them can be chosen. Selection of the algorithm takes place by selecting from the drop-down list (‘10’).

If the algorithm supports additional parameters for its operation, before the start of the simulation it is possible to configure the parameters in „Algorithm Properties” (‘16’).

The last parameter that can be set is the „Step delay” (‘11’). Here the number of milliseconds that the simulator will wait after each step of the algorithm can be specified. Note that due to the large variety of algorithms, this parameter is purely indicative.

Additional button „Clear logs” (‘12’), is used to delete the exported results of the algorithm run.

The last option available in the main settings area is a field which allows enabling or disabling algorithm run history (‘13’). When this option is enabled, step-by-step algorithm history analyse is possible in the „History” tab (‘15’).

Algorithm results field (C) is located under the main settings area. Basic results of algorithm run are shown in this field.

Below the simulation area two buttons marked „Start” and „Stop” are located (D). These buttons allow starting and stopping the simulation.

Current algorithm run information is shown in the live statistics area (E). These statistics are updated with every step of the algorithm, so if the delay of the algorithm iteration was set, it will be possible to analyse statistics during the run of the algorithm.

C. Concept of Research

Implementation environment allows for testing of the algorithms in several aspects. The index of performance treated as the measure of the efficiency, is the overall quality of the path d_i, which is obtained as a result of the algorithm run. The target function is expressed by (4).

$$F_c = \sum_i d_i \tag{4}$$

At the same time, the algorithm should visit the least amount of vertices possible, and take the smallest amount of time for its action. Number of vertices visited by the algorithm and the time of the execution are associated with its actual demand for resources and traffic generated by the algorithms in the network, therefore the quality of these parameters is not left without a meaning to the estimation of the quality of functioning of the algorithms.

Remaining at the level of abstract simulation of the behaviour of algorithms for searching paths in the graph, the quality of paths and quantity of visited vertices is taken into account and in this respect, the algorithms are compared.

V. INVESTIGATIONS

A. Research Theses

It is estimated that Dijkstra's algorithm provides an optimal, or very close to the optimal solution, but obtains it at great expense of calculation, which should result in relatively long run time. In the real network environment, the additional disadvantage of this algorithm is the need to process the entire graph each time a request to find the appropriate path is sent.

A* algorithm, based on the heuristic methodology, as a result of its action finds the optimal solution to the problem, using relatively large amount of resources to obtain it, so it predictably is to visit a large number of nodes in the graph.

Another approach to the problem is presented by the Ant Colony Optimization which in contrast to the other algorithms can run in the network for a long period of time, gradually improving the result and adapting to various network structure changes. In its abstract implementation, this algorithm should not show up in finding the optimal path, since the run time has been limited. Noteworthy, in the real implementation of the algorithm it exhibits a high degree of flexibility to adapt to rapidly changing network topology.

Experimental implementation of the hybrid algorithm is an interesting subject of research. It is difficult to accurately predict the algorithm behaviour and possible results, but according to the assumptions, the algorithm is to provide relatively satisfactory outcome in the short period of time, while showing a small number of visited vertices.

B. Experiment Design

Each algorithm was tested for five different total numbers of vertices in the graph. Instances of graphs with 20, 30, 50, 70 and 100 vertices were chosen, and saved in order to provide the same test environment for each of the algorithms. For each of the numbers of vertices in the graph and the values of parameters of each algorithm, 10 measurements were made, what allows to objectively asset the quality of the results, thus calculating the average results for each of the algorithms.

The experiment design, constructed along with the multistage experiments concept [13], was composed of the series of series of single executions of algorithms. The detailed values of the flexible parameters are specified in Table 2. It is necessary to mention, that all experiments were conducted in the environment described in the previous subsections.

TABLE 2. Experiment Design.

Algorithm	Parameter	Number of vertices				
Dijkstra	-	20	30	50	70	100
A*	Euclid	20	30	50	70	100
A*	Manhattan	20	30	50	70	100
ACO	$P_c = 0,0004$	20	30	50	70	100
ACO	$P_c = 0,0016$	20	30	50	70	100
ACO	$P_c = 0,0064$	20	30	50	70	100
ACO	$P_c = 0,0128$	20	30	50	70	100
Hybrid	n = 5	20	30	50	70	100
Hybrid	n = 10	20	30	50	70	100
Hybrid	n = 20	20	30	50	70	100

C. Results and Discussion

In the first case, the thesis, concerning the efficiency of Dijkstra's algorithm, was taken under consideration. Performed simulations of the algorithm run time for 100 vertices, shown in Fig. 3, confirm the assumption that the algorithm is characterized by a relatively low efficiency, needing a lot of time to process all the data.

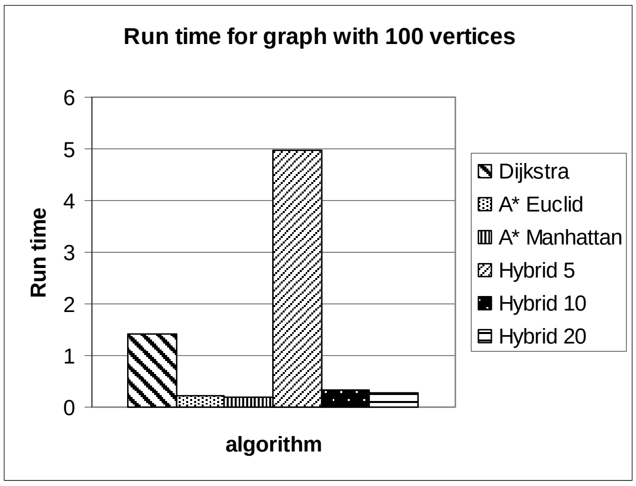

Figure 3. Run time of algorithms for 100 vertices.

It is worth to be mentioned, that a high processing time has also been obtained for the hybrid algorithm, which greater part for the graph of 100 vertices is Dijkstra's algorithm, which further confirms the truth of stated thesis.

A* search algorithm, due to the complex structure of the implementation using the heuristic methods, has proved to visit the largest number of vertices, which confirms the related thesis. Example of the number of visited nodes for the graph of 30 vertices, shown in Fig. 4, classifies it right after the Ant Colony Optimization, which in the actual implementation is intended to work without the time limitation.

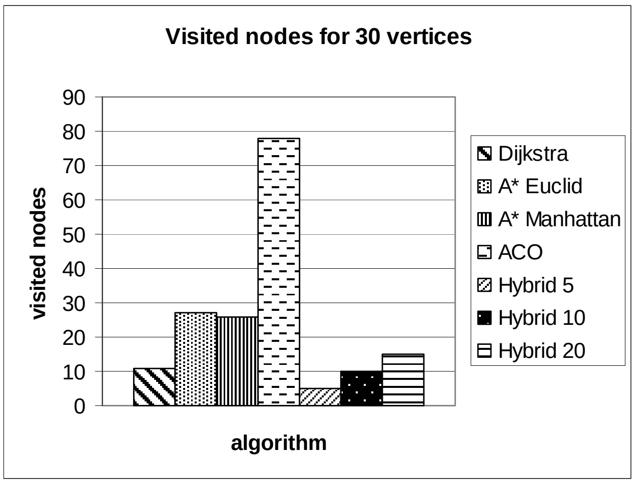

Figure 4. The total number of visited nodes for 30 vertices.

A noteworthy fact is that irrespective of the type of used heuristic function, A* algorithm, according to the thesis, is characterized by a large number of visited vertices, and so, in fact, a large number of generated connections, but generating the optimal solution of the path finding problem.

According to the thesis set for the Ant Colony Optimization, it did not provide optimal results, however, it is able to adapt to the network structure. Fig. 5 shows how the path quality obtained by the ACO differs from the quality of paths developed by other algorithms in adequate run time. Clearly, author's ACO algorithm is able to find very good quality path and is further characterized by very high flexibility of action.

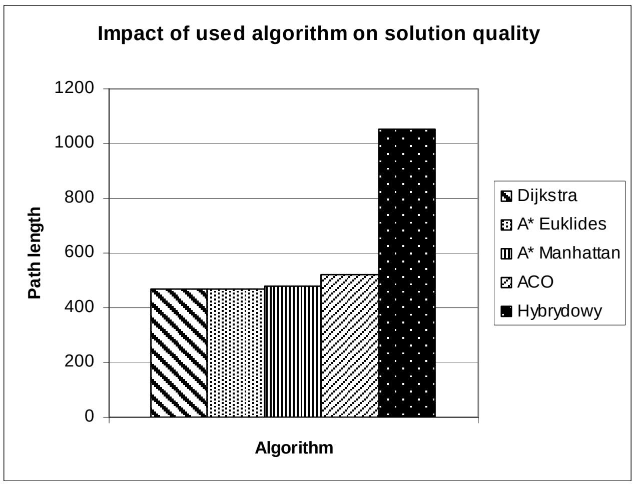

Figure 5. Path length for 30 vertices.

It is worth to note, that the quality of path obtained by the ACO changes with the pheromone concentration loss factor. Fig. 6 shows, that properly chosen pheromone loss factor can help to make the algorithm even more effective.

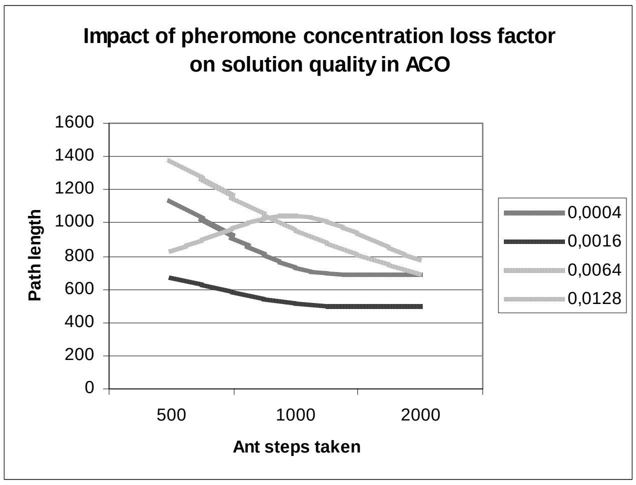

Figure 6. Impact of pheromone loss on solution quality in ACO.

The results of an experimental hybrid algorithm proved to be a confirmation of assumptions of its possible behaviour. With the increase in the contribution of modified Ant Colony Optimization, which means increasing the importance of the pseudo-random part of the algorithm, hybrid algorithm significantly increased the speed of its operation.

As shown in Fig. 7, the implementation of the first 10 steps using the modified ACO resulted in a drastic reduction of the algorithm run time, at the cost of decreasing the quality of the solution.

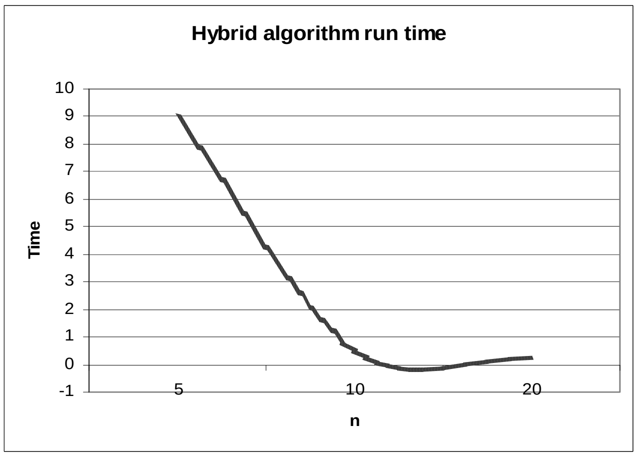

Figure 7. Hybrid algorithms run time.

With the increase of the *n* parameter, the number of steps taken by the algorithm has significantly decreased. The dependence is shown in Fig. 8. Number of visited vertices remained more or less stable, which further emphasizes the importance of pseudo-random part of the algorithm to reduce the amount of the calculation.

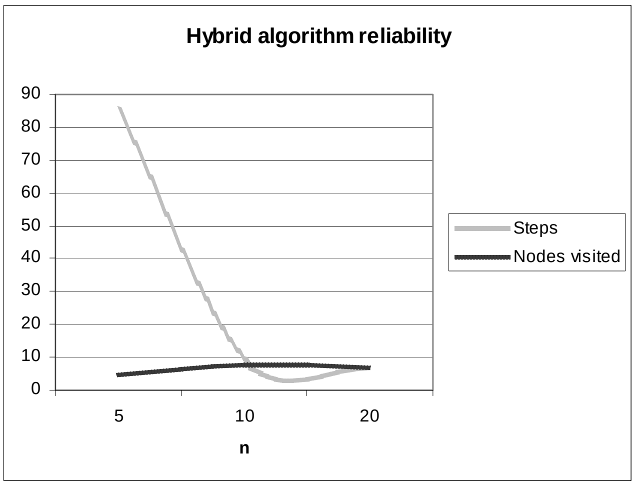

Figure 8. Hybrid algorithm reliability.

Close to random nature of the hybrid algorithm is stressed by the fact that for the *n* parameter value equal to 20, the number of performed steps has slightly increased, which is caused by too much involvement of the random part of the algorithm. Appropriately balanced algorithm parameters can improve the overall quality of obtained results and the algorithm itself provides promising results and a solid basis for further research and development.

VI. CONCLUSIONS

Research carried out allowed drawing far-reaching proposals for the design of systems based on the idea of finding a path in wireless ad-hoc networks.

Diversity of the algorithms realizing the routing in wireless ad-hoc networks available to implement requires clarifying and clearly specifying the system requirements. When it is known that the system must be resistant to changes in network and rapid adaptations to new conditions, it is advised to use algorithms that provide the desired flexibility, for example, Ant Colony Optimization algorithm. If the key is to obtain a satisfactory solution to the problem in the shortest time possible and subjects minimize the consumption of resources, a good solution could be a hybrid algorithm, similar to the algorithm proposed in this paper, which can combine the best features from selected algorithms while maintaining an appropriate balance between their drawbacks.

In the future implementation of similar project, the right direction would be to develop the idea for giving possibilities of simulations closer to the reality, gradually to move away from abstract approaches. This would enable more specific implementation of the algorithms for selected problems and to conduct more in-depth research. Nodes could use the parameters of the actual nodes of ad-hoc network, which combined with assigning more details to the connection between two nodes would increase the level of realism, which would help to carry out further tests, developing more accurate reflection of reality.

The computer experimentation system presented in this paper was designed with a possibility to expand it with additional modules. Increasing the functionality and reducing the level of abstraction can provide a solid basis for future research in this topic.

REFERENCES

[1] L. Koszalka, D. Lisowski and I. Pozniak-Koszalka, “Comparison of Allocation Algorithms with Multistage Experiments”, Lecture Notes in Computer Science, vol. 3984, Springer, 2006, pp. 58-67

[2] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs”, Numerische Mathematik, 1959.

[3] A. Kasprzak, “Packet Switching Wide Area Networks”, WPWR, Wroclaw, 1997 /in Polish/.

[4] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, “A review of routing protocols for mobile ad hoc networks”, University of Wollongong, 2003.

[5] N. Wirth, “ Algorithms + Data Structures = Programs”, Prentice Hall, 1976.

[6] M. K. Marina and S. R. Das, “On-Demand Multipath Distance Vector Routing in Ad Hoc Networks”, University of Cincinnati, 2001.

[7] M. Dorigo and T. Stützle, "Ant Colony Optimization”, MIT Press, 1997.

[8] C. Blum, “Ant colony optimization: Introduction and recent trends”, Physics of Life Reviews, 2005.

[9] M. Dorigo, “ *Ant Colony Optimization”,* Scholarpedia, 2007.

[10] Z. Michalewicz, “Genetic Algorithms + Data Structures = Evolution Programs”, Springer, 1996.

[11] A. Botea, M. Muller, and J. Schaeffer, “Near Optimal Hierarchical Path-Finding”, Journal of Game Development, 2004.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, “Dijkstra's algorithm”, Introduction to Algorithms, Section 24.3, MIT Press, 1990.

[13] D. Ohia, L. Koszalka, and A. Kasprzak, “Evolutionary Algorithm for Congestion Problem in Computer Networks”, Springer, Lecture Notes in Artificial Intelligence, vol. 5711, 2009, pp. 113-122.