Theseus - Interactive Mobile and Web Application for finding the shortest path in a simple polygon

Francisc-Sebastian Kalciov
Department of Computer Science,
West University Timișoara
Email: francisc.kalciov04@e-uvt.ro

January 29, 2025

Abstract

Finding the shortest path in a simple polygon is a fundamental challenge in computational geometry, where the objective is to find the minimum length path between two points that are placed entirely inside a given simple polygon (without holes).

In this paper, I have showcased the practical part of finding the shortest path in a simple polygon drawn on a paper and the theoretical aspects that create the foundation.

Contents

1 Introduction

Motivation

The problem of finding the shortest path in a given simple polygon is a cornerstone of computational geometry, with applications in various fields [1]. At the low level, the problem involves finding the most efficient route between two points inside the polygon without crossing its edges.

Informal example

Let's imagine that we have a robot in a empty house, with multiple rooms. We place the robot in Room A and we want it to reach Room B, that is, 3 walls away, as efficient as possible.

To do it without breaking all of its hardware components, it has to have some instructions to avoid collision with the walls.

Those instructions are part of the shortest-path problem.

Declaration of originality

My contributions to solving the problem of finding the shortest path are related to having an application that interconnects 2 devices, implementing the infrastructure of the application, implementing algorithms that are necessary to process the raw image in a digital format to be later used to complete the shortest path finding.

2 Implementation

The flow of the application is split into 2 devices: the first part happens on the mobile phone, where the picture is captured, the second part takes place on the computer, in the web browser, where the image is retrieved, processed and animated on the screen.

2.1 Technical details

The mobile application is developed in React Native [3] using Typescript.

For cloud storage, I used Firebase [2]

For Web application: The client side is developed in React [4], the server side is using Express [5]

2.2 Image transfer and image processing

To send the image from the mobile phone to the web application, I have used the Google Cloud service, Firebase.

To process the image, OpenCV framework functions have been used in Python [6].

2.3 Graph computation and triangulation

To compute the graph, I first triangulated the processed polygon with the starting and ending point using Delaunay triangulation [8]:

- Input: The algorithm takes a set of 2D points as input.
- Convex Hull Construction: The convex hull of the points is computed using the Quickhull algorithm, starting with an initial simplex and recursively dividing points into subsets based on their position relative to the current hull.
- Triangulation: The lower facets of the convex hull correspond to the Delaunay triangulation in 2D.
- Circumcircle Test: The algorithm ensures that no point lies inside the circumcircle of any triangle. If this condition is violated, edge flipping is performed to restore the Delaunay property.
- Output: The triangulation is finalized as a set of triangles, stored as simplices with adjacency and connectivity information.

After obtaining the triangulation, I constructed a graph representation where the nodes correspond to the triangle vertices, and the edges connect adjacent vertices. This graph serves as the foundation for building the shortest path using Dijkstra's algorithm [7].

2.4 Shortest path computation

To compute the shortest path from the graph generated from the processed image, I went with Dijkstra's algorithm because it fits well with the type of data extracted from the image (large numbers). Below is step-by-step explanation of the Dijkstra algorithm implementation [9] [7]:

- Initialization: The algorithm starts with a priority queue containing the start node with a distance of 0. A dictionary stores the shortest known distances to all nodes, initialized to infinity except for the start node. Another dictionary keeps track of the previous node for each node in the shortest path.

- Processing Nodes: The node with the smallest distance is extracted from the priority queue. If its current distance is greater than the stored distance, it is skipped.

- Updating Neighbors: For each neighbor of the current node, the algorithm calculates the new distance. If this distance is shorter than the previously known distance, it updates the shortest distance and records the previous node. The neighbor is then added to the priority queue.

- Path Construction: Once all nodes are processed, the shortest path is reconstructed by backtracking from the end node using the previous dictionary.

- Output: The function returns the shortest path as a list of nodes from start to end.

2.5 Data transfer from server to web and animation

The image processing, graph and path computation take place on the Express Server, then to be animated on the Web, the data had to be converted into JSON format, then sent to the web to be animated using D3.js [10] JavaScript Framework.

3 Problems

The main problem of the application is the image processing. The algorithm that I created to validate the polygon extracted from the image has flaws. If the drawn polygon does not have a margin of at least 5-6cm, then it might consider the edge of the paper to be a polygon, and then wrongly consider the polygon itself and the starting/ending points.

It is possible for the algorithm to fail for a random shaped polygon, without having any issues due to limited abilities of the OpenCV framework.

4 Conclusions and future work

To summarize, in this paper I addressed the technical details and the theoretical parts of finding the shortest path in a simple polygon.

For future work, to avoid the problem of failed image process, I have to use Machine Learning to develop a model that is trained on polygons, such that it returns only the polygon, without having to implement an algorithm to carefully extract the polygon without the noise.

References

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry: Algorithms and Applications

[2] https://firebase.google.com/

[3] https://reactnative.dev/

[4] https://react.dev/

[5] https://expressjs.com/

[6] https://docs.opencv.org/4.x/index.html

[7] Muhammad Adeel Javaid. Understanding Dijkstra's Algorithm

[8] O'Rourke, J., Computational Geometry in C, Cambridge University Press, 1994

[9] https://github.com/sebastiankalciov/Theseus

[10] https://d3js.org/