Algorithms for Shortest Path on a Plane Including Obstacles with Cost

Yuma Hayakawa (1310057)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 12, 2015

Keywords: algorithms, shortest path problem, obstacles with cost.

The Fukushima first nuclear power plant caused the nuclear power disaster that had never been experienced due to the East Japan great earthquake and huge tsunami on March 11, 2011. The work in reactor building which was polluted by high density was demanded from robot on behalf of human, but it must work in the environment which there was a limit of the activity time by it because of a problem of the battery, the importance for robot to arrive at the destination by way of the shortest path was strongly recognized. The car navigation system for us to use when we go to the destination by car is very convenient for driver and reduces burden of driver. Before departure the car navigation system computes the shortest path from the current position to the destination and shows to driver it. Recently, the automatic driving that is controlled by artificial intelligence and drives without the control by driver is being developed. Therefore the necessity of instant calculation for the shortest path increases.

It is discussed in computational geometry to compute the shortest path for finding the shortest path on a plane including obstacles. Given some polygonal obstacles on a plane, we want to find the shortest path between start point and destination point.

Recently, finding a shortest path in a big graph is required as society becomes complex. However, it requires huge memory proportional to the size

of the graph when we use a computer to solve it. Mitchell and Papadimitriou [**2**] suggested an algorithm for the shortest path that can compute $O(N^8)$ time using $O(N^4)$ workspace given the polygon with cost, start point, and target point, Where N is the total number of the vertices of the obstacles. But the algorithm is complicated, and computation time is long. In this paper, we propose an efficient and practical memory constraint algorithm that solves the shortest path problem on a plane.

When obstacle has its cost a mobile robot can pass it with paying the cost. Even if there is an obstacle at the middle of the course, the mobile robot chooses the course of the smallest cost considered by distance and the size of the obstacle. For example, we assume that the path a mobile robot chooses in the environment that there exists a pond on the straight line between start and destination. When we suppose it is difficult for mobile robot to pass a pond, we think that the mobile robot chooses the other path excepting the path to pass a pond and go straight to destination. we suppose in the other paths there are two paths, one path is that there doesn't exist any obstacle in the course, but it is detour path, the other path is that the mobile robot is burdened by weed of obstacle at the middle of the course, but distance is shorter. In that case, we think that the mobile robot chooses the path to pass weed. To simulate this path choice, we treat the pond is a big cost because the pond is big obstacle for a mobile robot and the weed is a small cost because the weed is small obstacle. The distance is defined by the total of geometrical distance by cost of obstacles. The shortest path for it of the paths we explained is the path to pass a weed. Therefore the simulation choose the path passing a weed and output it. Because we assume the case to choose such path, in this paper we solve the shortest path problem on a plane which considered the cost of obstacles.

In this paper, we use Asano and Doerr's [**1**] algorithm to obtain a space efficient algorithm. The sketch of the Asano and Doerr's [**1**] algorithm is as follows. First, the given grid graph is partitioned into some small grid graphs. The vertices of each small grid graph is called as *boundary vertices*. First, Dijkstra's algorithm runs on all vertices within each small grid graph independently, and computes each shortest path from starting point. In the first Dijkstra's algorithm running, all shortest path distances are undefined except for the vertices of small grid graph including starting

point. To use less memory space, we hold the shortest path distances for only boundary vertices. Such Dijkstra's algorithm running is called as *scan*. The scan for each small grid graph is repeated at most the number of all boundary vertices.

The next step is reporting a shortest path between start vertex and end vertex on given grid graph. First, we apply Dijkstra's algorithm for the small grid graph including end vertex, we compute the shortest distance between vertex on small grid graph and end vertex. Some of the boundary vertices which is memorized the shortest distance from start vertex, we call one *via-vertex*. The vertex belongs to small grid graph including end vertex and belongs to the small grid graph neighboring it. We determine the path between end vertex and via-vertex, and we execute Dijkstra's algorithm for the other small grid graph belonged to via-vertex, and we compute the shortest distance between via-vertex and the vertex on the small grid graph. We repeat this process between end vertex and start vertex, and we can report shortest path between start vertex and end vertex.

In this paper, we make the grid graph including diagonal on a plane at first. Therefore, we can report the path passing on a plane easily. We don't make entire grid graph on a plane, we appropriately make grid graph of the part in the track of computation. Therefore we can expect the memory-saved algorithm.

We make a grid graph of the part, we determine that the vertex belong to which polygonal inside, we compute the cost of vertex. And, we can compute the cost of edges between the neighbor vertex if we use the cost of vertex which is computed already. Therefore we can implement the path considered cost of obstacle. We use the algorithm by Asano and Doerr [**1**], and compute the shortest path. Because we manage the distance information for boundary vertices only, we can compute the shortest path between start vertex and end vertex with memory-saved algorithm.

However, the path computed with algorithm of this study is approximate solution of the real shortest path. When we show the shortest path visually, we choose the path that directions change many times.

We performed experiments with computer to inspect the algorithm which can compute the solution approaching the algorithm strictly could compute. This algorithm in this paper can output approaching path if we

increase the size of the grid, but the algorithm uses much memory. Therefore at first we performed the experiment to investigate the size of the grid which could output approaching real path. We set the experiment environment where we knew approaching the real shortest path, we investigated the size of the grid when we output approaching real path. we performed experiment to inspect path change if we change the cost of obstacle. We set the experiment an environment with the obstacle having cost on the course on the straight line between start point and end point, we investigated output path.

By a computer experiment this algorithm can compute the practical shortest path. This algorithm runs in $O(n^{\frac{2}{3}})$ space and $O(n^{1+\frac{2}{3}} \log n)$ time. However, this algorithm computes the shortest path without using the complicated algorithm, the path is on a grid. To approach the real shortest path, we need more memories. Future issue is to find the algorithm which is not complicated and to find the algorithm could computed highly accurate the path on a plane.

References

[1] T, Asano. and B, Doerr., "Memory-Constrained Algorithms for Shortest Path Problems", CCCG, 2011.

[2] J. S. B. Mitchell and C. H. Papadimitriou, "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision", JACM, Volume 38 Issue 1, pp. 18-73, Jan. 1991.