# Software solution for optimal planning of sales persons work based on Depth-First Search and Breadth-First Search algorithms

**Conference Paper** · May 2016

DOI: 10.1109/MIPRO.2016.7522330

**3 authors**, including:

Emir Zunic
Info Studio d.o.o. Sarajevo
**59** PUBLICATIONS **193** CITATIONS
SEE PROFILE

Almir Djedović
University of Sarajevo
**16** PUBLICATIONS **51** CITATIONS
SEE PROFILE

# Software Solution for Optimal Planning of Sales Persons Work based on Depth-First Search and Breadth-First Search Algorithms

E. Žunić[*], A. Djedović[*], B. Žunić[**]
[*] Info Studio Ltd, Sarajevo, Bosnia and Herzegovina
[**] Goethe-Institut, Sarajevo, Bosnia and Herzegovina
emir.zunic@infostudio.ba

**Abstract - This paper presents and describes the practical usage of Depth-First Search and Breadth-First Search algorithms in the planning and optimization of sales persons work. Experiments for optimal implementation of these two algorithms for planning purposes are made through a specially developed MATLAB simulator. The application consists of two parts: Web application and Mobile application. Web application is developed using the Application Development Framework Technology (ADF), while for the development of the mobile version of application, Oracle ADF Mobile is used. Both applications are interactive with Google Maps, and based on the selected input parameters (such as the origin and destination of sales persons, the way of planning, any possible obstacles, etc.) they visually show the results, indicators and analysis. The commercial version of the software, eSalesmanPlan, which uses the aforementioned algorithms, is used in several companies in Bosnia and Herzegovina. All indicators point to significant savings both in human as well as financial resources and it will be also presented.**

**Keywords - Software, Simulator, Depth-First Search, Breadth-First Search, Sales Persons Planning**

## I. INTRODUCTION

The objective of this paper is to provide an introduction about graphs and the commonly used algorithms used for traversing the graph and their concrete practical usage. Breadth-First Search (BFS) and Depth-First Search (DFS) are the two popular algorithms asked in most of the programming interviews. Graphs are one of the most interesting data structures in computer science. Graphs and the trees are somehow similar by their structure. In fact, tree is derived from the graph data structure. However there are two important differences between trees and graphs:

- Unlike trees, a node can have many parents in graphs,
- The link between the nodes may have values or weights.

Graphs are good in modeling real world problems like representing cities which are connected by roads and finding the paths between cities (such as salespersons' problem which will be described in this paper), modeling air traffic controller system, etc. These kinds of problems are hard to represent using simple tree structures. A graph can be a directed/undirected and weighted/un-weighted graph. Every graph has two components, Nodes and Edges. This is the starting point for any software implementation of these algorithms. Nodes are implemented by class, structures or as Link-List nodes. Edges represent the connection between nodes. There are two ways to represent edges: Adjacency Matrix and Adjacency List.

The BFS and DFS are the two algorithms used for traversing and searching a node in a graph. They can also be used to find out whether a node is reachable from a given node or not. The aim of DFS algorithm is to traverse the graph in such a way that it tries to go far from the root node. Stack is used in the implementation of the DFS. The aim of BFS algorithm is to traverse the graph as close as possible to the root node. Queue is used in the implementation of the BFS. This is a very different approach for traversing the graph nodes.

As we can see, it is necessary to develop a programming environment which will use these algorithms to optimize the planning of commercial work. For this purpose, numerous tests of these algorithms have been performed in order to determine which algorithm in some situations shows better results. That is how a stimulator was created whose graphic results will be presented in the fourth chapter.

In the application of these algorithms, a big step was made using a concrete example of their application in solving one of the major problems in companies dealing with transport of goods with lots of commercialists working there. The problem is reflected in difficulties during the optimal planning of their everyday work. Therefore, we made the software which creates a presumption of optimal plan of commercialists' work on the basis of demonstrated knowledge and skills relating these two algorithms, as well as information about real-time locations. This software is being improved by adding new obstacles that could be found on optional path.

## II. DESCRIPTION OF ALGORITHMS

Breadth-First Search is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a “search key” [1]) and explores the adjacent nodes first, before moving to the next adjacent level.

BFS was invented in the late 1950s by E. F. Moore, who used it to find the shortest path out of a maze [2], and discovered independently by C. Y. Lee as a wire routing algorithm [3][4] (published 1961). BFS is one of the most popular graph algorithms, and forms the basis of a number of more advanced topics within graph theory. The main goal of BFS is to take a graph, denoted by $G = (V, E)$ where $V$ represents the vertices in the graph and $E$ represents the edges, and find the shortest path between the source vertex (analogous to the root node in the tree data structure) and all other vertices that are reachable from that source vertex. In this case, the shortest path is simply denoted as the number of edges travelled from the source vertex to the destination one. In doing so, BFS produces what is referred to as a breadth-first tree, where the source vertex is set as the root node and all reachable nodes are set as children of that root.

As indicated by its name, BFS works in a breadth-first manner, meaning that all reachable vertices at a specified distance, $k$, are discovered before all other ones reachable at $k + 1$. Algorithmically, this is done by “coloring” the vertices as they are visited. An unvisited vertex is denoted as a “white” one, vertices that have had all edges extended from them explored are marked as “black”, and vertices that have been visited, but whose edges have not all been explored, are denoted as “gray.” Therefore, a BFS algorithm allows for each reachable vertex in the graph to be explored, and the path between the source vertex and each reachable vertex to be tracked.

Depth-First Search is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking. A version of DFS was investigated in the 19th century by French mathematician Charles Pierre Trémaux [5] as a strategy for solving mazes [6][7].

DFS is similar to BFS. While the goals of each algorithm are the same, the ways we reach them are very different. As implied by its name, DFS searches as “deep” into a graph as possible before pulling back and exploring other paths. For example, if the source vertex has two adjacent edges, DFS will pick one and explore it fully before backtracking and moving on to the second.

DFS is also aided algorithmically by “coloring” each visited vertex, in the same manner as BFS. However, a vertex is not marked as being “black” until its adjacency list has been completely discovered and explored, which in turn can only happen when each vertex in that adjacency list has also had its adjacency list fully explored. Whereas BFS explores each current adjacency list at the same level, DFS picks one and explores it fully before moving on to the next one.

DFS has many attractive characteristics, such as the parenthetical structure property and the white-path theorem, each of which is useful when other algorithms attempt to extend the DFS algorithm.

## III. DFS AND BFS ALGORITHMS INSTRUCTIONS

DFS and BFS are common methods of graph traversal, which is the process of visiting every vertex of a graph [8]. Stacks and queues are two additional concepts used in the DFS and BFS algorithms.

A stack is a type of data storage in which only the last element added to the stack can be retrieved. It is like a stack of plates where only the top plate can be taken from the stack. The three stacks operations are:

- *Push* – put an element on the stack,
- *Peek* – look at the top element on the stack, but do not remove it,
- *Pop* – take the top element off the stack.

A queue is a type of data storage in which the elements are accessed in the order they were added. It is like a cafeteria line where the person at the front of the line is next. The two queues operations are:

- *Enqueue* – add an element to the end of the queue,
- *Dequeue* – remove an element from the start of the queue.

Considering a given node as the parent and connected nodes as children, DFS will visit the child vertices before visiting siblings using this algorithm:

```
Mark the starting node of the graph as visited and push it onto the stack
While the stack is not empty
    Peek at top node on the stack
    If there is an unvisited child of that node
        Mark the child as visited and push the child node onto the stack
    Else
        Pop the node off the stack
```

BFS will visit the sibling vertices before the child vertices using this algorithm:

```
Mark the starting node of the graph as visited and enqueue it into the
queue
While the queue is not empty
    Dequeue the next node from the queue to become the current node
    While there is an unvisited child of the current node
        Mark the child as visited and enqueue the child node into the
        queue
```

Examples of the DFS and BFS algorithms are given next.

### A. Examples of DFS Algorithm

An example of the graph which will be discussed in details is shown in Figure 1.

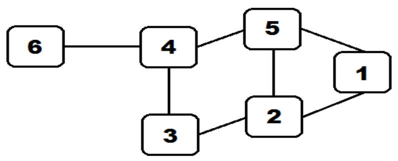


Figure 1. Example of the graph which will be analyzed

In the case of this graph, DFS algorithm will process the presented graph in the order shown in Figure 2.

| Action | Stack | Unvisited Nodes | Visited Nodes |
|---|---|---|---|
| Start with node 1 | 1 | 2, 3, 4, 5, 6 | 1 |
| Peek at the stack Node 1 has unvisited child nodes 2 and 5 | 1 | 2, 3, 4, 5, 6 | 1 |
| Mark node 2 visited | 1, 2 | 3, 4, 5, 6 | 1, 2 |
| Peek at the stack Node 2 has unvisited child nodes 3 and 5 | 1, 2 | 3, 4, 5, 6 | 1, 2 |
| Mark node 3 visited | 1, 2, 3 | 4, 5, 6 | 1, 2, 3 |
| Peek at the stack Node 3 has unvisited child node 4 | 1, 2, 3 | 4, 5, 6 | 1, 2, 3 |
| Mark node 4 visited | 1, 2, 3, 4 | 5, 6 | 1, 2, 3, 4 |
| Peek at the stack Node 4 has unvisited child node 5 | 1, 2, 3, 4 | 5, 6 | 1, 2, 3, 4 |
| Mark node 5 visited | 1, 2, 3, 4, 5 | 6 | 1, 2, 3, 4, 5 |
| Peek at the stack Node 5 has no unvisited children | 1, 2, 3, 4, 5 | 6 | 1, 2, 3, 4, 5 |
| Pop node 5 off stack | 1, 2, 3, 4 | 6 | 1, 2, 3, 4, 5 |
| Peek at the stack Node 4 has unvisited child node 6 | 1, 2, 3, 4 | 6 | 1, 2, 3, 4, 5 |
| Mark node 6 visited | 1, 2, 3, 4, 6 | | 1, 2, 3, 4, 5, 6 |

Figure 2. DFS algorithm steps

There are no more unvisited nodes so the nodes will be popped from the stack and the algorithm will terminate.

## *B. Examples of BFS Algorithm*

In the case of analyzed graph, BFS algorithm will process the presented graph in the order shown in Figure 3.

| Action | Current Node | Queue | Unvisited Nodes | Visited Nodes |
|---|---|---|---|---|
| Start with node 1 | | 1 | 2, 3, 4, 5, 6 | 1 |
| Dequeue node 1 | 1 | | 2, 3, 4, 5, 6 | 1 |
| Node 1 has unvisited children nodes 2 and 5 | 1 | | 2, 3, 4, 5, 6 | 1 |
| Mark 2 as visited and enqueue into queue | 1 | 2 | 3, 4, 5, 6 | 1, 2 |
| Mark 5 as visited and enqueue into queue | 1 | 5, 2 | 3, 4, 6 | 1, 2, 5 |
| Node 1 has no more unvisited children, dequeue a new current node 2 | 2 | 5 | 3, 4, 6 | 1, 2, 5 |
| Mark 3 as visited and enqueue into queue | 2 | 3, 5 | 4, 6 | 1, 2, 5, 3 |
| Node 2 has no more unvisited children, dequeue a new current node 5 | 5 | 3 | 4, 6 | 1, 2, 5, 3 |
| Mark 4 as visited and enqueue into queue | 5 | 4, 3 | 6 | 1, 2, 5, 3, 4 |
| Node 5 has no more unvisited children, dequeue a new current node 3 | 3 | 4 | 6 | 1, 2, 5, 3, 4 |
| Node 3 has no more unvisited children, dequeue a new current node 4 | 4 | | 6 | 1, 2, 5, 3, 4 |
| Mark 6 as visited and enqueue into queue | 4 | 6 | | 1, 2, 5, 3, 4, 6 |

Figure 3. BFS algorithm steps

There are no more unvisited nodes so the nodes will be dequeued from the queue and the algorithm will terminate.

## IV. RESULTS OVERVIEW

Development and implementation of the entire system consisted of several steps and stages. The first experimental part consisted of programming of analyzed algorithms as well as creating simulators which will visually present their application. MATLAB was used for these purposes. The input parameter for each algorithm is a matrix showing the very shape of the graph, as well as the definition of initial and final graph node. For example, the matrix with seven nodes will be shown as Figure 4 in simulator.

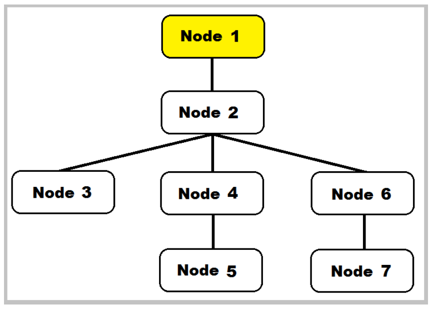


Figure 4. Simulator before algorithm search

DFS results will be shown here, since the procedure is identical for BFS too (they are complement). If in the following example the initial node is Node 1, and the ultimate one Node 7, then the steps will be as follows shown in Figure 5.

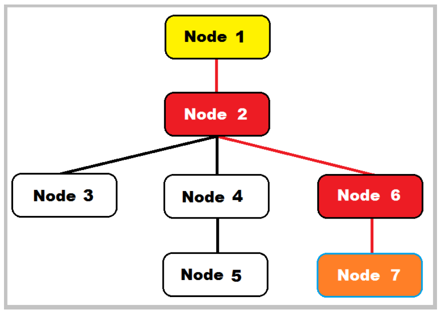


Figure 5. Simulator after algorithm search

Since this example is optimal and rarely used in practice, the more realistic problem containing certain obstacles from the initial to the ultimate node, as it is shown in the Figure 6, is being analyzed.

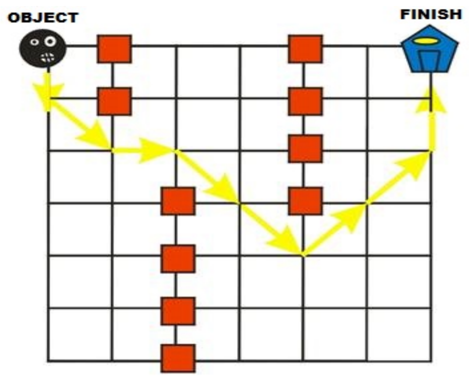


Figure 6. Example graph with obstacles

The object can move in paths north/west, west/east, and diagonally NW/NE/SW/SE in each node, or a section of the grid. The yellow line represents the path an object can achieve. The question is: How can an object reach the goal using the path bypassing the obstacles with? To solve this problem, we use DFS algorithm to produce the path of the object with given limitations. This example can be easily and in an identical way used during the movement and orientation of robots in the labyrinths. Solving this problem is of crucial importance, since the autonomous robots require this capability in planning their route.

Avoiding the obstacle is very important, especially if there is a possibility of robot damaging objects with strikes or vice versa. Programming of objects to move from the initial to the final location is usually more complicated than programming of rectilinear path, because there are obstacles between the initial and final point. There are several ways of avoiding obstacles used in generating the path, from the very simple ones of “changing the direction when reaching the obstacle” to using an A* algorithm which uses the information from the previous conditions and possible future ones for generating the path. DFS starts on the starting position of the object (the root node) and searches the target by expansion of all the following nodes (successors). The successors can only be those the object is allowed to go to, actually only those nodes that will not lead to the collision or removing the object from the working area. The Figure 7 shows all the directions the object can move in (8 directions).

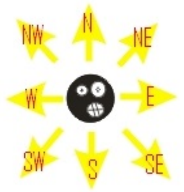


Figure 7. Directions the object can move

If DFS comes to the point where it is impossible to find successors for the expanding node, it will move to the latest discovered one that has not been expanded yet. DFS will be easier to understand if we look at the tree and use the LIFO (last in-first out) structure. The structure contains a list of new nodes. The newest one will be on the top of the list. The new expanding node is taken from the top of the list and all its successors have been added to the list. When the goal is reached, the searching of the graph can go back through the tree and determine the shortest path or the path of the robot in the case of labyrinth. The simulator has also been developed for such cases of finding the path from the initial to the final node if obstacles are included.

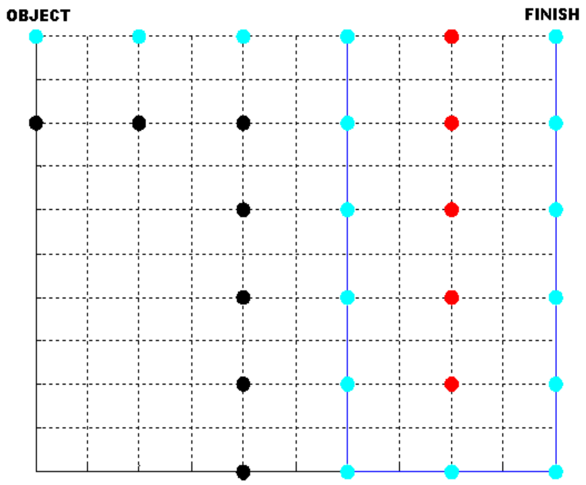


Figure 8. Simulator result with obstacles

The obstacles are presented by red dots, the searching limit by black ones, while the path from the initial to the final node is presented by cyan dots and the blue line.

The next thing we wanted to do is setting the obstacle in the way that algorithm reaches it, without any possibility of further progress and expansion, actually to be obliged to go back to the previous node and continue the searching of the space.

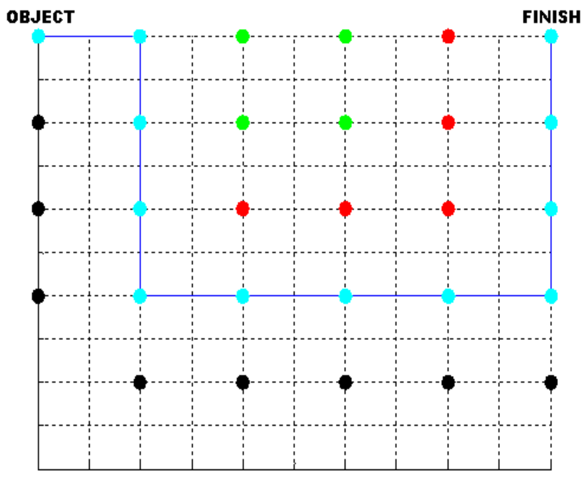
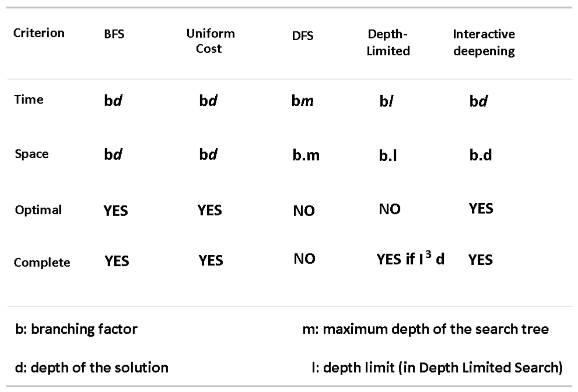


Figure 9. Simulator result with obstacles

The previous illustration shows the searching procedure and the path determination from the initial to the final node. We can also notice the green dots aside from the red, black and cyan ones, as well as the blue line. They explain that these nodes were notices during the search, but their further explanation was not possible because of the obstacle, so it was better to go back to some of the previous nodes in order to find the path to the target node. Initial simulator is made for the BFS algorithm also.

According to the searching graph results and finding the path from the initial to the final node and depending on the obstacles, a mechanism was made which, on the basis of comparative results of these algorithms determines which of them is better for an adequate situation. Criteria used for this purpose were the optimum path (fewer nodes) and the searching graph speed.



| Criterion | BFS | Uniform Cost | DFS | Depth-Limited | Interactive deepening |
|---|---|---|---|---|---|
| Time | b*d* | b*d* | b*m* | b*l* | b*d* |
| Space | b*d* | b*d* | b.m | b.l | b.d |
| Optimal | YES | YES | NO | NO | YES |
| Complete | YES | YES | NO | YES if $l^3$ d | YES |

b: branching factor — m: maximum depth of the search tree

d: depth of the solution — l: depth limit (in Depth Limited Search)

Figure 10. Comparative results between DFS and BFS algorithms

According to the numerous tests, BFS algorithm proved to be superior to the DFS. But in some situations DFS is better than the BFS. The results of the analyses are shown in the Figure 10.

Then, we started to create the application for the practical usage on the basis of all the results. Web application is developed using the Application Development Framework Technology (ADF) [9], while for the development of the mobile version of application, Oracle ADF Mobile is used [10]. Both technologies being used are based on Oracle ADF components, as well as Java EE. Both applications are interactive with Google Maps, and based on the selected input parameters (such as the origin and destination of sales persons, the way of planning, any possible obstacles, etc.) they visually show the results, indicators and analysis. After the users logs into the application, they can see the screen where they, with the help of interactive map, chooses initial and final point (starting point and the destination) and possible additional obstacles that might exist in the database. It is important to understand that there are stored predefined values (coordinates) of eventual obstacles (rivers, mountains, etc.). The third step important for user to do is choosing additional parameters, such as the way of planning (DFS, BFS, or the system will automatically determine which algorithm is more favorable) and perhaps storing of additionally added obstacles in the database and expand it for future usage. That is the way of improving and expanding knowledge base for better result of this software.

Once the user has finished the initializing, the mechanism of creating the graph starts over the input data. It is of great importance to know that initial and final point are placed on the edges of the graph (as it has been shown on the very simulator), and network is being created between the mentioned edges with extremely high density. The nodes of the graph are created in these horizontal and vertical sections. Simplified results (Banja Luka - Sarajevo) are shown in the Figure 11. Using this software, the system administrator has the ability to prepare the work plan for any worker (commercialist) and to send it via mail and/or push notifications to the one the plan refers to. With his user application, commercialist has an access in such prepared plans he can behave on.

The commercial version of the software, eSalesmanPlan, which uses the aforementioned algorithms is used in several companies in Bosnia and Herzegovina. All indicators point to significant savings both in human as well as financial resources and it will be also presented. Thus, in one company using this software during the 2015 financial expenses were decreased by 7%, compared to the period software was not used in. Saving in human resources being very visible is important to mention. According to some analyses, out of 100 commercialists, two to three persons could stop doing this amount of work by the implementation of this system. This s showed that the detailed analyses of the problem and usage of these algorithms can practically lead to favorable and successful results.

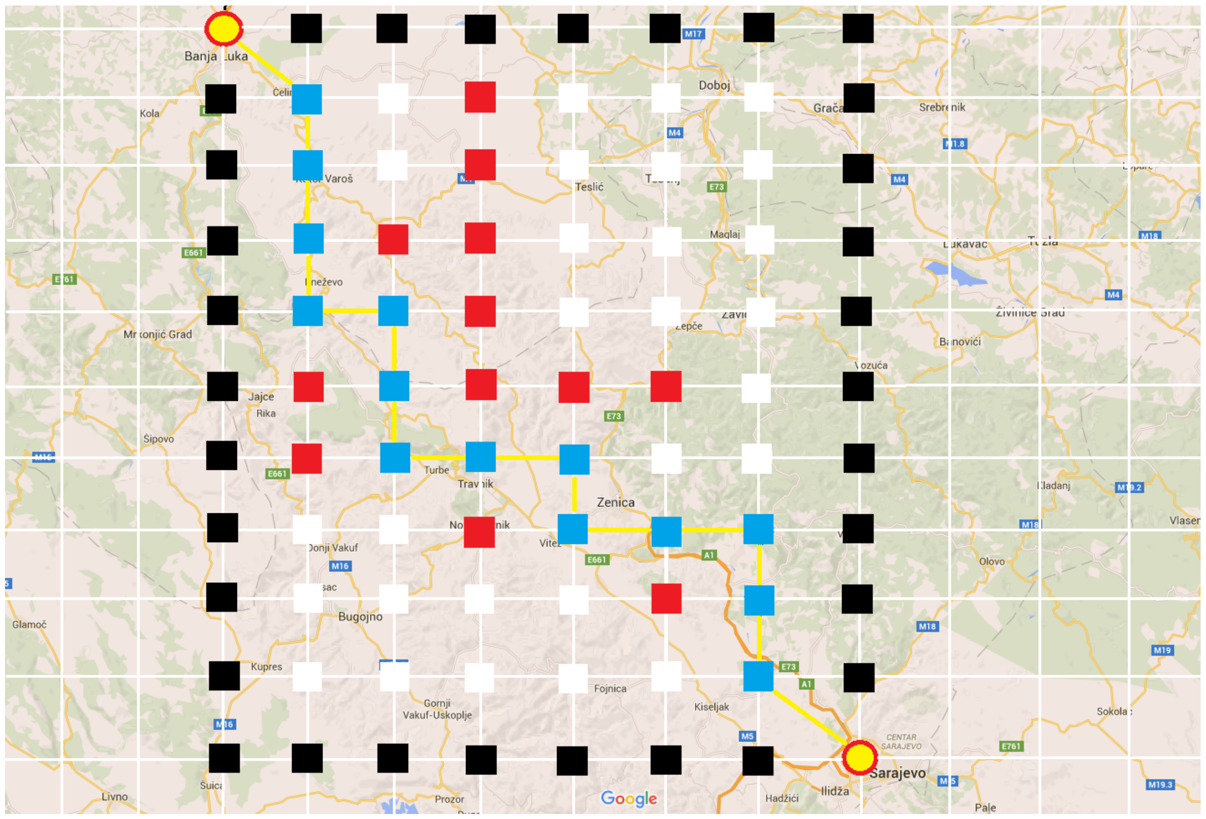


Figure 11. Web application, eSalesmanPlan, simplified result (Banja Luka – Sarajevo)

## V. CONLUSION

Depth-First Search and Breadth-First Search are search algorithms used for graphs and trees. When you have an ordered tree or graph, it’s quite easy to search the data structure to find the node that you want. But, when given an unordered tree or graph, the BFS and DFS search algorithms can come in handy to find what you’re looking for. The decision to choose one over the other should be based on the type of data that one is dealing with.

BFS and DFS both are techniques that use searching for some solution in a state space given. Both of these construct spanning trees with certain properties are useful in other graph algorithms. Besides, both of them are very useful for directed graphs. Depth-First Search is “any search algorithm that considers outgoing edges of a vertex before any neighbors of the vertex that is, outgoing edges of the vertex’s predecessor in the search. Extremes are searched first”.

During the very implementation of the software, the realization of the simulator was the first step to find out the comparative results for the two algorithms. Comparing BFS and DFS, the big advantage of DFS is that it has much lower memory requirements than BFS, because it’s not necessary to store all of the child pointers at each level. Depending on the data and what you are looking for, either DFS or BFS could be advantageous. The advantages of either vary depending on the data and what you’re looking for.

On the basis of observed findings, we started the realization of the application that could have a practical application. In accordance to everyday problems of many companies dealing with this kind of work, it was found out that this software could be created for optimizing commercialists’ work. It was done, so the applications (web and mobile ones) justified all the expectations and returned the invested funds within the first year of their application.

Beside the algorithms already mentioned, testing and using Dijkstra's algorithm for the problem is the next step in software development, directed graph and all weights must be non-negative It applies the greedy method pattern to the single-source shortest-path problem. For that reason it will be the next challenge in our next work.

## REFERENCES

[1] S. Skiena, “The Algorithm Design Manual,” Springer. p. 480, 2008.

[2] C. E. Leiserson, Schardl, Tao B, “A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers),” ACM Symp. on Parallelism in Algorithms and Architectures, 2010.

[3] C. Y. Lee, “An Algorithm for Path Connections and Its Applications,” IRE Transactions on Electronic Computers, 1961.

[4] S. Russel, P. Norvig , “Artificial Intelligence: A Modern Approach (2nd ed.),” Prentice Hall, 2003.

[5] C. P. Trémaux, “École Polytechnique of Paris (X:1876),” French engineer of the telegraph in Public conference, December 2, 2010 – by professor Jean Pelletier-Thibert in Académie de Macon (Burgundy – France) – Abstract published in the Annals academic, March 2011

[6] S. Even, “Graph Algorithms (2nd ed.),” Cambridge University Press, 2011.

[7] R. Sedgewic, “Algorithms in C++: Graph Algorithms (3rd ed.),” Pearson Education, 2002.

[8] A. Aziz, A. Prakash, “4. Algorithms on Graph,” Algorithms for Interviews, 2010.

[9] S. O’Brien, S. Shmeltzer, “Oracle Application Development Framework Overview,” An Oracle White Paper, June 2011.

[10] F. Desbiens, “Back-end to the Future: Using your Existing Oracle ADF Applications as a Pillar of your Mobile Strategy,” An Oracle White Paper, December 2013.