TIA Project Report

on

Shortest Path in a Polygon

Submitted in partial fulfilment of

the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

Submitted to

Dr.Subhasree

Submitted by

SHAHIN JOHN JS

B160943CS

Department of Computer Science and Engineering

National Institute of Technology Calicut

Calicut, Kerala, India - 673 601

Winter Semester 2018

Department of Computer Science and

Engineering

National Institute of Technology Calicut

Certificate

This is to certify that this is a bona fide record of the project presented by

the students whose name is given below during Winter Semester 2018 in

partial fulfilment of the requirements of the degree of Bachelor of Technology

in Computer Science and Engineering.

Roll No: B160943CS

NAME:SHAHIN JOHN JS

Dr.Subhasree M

(Course Coordinator)

Date:

**Abstract**

Let p and q be two points in a simple polygon P. An open problem in computational geometry asks to devise a simple linear-time algorithm for computing a shortest path between p and q, which is contained in P, such that the algorithm does not depend on a (complicated) linear-time triangulation algorithm. This report provides a contribution to the solution of this problem by applying a different algorithm. The obtained solution has O(n2) time complexity. It has applications in 2D pattern recognition, picture analysis, robotics, and so forth.

Introduction

The Euclidean shortest path problem is a formal version of an everyday question: what is the shortest route for moving an object from one place to another? The problem specifies the object to be moved, a set of obstacles, and a pair of locations. A solution to the problem is a minimum-length path for the object that connects the two locations and avoids the obstacles. There are many possible versions of the problem; this paper focuses on what is perhaps the simplest: mooing a point inside a simple polygon in the plane.

Several algorithms have been proposed to find shortest paths inside a simple polygon with n vertices. All the methods are based on a triangulation of the polygon. The algorithm of Lee and Preparata finds the shortest path between two points inside a simple polygon in linear time, once a triangulation is known [LPI. Reif and Storer’s [RS] approach uses precomputation to speed up queries. Given a source point inside the polygon, their method produces a search structure so that the distance from a query point to the source can be found in O(log n) time. The shortest path itself can be obtained in time proportional to the number of turns along it. Reif and Storer’s method uses the Delaunay triangulation of the polygon and hence takes O(n log n) preprocessing time. OPTIMAL SHORTEST PATH 127 The sub-linear query algorithms mentioned above pre-select a fixed source point. If neither endpoint of the path is predetermined, each of these algorithms takes linear time to find the shortest path (or even its length).

Pseudocode:

The input points are got from an user interface provided by QT and stored inside list Polygon\_list and the user shall enter the starting point and the terminating point in the terminal. These are all done in the main function and the main calls SPP function which returns the optimal path from source to destination.

SPP (List Polygon\_list, Point start\_point, Point end\_point)

{

List L, R, Reduced\_L, Reduced\_R;

L = Getleftchain(Polygon\_lsit, start\_point, end\_point);

R = Getrightchain(Polygon\_lsit, start\_point, end\_point);

Reduced\_L = GetReduced\_left\_chain(L, Polygon\_list);

Reduced\_R = GetReduced\_right\_chain(R, Polygon\_list);

}

**Implementation:**

First, we need to get the two list, which represents the left and right chain starting from the starting point and ending at the terminating point. The left chain contains all the points from starting point, covering all the points in the polygon in a clockwise direction until the terminating point. Similarly, the right chain contains all the points from starting point, covering all the points in the polygon in a counter clockwise direction until the terminating point. This is done by the following code.

Input : A List containing all the nodes in the polygon in clockwise order.

Output : Two Lists representing the left and right chains of the polygon.

Iterator startit,finishit,lit,rit,tempit,nextit,preit,pit,ppit;

std::list<Point\_2> list\_left,list\_right,short\_left,short\_right;

std::list<Point\_2>::iterator bit,git,ppit1;

int flag = 0,start\_check = 0,finish\_check = 0;

for(it = b; it != e; ++it)

{

++it;

if (it == e)

break;

else

--it;

if((x1==(it->x())) && (y1==(it->y())))

{

std::cout << "hi";

flag = 1;

start\_check = 1;

startit = it;

}

else if((x2==(it->x())) && (y2==(it->y())))

{

std::cout << "ho";

flag = 2;

finish\_check = 1;

finishit = it;

}

else if(flag == 1)

list\_left.push\_back(\*it);

else if(flag == 2)

list\_right.push\_front(\*it);

}

if(!(start\_check && finish\_check))

{

std::cout << start\_check << finish\_check;

std::cout<<"The points entered are not on the polygon"<<std::endl;

return;

}

it = b;

while(\*it != \*startit)

{

list\_right.push\_front(\*it);

++it;

}

list\_left.push\_front(\*startit);

list\_right.push\_front(\*startit);

list\_right.push\_back(\*finishit);

list\_left.push\_back(\*finishit);

This operation can be carried out in time linear in the size of the input list, which is the number of the points on the polygon.

Next, we remove all the unwanted points in the polygon and shrink it. So that we can work on a reduced space. The new space is equivalent to the old one in the sense that optimal solution to both the spaces are the same. The idea of this function is to remove points that are trivially not needed in the computation of the shortest path starting from starting point to the terminating point.

Input : A list representing the left chain.

Output : A list representing the reduced left chain.

bit = list\_left.begin();

do{

if(short\_left.size() < 3)

{

short\_left.push\_back(\*bit);

}

else

{

while(1)

{

it = short\_left.end();

tempit = --it;

pit = ----tempit;

CGAL::Segment\_2 <K> s1(\*pit,\*it);

q = \*it;

std::cout <<"s1-"<< s1<<std::endl;

flag = 0;

for(git = b; git != e; ++git)

{

Point\_2 pointz;

CGAL::Segment\_2 <K> segmentz;

tempit = git;

++tempit;

if(tempit == e)

{

flag = 0;

break;

}

CGAL::Segment\_2 <K> s2(\*git,\*tempit);

std::cout<<"s2-"<<s2<<std::endl;

CGAL::Object result = CGAL::intersection(s1,s2);

if(CGAL::assign(segmentz, result))

{

flag = 1;

break;

}

if (CGAL::assign(pointz, result))

{

if(pointz == \*pit)

{

std::cout << "pit-"<< \*pit<<std::endl;

if (pit == short\_left.begin())

{

ppit1 = list\_right.begin();

++ppit1;

p = \*ppit1;

}

else

{

ppit = pit;

--ppit;

p = \*ppit;

}

std::cout << "p-" << p << "pit-" << \*pit << "tempit-" << \*tempit <<"it-"<<\*it<< std::endl;

if(CGAL::right\_turn(p,\*pit,\*tempit))

{

if(!(CGAL::right\_turn(p,\*pit,\*it) && CGAL::left\_turn(\*tempit,\*pit,\*it)))

{

std::cout<<"s"<<std::endl;

flag = 1;

break;

}

}

else

{

if(CGAL::left\_turn(p,\*pit,\*it) && CGAL::right\_turn(\*tempit,\*pit,\*it))

{

std::cout<<"z"<<std::endl;

flag = 1;

break;

}

}

}

if((pointz != \*it) && (pointz != \*pit))

{

std::cout<<"k"<<std::endl;

flag = 1;

break;

}

}

}

if(flag == 0)

{

std::cout<<"h"<<std::endl;

git = --short\_left.end();

while(\*git != \*pit)

{

std::cout<<"w"<<std::endl;

short\_left.erase(git);

git = --short\_left.end();

}

std::cout<<"f"<<std::endl;

std::cout<<\*bit<<\*it;

short\_left.push\_back(\*bit);

std::cout << "x" << std::endl;

}

else

{

break;

}

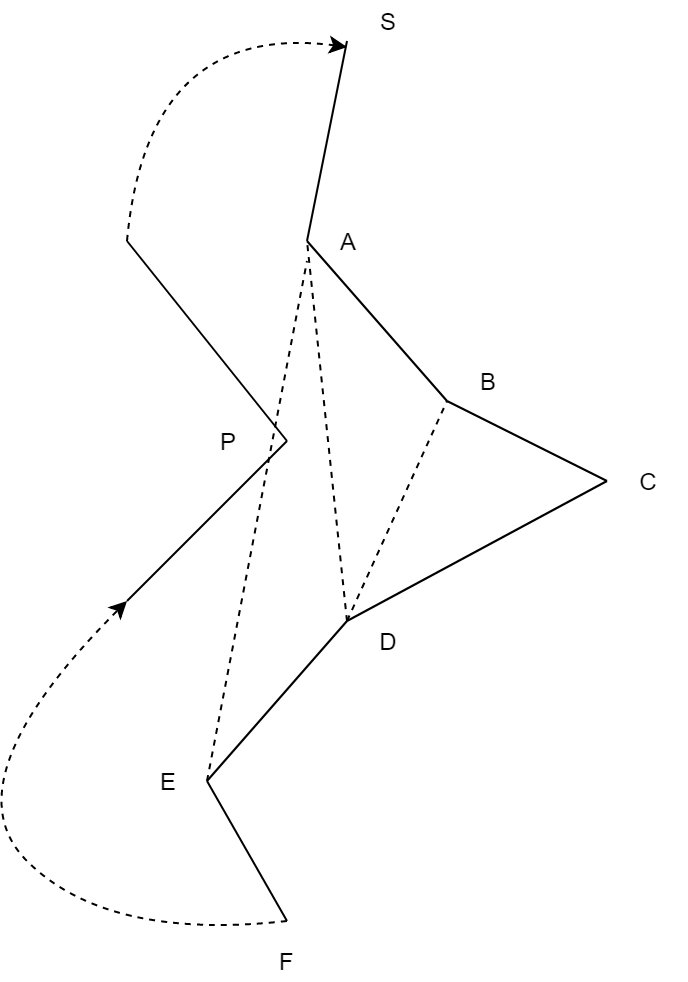
}

}

++bit;

} while(bit != list\_left.end());

Which points to remove from the list?



Suppose this is the polygon. Consider the figure above here, our starting point is S so we first push that into the stack then we push the point A, B into the stack. The stack now contains three points S,A,B. So with three elements we can check if a line SB can be drawn.

In this case it is not possible as the line SB would not lie inside the polygon so we can not remove the point A from the stack. We proceed on to the next point in the list (left or right chain). Here it is the point C. No reduction is possible in this case also. So we proceed on to the next point that is D. As line DB lie on the interior of the polygon and the It does not intersect with any other boundary of the polygon. We can have a reduction here we remove the element C from the stack and include D. So the list now becomes like {S,A,B,D}.we can perform the same kind of reduction again for the next point and that leads us to list {S,A,D}.We cannot do the same thing for E as the boundary of the polygon intersects the line AE. This we include that into the stack. This is done of all the points until we reach the terminating point. This function thus reduce the problem instance into a smaller one. Sometimes the polygon itself may be in reduced form then this function would return the input list itself as the output.

This same approach can be done for the right chain also with a little bit modification to the condition checks.

Now we are left with two reduced list. These two list have some property that makes the further computations simpler.

**Complexity:**

The function for computing the left and the right chain can be done in O(n) time as it just going through all the points in the polygon exactly once. The function for computing the reduced chains requires O(n2) time as for every removal of a point from the left or right chain, we need to check if the new line intersects with the boundary of the polygon.

**Conclusion:**

The shortest path on a polygon problem was implemented in CGAL QT. The project gave me insights on how the computational geometry problems are solved and how they were implemented in CGAL. The use of CGAL libraries were very useful in solving sub problems. Over the course of this project, I was able to gain knowledge on CGAL and QT.

Acknowledgments

I would like to thank myself

SHAHIN JOHN JS,

Winter Semester 2018

National Institute of Technology Calicut

REFERENCE

* Geometry Lab. Calculating the shortest path in a simple polygon - the Funnel Algorithm. http://web.informatik.uni-bonn.de/I/GeomLab/ShortestPathLP/ [2]
* Stanford CS268. 1992. Shortest Path Problems. Jim Stewart. https://graphics.stanford.edu/courses/cs268-09-winter/notes/handout7.pdf [3]
* University of Waterloo CS 860. Fall 2014. Basic geometric shortest path algorithms – shortest paths in 2D. Anna Lubiw. <https://cs.uwaterloo.ca/~alubiw/CS860/Lecture2-class.pdf>
* JOURNAL OF COMPUTER AND SYSTEM SClENCES 39, 126-152 (1989)