Comparative Analysis of the Algorithms for Pathfinding in GPS Systems

Dustin Ostrowski

Metegrity Inc.
Edmonton, Canada
e-mail: dustin.ostrowski@gmail.com

Iwona Pozniak-Koszalka, Leszek Koszalka, and Andrzej Kasprzak
Wroclaw University of Technology
Wroclaw, Poland
e-mail: {iwona.pozniak-koszalka, leszek.koszalka, andrzej.kasprzak}@pwr.edu.pl

***Abstract*—The objective of the paper was to determine which search method is suitable for implementation in GPS systems. The properties of pathfinding algorithms were tested and discussed taking into account this type of systems. Six algorithms have been evaluated, including three different implementations of Dijkstra algorithm, Bellman-Ford algorithm, A* star algorithm, and bidirectional Dijkstra's algorithm. Simulation experiments were carried out using the real digital maps and with the designed and implemented experimentation system. Studies were performed with respect to various parameters. After thorough examination and interpretation of conclusions, the algorithms which fit to GPS systems were selected.**

***Keywords*—GPS; search algorithms; experimentation system; path finding**

I. INTRODUCTION AND MOTIVATION

Nowadays we are living in the age where new technologies, innovations, great inventions such as the location with GPS have become an integral part of everyone's life. It is hard to imagine how the world could function without the GPS systems. The number of the GPS users is increasing very rapidly [1]. One of the major advantages of the GPS over the traditional searching the route to the target destination is the speed of action. GPS system consists of two basic components – digital maps and the shortest path search algorithm [2] and [3]. The ordinary user does not even realize that algorithmics surrounds him.

There are proposed in literature many search algorithms for solving pathfinding problem e.g., [1][3[4]. In some works, the properties of these algorithms are evaluated [4]. In this paper, the six search algorithms implemented by the authors aretested and evaluated. All the research was carried out on real digital maps. Based on the implemented experimentation system (programs in C++) and properly designed scenarios – the authors made a comparison of the results produced by these algorithms. The main objective of this paper was to find an algorithm that is most suitable for GPS systems.

The paper is organized as follows. In Section II, we formulate the formal model of the considered problem. Section III contains the description of the considered pathfinding algorithms. The experimentation system is presented in Section IV. The results of the simulation experiments are discussed in Section V. The final remarks, conclusion, and suggestions to further research in the area appear in Section VI.

II. PROBLEM STATEMENT

A. Mathematical model

The shortest path problem is an issue consisting in finding the shortest connection between vertices in the weighted graph [2]. There is a directed graph G (V, E), whereV denotes the set of vertices, E is the set of edges connecting vertices.

The weighted function $w: E \to R$ is assigning real-valued weights to the edges – the weights canbe interpreted as costs to traverse the edges. The total cost of a path $p = (v_0, v_1 \dots v_k)$ is defined by (1):

$$c(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{1}$$

The cost of the shortest path from u to$v$ can be expressed in the form given by (2):

$$\delta(u,v) = \begin{cases} \min\{c(p): u \xrightarrow{p} v, & \text{if path from } u \text{ to } v \\ & \text{exists} \\ \infty, & \text{otherwise} \end{cases} \tag{2}$$

The shortest path from u to$v$ *i*s each path *p from* u to v fulfilling the condition defined by (3):

$$c(p) = \delta(u,v) \tag{3}$$

More information about the shortest path problem is widely available, e.g., in [3] and [4].

B. Representation of the graph in a computer system

In the considered problem, the map can be represented as a huge graph [5].In such a model, the edges can be represented by means of roads and the vertices can be represented by intersections. In this work, a graph is represented by a list of incidence L. In the *n-th* list are stored incident vertices with the *m-th* vertex. Each element

 ISBN: 978-1-61208-398-8

of the list contains the ID of vertex, which is connected and a weight of a proper edge. The list has only as many elements as the number of possible connections for a given vertex what allows saving a lot of memory resources. This is the recommended solution for GPS system – it significantly increases the efficiency and speed of operations.

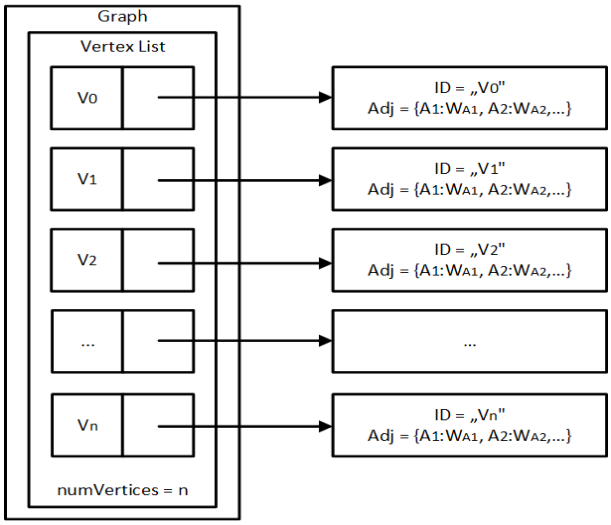

Figure 1. Representation of incidence list L.

This type of list is presented in Figure 1, where:

n – Number of vertices in graph,
ID – Given single vertex in graph,
Adj – Adjacency vertices to a given ID,
W – Weight of given adjacency vertex.

C. The weighted functions

The most commonly used weighted functions in GPS system correspond to the shortest route case and the fastest route case [6].

In the shortest route case, the weighted function is defined as the sum of the distances between successive intersections on the route. The total cost can be described by (1) in which $w(v_{i-1}, v_i) = d(v_{i-1}, v_i)$, where *d (u, v)* is the distance between the intersection *u* and *v*, i.e., by (4):

$$c(p) = \sum_{i=1}^{k} d(v_{i-1}, v_i) \quad (4)$$

In the fastest route case, the weighted function is defined as the sum of ratios of the distances between successive intersections and speed limits on the road between intersections. The total cost can be described by (5), where *s (u, v)* describes speed limit between *u* and *v*.

$$c(p) = \sum_{i=1}^{k} \frac{d(v_{i-1}, v_i)}{s(v_{i-1}, v_i)} \quad (5)$$

Search assumption is that that we are looking for the shortest or fastest route from the starting point (source) to the destination point – it is only one destination point. An additional assumption is that we are dealing with a map, so the edges in the graph do not have negative weights.

III. ALGORITHMS

A. Dijkstra's algorithm

Dijkstra's algorithm is used for solving a single source shortest path problem. It can find the path with the lowest cost between an initial vertex and every vertex in the graph. It can also find the shortest path from single vertex to single destination point by stopping the algorithm when the final shortest path was found – this way was applied in the designed experimentation system (simulator) allowing comparing this algorithm with the other considered algorithms. In our case, the graph should be directed, weighted and the edges should have non-negative weights. The basic idea of the algorithm lies in the fact that the information about predecessors is stored together with the information about the shortest path to a given vertex. In our implementation, we maintain a priority queue of vertices that provides three operations [6]and [7]:

Step 1. Inserting new vertices to the queue.
Step 2. Removing the vertex with the smallest distance.
Step 3. Decreasing the distance value of some vertex during relaxation.

Priority queue affects the performance of the algorithm in very large extent. For this reason, three different ways of implementing a queue were used what allow creating three versions of the algorithm.These versions are:

1) Priority queue as an array.
2) Priority queue as a binary heap.
3) Priority queue in the form of the Fibonacci heap [8].

B. Bellman-Ford

The Bellman-Ford algorithm solves the shortest path problem basing on the relaxation. The algorithm iteratively generates a better solution from a previous one until it reaches the best solution. It activates two loops, one running n-1 iterations and the other going through all edges. Bellman-Ford is slower than Dijkstra's in most cases but it can be more useful in some cases [3]. The details about this algorithm can be found, e.g., in [2] and [7].

C. Bidirectional Dijkstra's algorithm

This algorithm searches simultaneously from the source vertex (node) onward and from the destination vertex (node) backwards and stop when the two routes meet in the middle [9]. Searching the shortest path can be divided into two stages. When two paths meet at one vertex it is advisable to check whether the current path is the shortest. The principle consists of saving weight s of thefounded paths(in the

 ISBN: 978-1-61208-398-8

tables). If the weight is less than the sum of the values in the tables (for both instance of the algorithm) for vertices at the beginning of both priority queues – then the path is the shortest. Binary heap was used in our implementation as a priority queue. Searching from both the source and destination in a homogenous graph can reduce the search space to approximately half the size compared to only searching from the source.

D. A algorithm*

A star (A*) algorithm is a heuristic algorithm [7]. The algorithm allows findingan approximated solution - the least cost path from the start point to the destination point. A star uses a distance-plus-heuristic function*f(x)* to determine the order of visiting the nodes in a tree. This function can be defined by (6):

$$f(x) = g(x) + h'(x) \quad (6)$$

where *g(x)* – is the total distance it has to be taken to get the current position *x*, *h'(x)* – is the estimated distance from the current position *x* to the destination point. Such a heuristic is used to estimate on how far away it will take to reach the destination.

IV. EXPERIMENTATION SYSTEM

The experimentation system was created in order to properly investigate the properties of the pathfinding algorithms. It contains the programmed simulator with thesix implemented algorithms. It also ensures creation of the experimental scenarios – maps. Twoapplications (modules of the system) are available:

(I) *Real maps module*– allowing for creating a graph based on the loaded map, what gives possibilities of using the actual digital maps in order to reproduce the real conditions.

(II) *Arbitrary maps module*– allowing for generating an arbitrary created graph - with the chosen number of vertices, density, start point, number of iteration, etc. This part can be used to make simulation experiments in automatic way.

The block-diagrams of the functionality of these modules ofthe experimentation system are presented in Figure 2 (module I) and Figure 3 (module II).

Input problem parameters of the module I are:

U1 – Starting point as the node (*U*) marked on the map area.

U2 – Destination (End) point as the node (*V*) marked on the map area.

U3 – List of incidence *L* (see an example in Figure 1).

U4 – Weighted functions(weights assigned to the edges).

Output parameters of the moduleIare:

Q1 – Path found - denoted by $\delta(u, v)$- see formula expressed by (3).

Q2 – The total cost (length/time) of the founded route.

Q3 – The average execution time of the algorithm.

In both modules, the algorithm is treated as a special input. The experimentation system gives opportunities to use the following six algorithms:

- A star (A*),
- Dijkstra,
- Dijkstra binary,
- Dijkstra bi-directional,
- Dijkstra Fibonacci,
- Bellman-Ford.

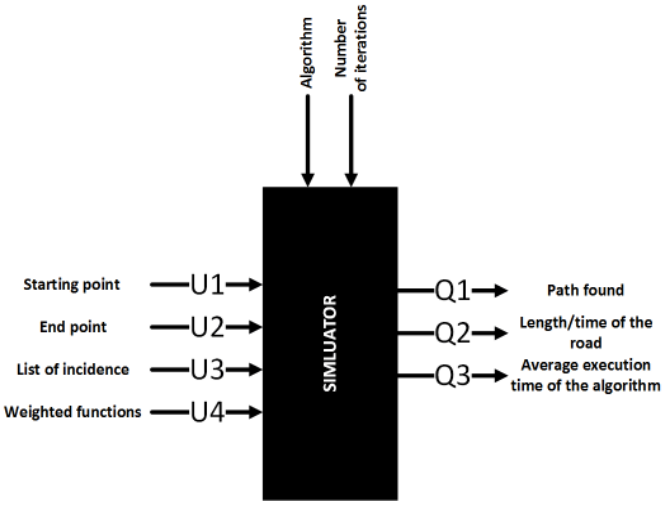

Figure 2. Input/output –module I - with real maps.

Input parameters of the module II are:

U1 – Starting point as the node (*U*) marked on the map area.

U2 – Destination (End) point as the node (*V*) marked on the map area.

U3 – Density of graph. This value expresses the amount of edges to remove (in relation to the full graph).

U4 – Hash - the value used for the pseudorandom number generator.

U5 – Weighted functions (weights assigned to the edges).

U6 – List of incidence *L* (see an example in Figure 1).

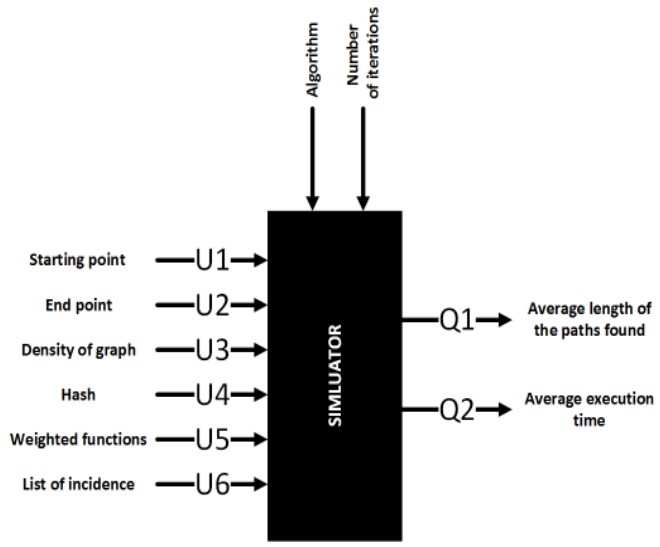

Figure 3. Input/output – module II – arbitrary maps.

Output parameters of the moduleII are:

- Q1 – The total cost (length/time) of the founded route.
- Q2 – The average execution time of the algorithm.

To investigate and compare the algorithms OsmGPS application was used. The tool has been implemented in C# environment. The main application window for the module I can be seen in Figure 4.

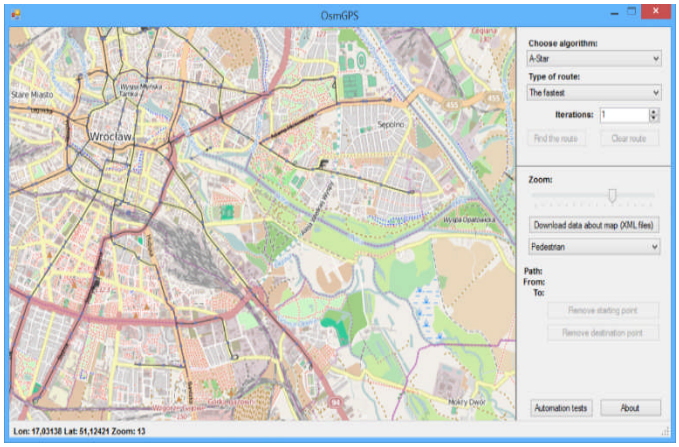

Figure 4. Main window - real map.

The main application window for the module II is shown in Figure 5.

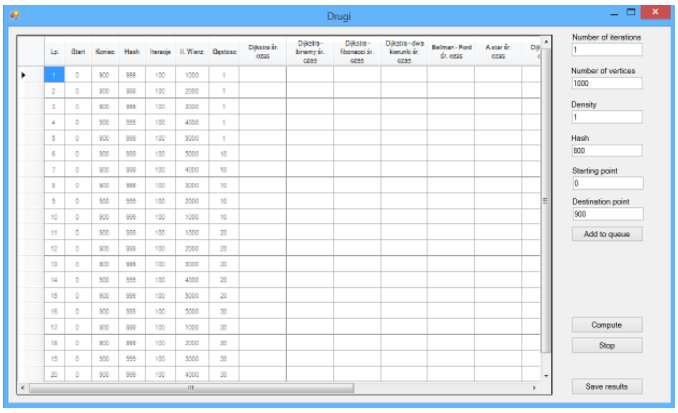

Figure 5. Main window–arbitrary map.

Both modules allow searching for the shortest and fastest routes. The experiments were made on MS Windows 8.1.

V. INVESTIGATION

A. *Experiment # 1*

The aim of Experiment #1 was checking the relationship between the execution time and the number of vertices in the graph. The locations of the starting points and destination points were always the same. Only the area of the map was changed(by increasing the number of vertices and edges).The results are presented in Figure 6 and Figure7.

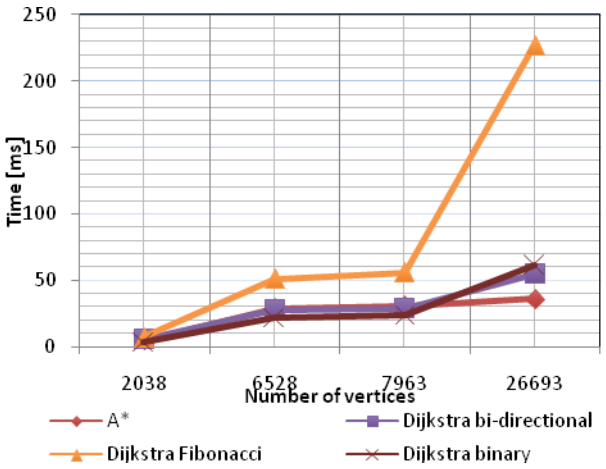

Figure 6. Execution time and number of vertices – the fastest algorithms.

 ISBN: 978-1-61208-398-8

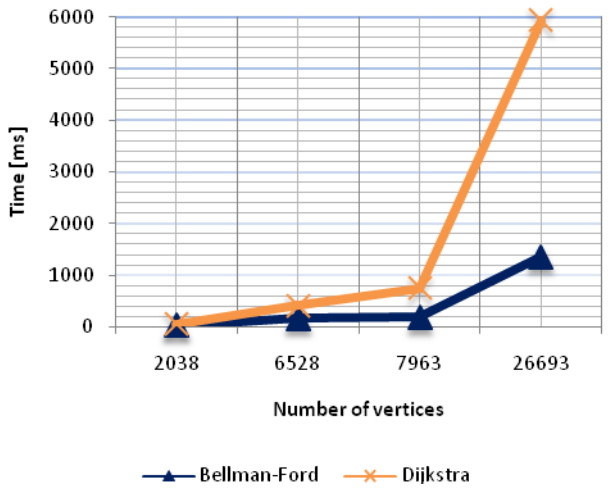

Figure 7. Execution time and number of vertices – the slowest algorithms.

As it can be seen in Figure 6, the A-star reached the lowest execution times. Dijkstra binary priority queue and Dijkstra bi-directional both reached a little bit worse times. The mentioned algorithms get the best times for graphs with high and low number of vertices. Two remaining algorithms (Bellman-Fordand Dijkstra) achieved much worse results.As expected, the execution timeswere raising a lot with increasing amounts of vertices in the graph. All the algorithms get similar execution time for graphs with less than the number of 6528 vertices.

B. *Experiment # 2*

The objective of this experiment was checking the relationship between the execution time and the path length. In the same area (constant number of vertices and edges) was set starting point – always the same (constant). The destination point was systematically moved away. The results are presented in Figure 8 and Figure9.

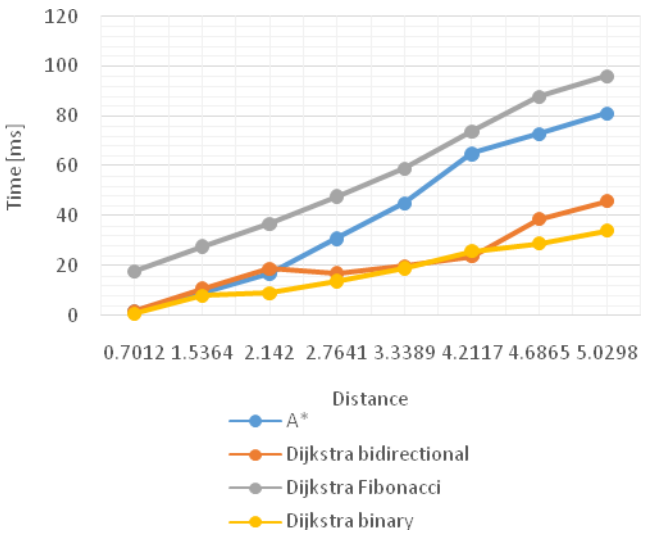

Figure 8. Execution time and path length – the fastest algorithms.

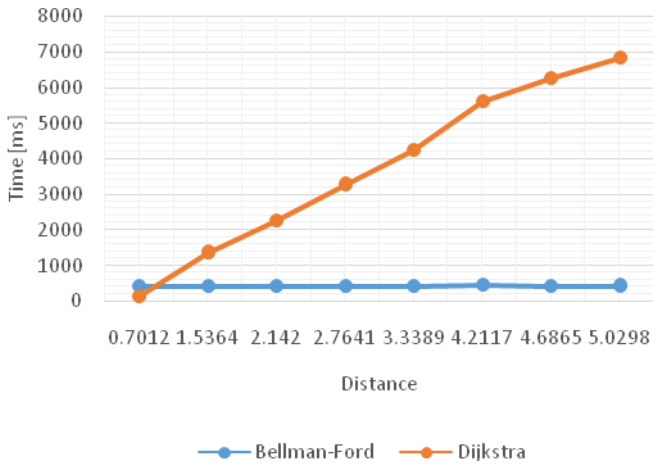

Figure 9.Execution time and path length – the slowest algorithms.

It can be seen in the figures that the execution time is growing steadily with the increasing distance. The Dijkstra based on binary queue and Dijkstrabi-directional were the fastest. For short distances, time differenceswere imperceptible. Only Dijkstra Fibonacci achieved worse results at each stage of the test. A-star and Dijkstra algorithm were apparently slower in comparison to others.

Bellman-Ford and Dijkstra again were the slowest of all. The execution time of Bellman-Ford algorithm was almost the same for different distances because of constant number of checks. This is due to the fact that algorithm searches paths to each of vertices and is not aborted before. The execution time for Dijkstra algorithm was constantly growing with increasing distance.

C. *Experiment # 3*

This experiment was a similar test to the previous experiment. It was performed for the area of Wroclaw city but this time the starting point and the end point were gradually moved close towards each other. Initially starting point was set in the southern part of the city and end point in the northern part of the city. In the middle of the founded route was the center of the city – a large collection of edges and vertices. The results are presented in Figure 10 (logarithmic scale).

 ISBN: 978-1-61208-398-8

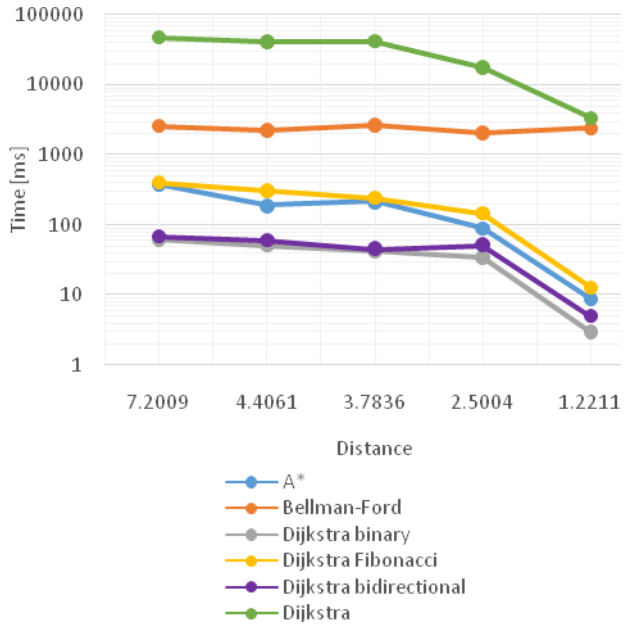

Figure 10. Relationship between execution time and distance – points are moved close towards each other.

It can be observed in Figure 10, that the best results have been produced by Dijkstra based on binary queue, Dijkstrabidirectional and A* algorithm. Very good performance of Dijkstra bidirectional should not be surprising in this test. This is due to the principle of the algorithm where it searches simultaneously from starting point onward and next from destination point backwards, and it stops when the two paths meet in the middle. Approaching the starting point and the destination point we facilitate the work of the algorithm. Once again the worst algorithms were: Bellman-Ford and simple Dijkstra.

D. *Experiment # 4*

The objective was to verify the performance of algorithms based on the graphs with given amounts of vertices, destination point, starting point, the density and the number of iterations. In this study, the attention was focused on the density of graphs. Tests were performed for different number of vertices but with the same conditions, e.g., for the same graph. The results are presented in Figure 11.

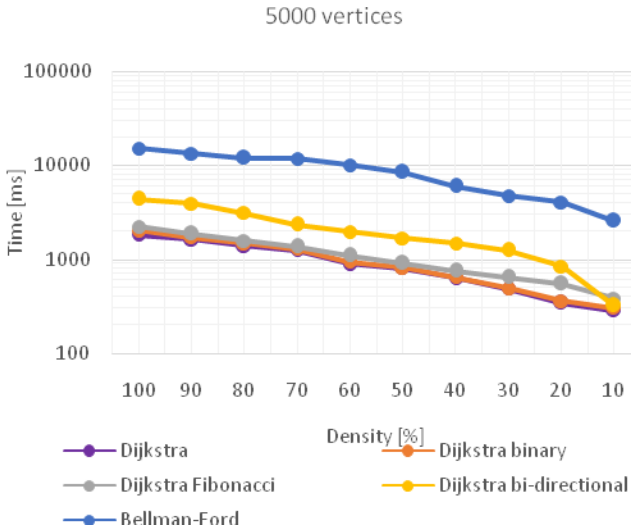

Figure 11. Relationship between density of graph and execution time - graph with 5000 vertices.

E. *Experiment # 5*

The aim of this experiment was toshow the relations between execution time and the number of vertices for 100% density.The results are presented in Figure 12.

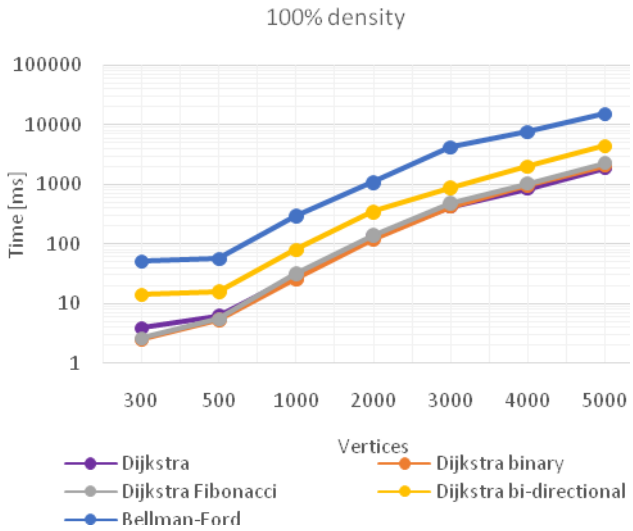

Figure 12. Relationship between vertices and execution time – density 100%.

Figure 12 demonstrates that the Dijkstra's binary achieved the best results for graphs with high density. Competitive for this algorithm were Dijkstra Fibonacci heap and simple Dijkstra algorithm.

VI. FINAL REMARKS AND PERSPECTIVES

Based on the simulation experiments, we may say that for GPS system the Dijkstra algorithm with priority queue in the form of the binary heap performed as the best. This algorithm achieved very good results almost in each

 ISBN: 978-1-61208-398-8

experiment – no matter how big the graph was. Also, easy implementation is a big strength of this method. However, Fibonacci heap definitely did not work well in graphs with high density. This is due to the structure of the heap (many nodes with pointers to the neighbors, parents, list of children). In binary heap we have only the array of elements. It is worth to mention that the bidirectional Dijkstra algorithm also achieved good results in many cases, in particular for graphs with low density. Using this algorithm can be recommended as a good solution in GPS systems because the maps have usually density at approximately a few percent levels.

The authors are planning to focus on the algorithms based partially on the evolutionary approaches, e.g., presented in [10] and [11]. There are also several interesting issues that might be considered in the future work, includingmore complex experiments with: more exact/heuristic algorithms, larger topologies, and detailed analysis of computational complexity of algorithms, as well as memory usage, following the ideas of multistage experiment design presented in [12].

ACKNOWLEDGEMENT

This work was supported by the statutory funds S40029_ K0402, Wroclaw University of Technology, Poland.

REFERENCES

[1] Royal Pingdom, “Google Maps turns 7 years old – amazing facts and figures” [retrieved: July, 2014] at :http://royal.pingdom.com.

[2] A. Kasprzak, Wide Area Networks, OWPW Publishing House, Wroclaw, 2001 /in Polish/.

[3] J. Larsen and J. Clausen, “The shortest path problem” [retrieved: February, 2015] at: http://imada.sdu.dk/~jbj/DM85/lec6a.pdf

[4] D. Johansson, “An evaluation of shortest path algorithms on real Metropolitan Area Network”, Linköpings Universitet, Report: SE-581 83, Linköping, Sweden, 2008.

[5] J. Koszalew, “Data structure for graph representation: selected algorithms,”[retrieved: July, 2014] at: http://asdpb.republika.pl.

[6] W. Lung and D. Tseng, “Graph theory: shortest path”, [retrieved: June, 2014] at: http://www.cs.cornell.edu/~wdtseng/icpc/notes.pdf.

[7] T. H. Cormen, Algorithms Unlocked, MIT Press, Cambridge, 2013

[8] K. Wayne, “Fibonacci heaps” [retrieved: January, 2015] at: http://cs.princeton.edu /~wayne/cs423/.

[9] G. Vaira and O. Kurasova, “Parallel bidirectional Dijkstra’s shorthest path algorithm”, Frontiers in Artificial Intelligence and Applications, vol. 224, 2011, pp. 422-435.

[10] T. Miksa, L.Koszalka, and A. Kasprzak,“Comparison of heuristic methods applied to optimization of computer networks”,Proc. of 11th Intern.Conf. on Networks, ICN 2012, IARIA, pp. 34-38.

[11] D. Ohia, L. Koszalka,and A. Kasprzak, “Evolutionary algorithm for solving congestion problem in computer network”, LNCS, Springer, vol. 5711, 2009, pp. 112-121.

[12] A.Kakol, I.Pozniak-Koszalka, L.Koszalka, A. Kasprzak, and K.J. Burnham, “An experimentation system for testing bee behavior based algorithm for a transportation problem”, LNCS, Springer, vol. 6592, 2011, pp.11-20.

 ISBN: 978-1-61208-398-8