FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER EDUCATION ITMO UNIVERSITY

Report
on the practical task No. 6
"Algorithms on graphs.
Path search algorithms on weighted graphs"

Performed by

Rami Al-Naim

J4132c

Accepted by

Dr Petr Chunaev

St. Petersburg
2020

1 Goal

The use of path search algorithms on weighted graphs (Dijkstra's, A* and Bellman-Ford algorithms).

2 Formulation of the problem

A random adjacency matrix for a simple undirected weighted graph of 100 vertices and 500 edges with assigned random positive integer weights was generated. The Dijkstra's and Bellman-Ford algorithms were used to find shortest paths between a random starting vertex and other vertices. For each algorithm the average execution time was measured on 10 consecutive runs.

As a next step a 10x10 cell grid with 30 obstacle cells was generated. Two random non-obstacle cells we chosen and a shortest path between them was found using A* algorithm. The experiment was repeated 5 times with different random pair of cells.

3 Brief theoretical part

An weighted graph is a graph where each edge between the vertices has a weight assigned to it. The weight is expressed as a number.

Dijkstra's algorithm finds shortest path from the source vertex to all the others in the graph with positive weights. It builds a tree of shortest paths (or shortest path tree, SPT) by maintaining two sets: set with visited vertexes from the SPT and set with unvisited vertices. On each step it finds a vertex which from the unvisited tree with the smallest distance from the source node. The complexety of the algorithm can vary from $O(|V|log|V|)$ to $O(|V|^2)$.

The A* algorithm is an algorithm for finding path between two vertices in a weighted graph with positive edge's weights. On each step it decides the next visited node based on the distance from start and heuristic distance to the target node. The most popular heuristic functions for distance estimation are Manhattan distance and Euclidean distance. The time complexity of this algorithm is $O(|E|)$.

The Bellman-Ford algorithm can be used on a weighted graph with negative weights. Moreover, it detects negative cycles which means that there is no shortest path between starting vertex and any vertex in this cycle. It calculates the shortest paths between the vertices iterating over the edges of the graph. After relaxation stage ($|V|$ iterations in which shortest paths are computed) it iterates on the graph once more. If any of the path became shorter then there is a negative cycle present in the graph. This algorithm has computation complexity of $O(|V||E|)$.

This work was done using Python3.7 and mainly with module `networkx`.

4 Results

The random graph was generated using `networkx`'s random graph generator function (Fig. 1). A random starting vertex was chosen and each of Dijkstra's algorithm and Bellman-Ford's algorithm was used to find shortest paths to all other vertices in the graph. The implementation of Bellman-Ford's algorithm and Dijkstra algorithm from module `networkx` was used. The results of the comparison over 10 runs are presented in the Table 1.

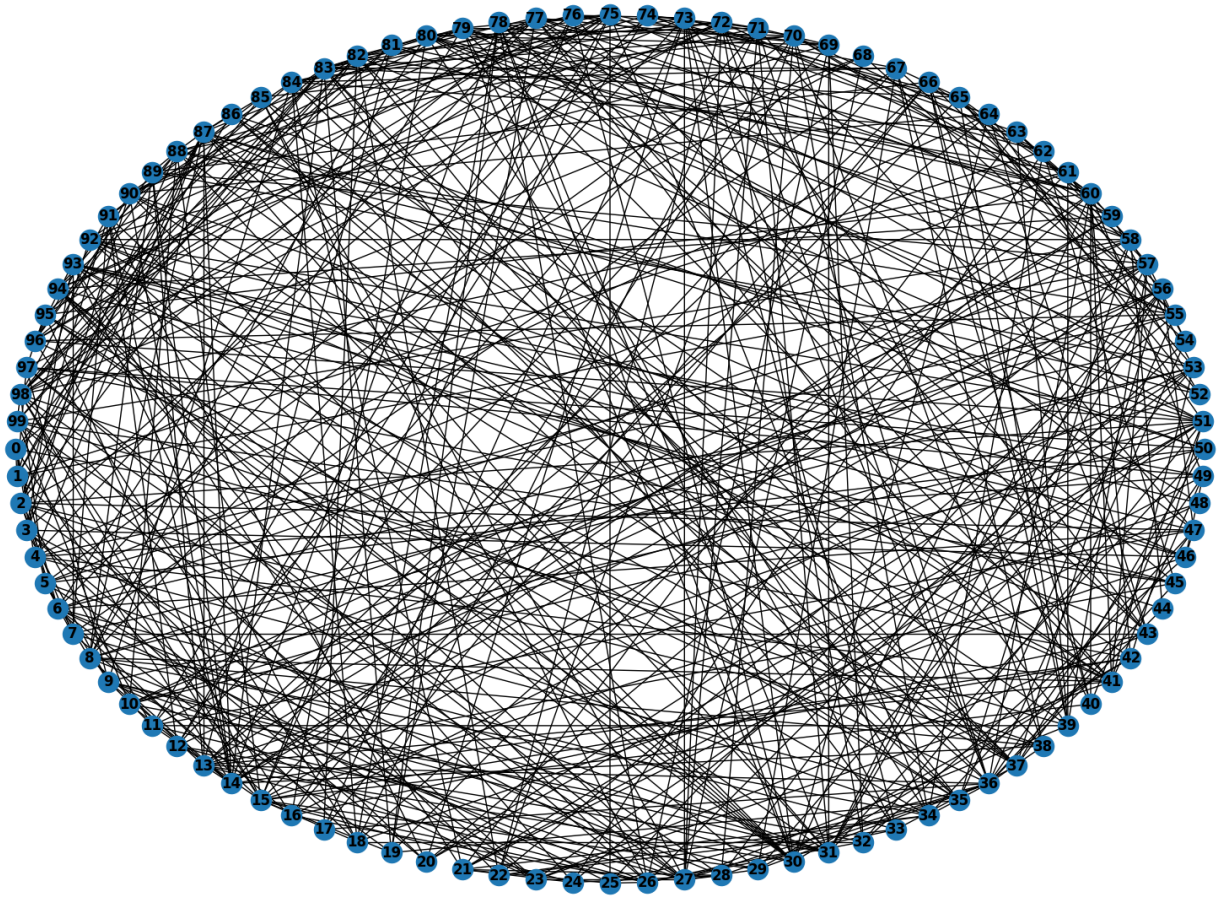


Figure 1: Generated graph with 100 vertices and 200 edges

Algorithm	Average execution time, ms
Dijkstra's algorithm	0.6
Bellman-Ford's algorithm	2.8

Table 1: Comparison of two algorithms

After completing the previous step, a 10 by 10 grid was generated and visualized as an image (Fig. 2). 30 obstacles were placed randomly on the grid. During this work the cross-accessibility was chosen (only tangent cells are connected, no diagonal connections). The A* search from module `networkx` was used. Two

heuristics were used: Manhattan norm and Euclidean norm. The result of five runs are present in Figure 4.

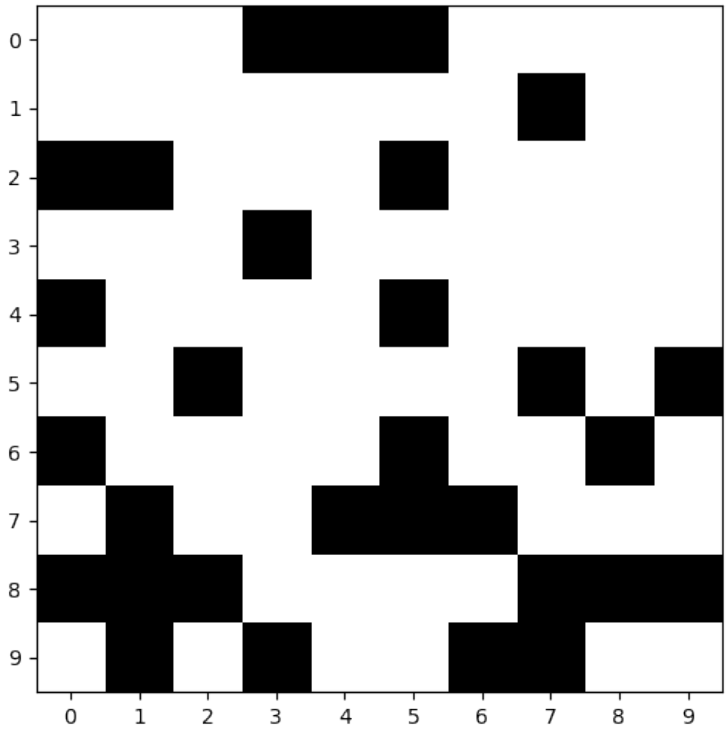


Figure 2: Generated graph with 100 vertices and 200 edges

5 Conclusions

During this work the graphs and the main algorithms on them were studied.

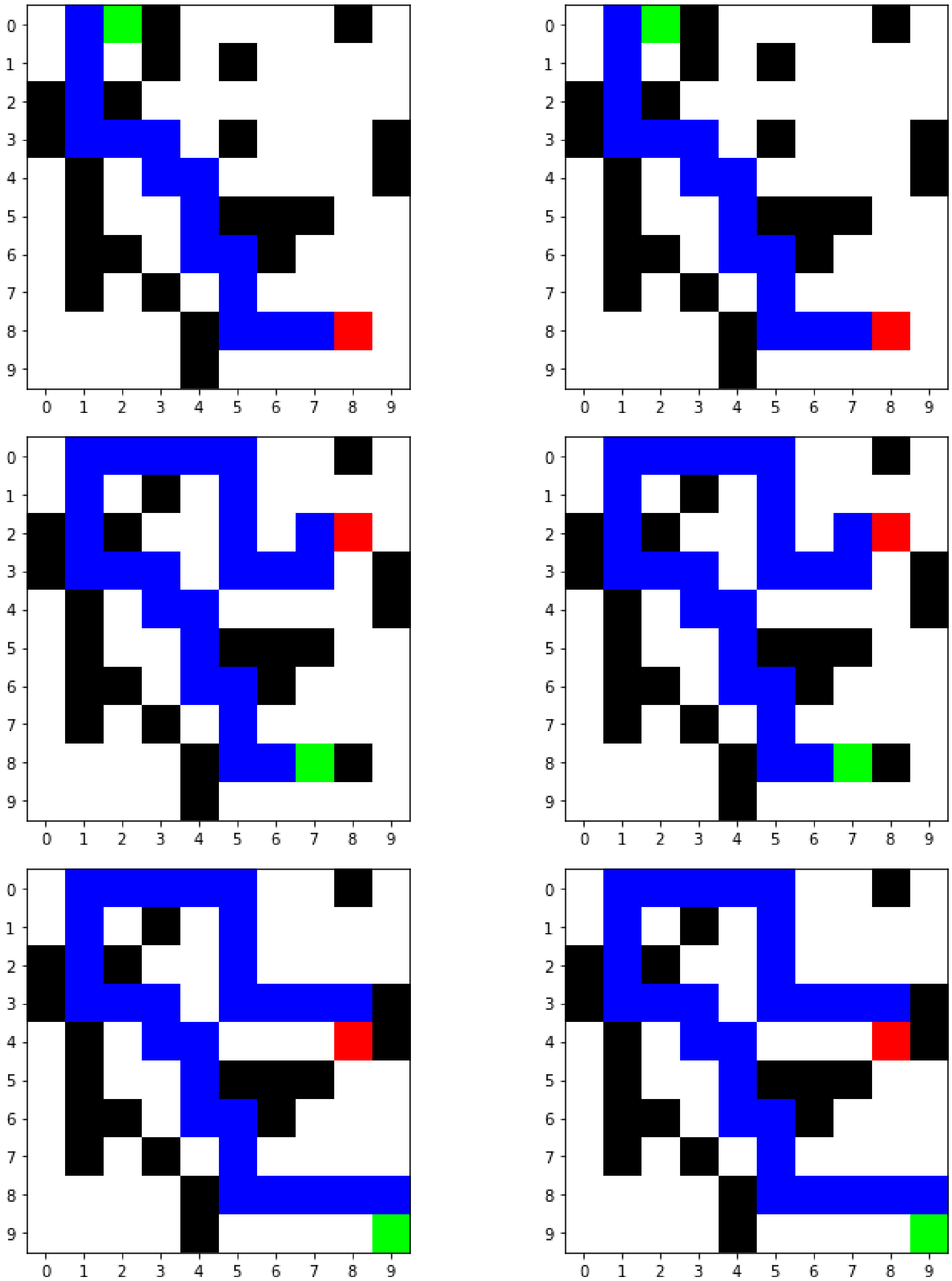
The random graph with 100 vertices and 200 edges was generated. The graph was visualized and two algorithms (Dijkstra and Bellman-Ford) were used for finding the shortest path from random starting vertex.

As it can be seen from Table 1 Dijkstra's algorithm is much faster since it is a greedy algorithm and it makes the best local choice. Bellman-Ford's algorithm is nearly 5 time slower, however, it can be used on graphs with negative weights and detect negative cycles which can be crucial in some cases.

The A* algorithm was tested on the randomly generated grid with obstacles. During this work the Manhattan and Euclidean metrics were used. From Figure 4 is can be seen that runs with different metrics and same starting and target vertices resulted in the same path. The explanation is that in this experiment it was prohibited to make diagonal steps. With allowed diagonal steps the result could be significantly different.

Appendix

The Python code for this task can be found here: GitHub.



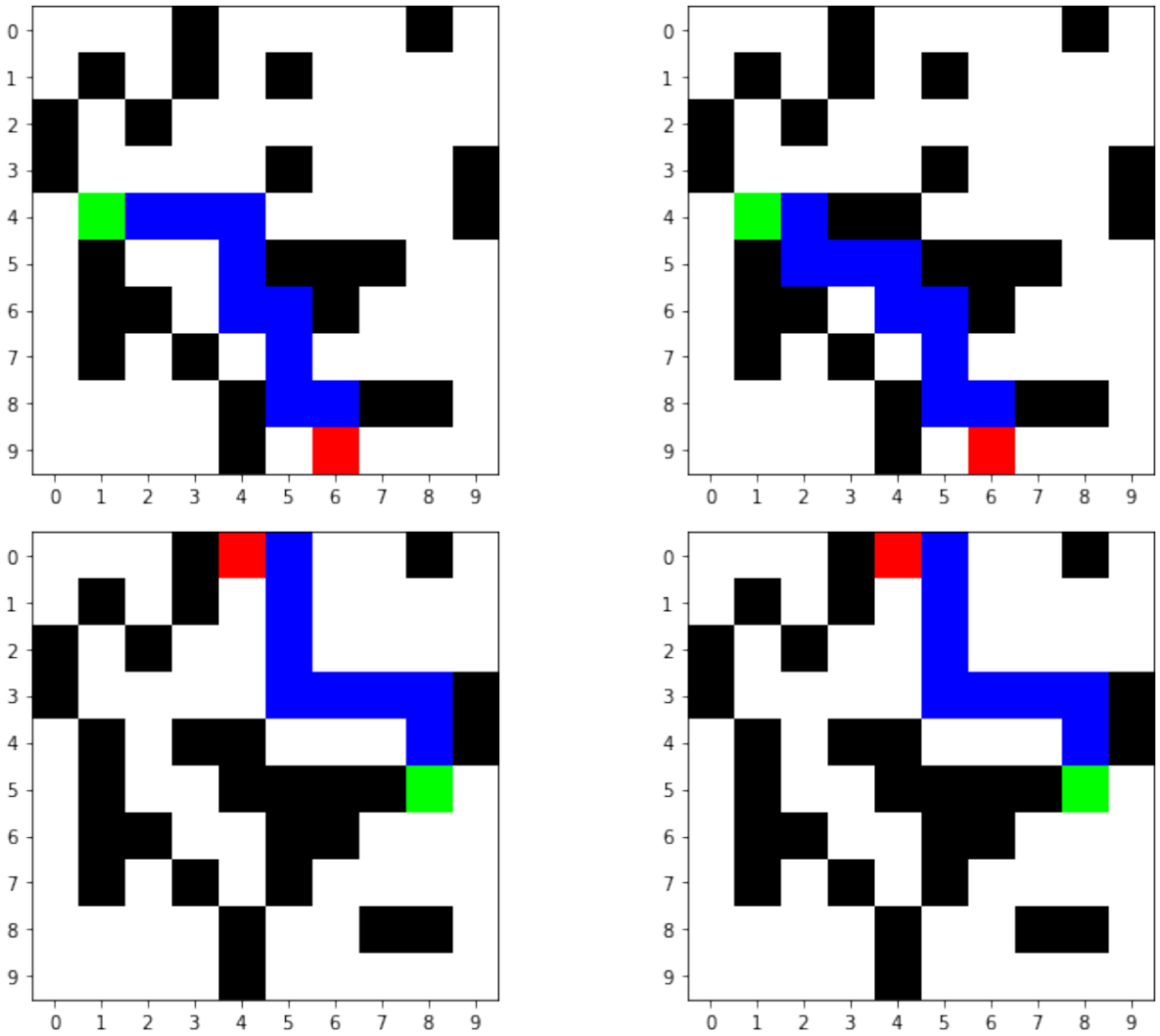


Figure 4: Results of A* algorithm: left column - Manhattan metric, right column - Euclidean metric. Green cell - start; red cell - finish, blue cells - path