

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2006

On some geometric optimization problems.

Samidh Chatterjee
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Chatterjee, Samidh, "On some geometric optimization problems." (2006). *Electronic Theses and Dissertations*. 2922.
<https://scholar.uwindsor.ca/etd/2922>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

On Some Geometric Optimization Problems

by

Samidh Chatterjee

A Thesis

Submitted to the Faculty of Graduate Studies and Research
through Computer Science

in Partial Fulfillment of the Requirements for the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2006



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-17076-2

Our file Notre référence

ISBN: 978-0-494-17076-2

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

© Samidh Chatterjee, 2006

All Rights Reserved

Abstract

An optimization problem is a computational problem in which the objective is to find the best of all possible solutions. A geometric optimization problem is an optimization problem induced by a collection of geometric objects. In this thesis we study two interesting geometric optimization problems. One is the all-farthest-segments problem in which given n points in the plane, we have to report for each point the segment determined by two other points that is farthest from it. The principal motive for studying this problem was to investigate if this problem could be solved with a worst-case time-complexity that is of lower order than $O(n^2)$, which is the time taken by the solution of Duffy et al. (13) for the all-closest version of the same problem. If h be the number of points on the convex hull of the point set, we show how to do this in $O(nh + n \log n)$ time. Our solution to this problem has also triggered off research into the hitherto unexplored problem of determining the farthest-segment Voronoi Diagram of a given set of n line segments in the plane, leading to an $O(n \log n)$ time solution for the all-farthest-segments problem (12). For the second problem, we have revisited the problem of computing an area-optimal convex polygon stabbing a set of parallel line segments studied earlier by Kumar et al. (30). The primary motive behind this was to inquire if the line of attack used for the parallel-segments version can be extended to the case where the line segments are of arbitrary orientation. We have provided a correctness proof of the algorithm, which was lacking in the above-cited version. Implementation of geometric algorithms are of great help in visualizing the algorithms, we have implemented both the algorithms and trial versions are available at www.davinci.newcs.uwindsor.ca/~asishm.

Dedication

DEDICATED TO MY PARENTS
Samiran Chatterjee and Tapati Chatterjee

Acknowledgements

The successful completion of my M.Sc. degree at the University of Windsor has been possible due to the sustained cooperation of a number of individuals. I take this opportunity to thank them all who have been kind enough to subject themselves to the various drafts through which this work evolved.

The intellectual debts for this enterprise are indeed numerous. Let me begin by acknowledging the inspiring influence of my advisor, Dr Asish Mukhopadhyay. Under his supervision, learning and research went hand in hand for me. With his enthusiasm, he ceaselessly reminded me to pose new questions, think variously, reevaluate my arguments constantly—a way of thinking that will extend far beyond this thesis. Without his support, drive and explicit push, this work would not have been what it is now.

The communities of scholarship of which I was fortunate enough to become a part of contributed to my intellectual growth, and helped me mature as a researcher. In this regard, I would like to thank Dr Arunita Jaekel for her time and energy in helping me improve this document with her suggestions on certain aspects of my thesis. I record my appreciation for Dr Chitra Rangan for her insightful comments, ideas, criticisms and encouragement. I am indebted to Dr Akshai Aggarwal for his co-operation and camaraderie whenever I required. The companionship and constant support of my friend Augustus have gone into many pages of this thesis. My thanks to all my friends at Windsor for providing a stimulating academic ambience and sharing with me my blues. I would like to thank my department secretaries, Ms. Margaret Mayer-McKnight and Ms. Roxana Moreira as well as the library personnel for helping

me in several possible ways.

My deepest gratitude is due to my parents. They have been a constant source of motivation right from my childhood. During my M.Sc. studies here, there was no exception. Without their inspiration and sacrifice I would not have been where I am now. Last but not the least, I am deeply indebted to the love of my life, Mimi, who has always been an emotional anchor and kept me going through trying times.

Contents

Abstract	iv
Dedication	v
Acknowledgements	vi
Contents	viii
List of Figures	x
1 Introduction to Computational Geometry and Geometric Optimization	1
1.1 Introduction	1
1.2 Computational Geometry	2
1.2.1 Limitations of Computational Geometry	3
1.3 Geometric Optimization	4
1.3.1 Optimization Techniques	4
1.4 Some Application Areas of Geometric Optimization	7
1.4.1 Layered Manufacturing	7
1.4.2 Geographical Information Systems	7
1.4.3 Geometric Modeling and Industrial Geometry	8
1.5 Organization of the thesis	9
2 On the All Farthest Segments Problem for a Planar Set of Points	11

2.1	Some Computational Geometry Concepts	11
2.1.1	Convex Hull	11
2.2	Voronoi Diagrams	12
2.2.1	Some Properties of Voronoi Diagrams	13
2.2.2	Farthest Point Voronoi Diagram	14
2.3	Previous Work	14
2.4	Characterization of a farthest segment	16
2.5	Algorithm	20
2.6	Analysis	21
2.7	Conclusion	21
3	On Intersecting a Set of Vertical Line Segments with a Convex Poly- gon of Minimum Area	22
3.1	Previous Work	22
3.2	Definitions and Notations	23
3.3	Characterization	23
3.4	Solving the Optimization Problem	27
3.5	The Algorithm	30
3.6	Analysis of the Algorithm	31
3.7	Conclusion	31
4	Experimental Results	32
4.1	Results on the first problem	32
4.2	Results on the second problem	39
5	Conclusions and Future Research	44
	Bibliography	46
	Vita Auctoris	50

List of Figures

1.1	<i>Minimum Enclosing Circle for a set of points</i>	3
1.2	<i>2-dimensional linear programming (27)</i>	5
1.3	<i>Optimal orientation found by Majhi et al's weighted stair stepping algorithm for a speedometer component (21)</i>	8
1.4	<i>Area labelling:labelling countries with curved labels</i>	8
1.5	<i>Input Object (19)</i>	9
1.6	<i>Reconstructed Object (19)</i>	9
2.1	<i>Convex Hull of a set of points (27)</i>	12
2.2	<i>Voronoi Diagram</i>	12
2.3	<i>Voronoi Diagram with its Delaunay triangulation</i>	13
2.4	<i>Farthest Point Voronoi Diagram (17)</i>	14
2.5	<i>Farthest distance from p_i to segment (p_j, p_k) is to an intermediate point (Type A segment)</i>	16
2.6	<i>Farthest distance from p_i to segment (p_j, p_k) is to an endpoint (Type B segment)</i>	17
2.7	<i>If a type A segment is not a convex hull edge</i>	17
2.8	<i>Illustrating the case when p_j and p_k are both internal vertices of the convex hull of S)</i>	18
2.9	<i>p_j and p_k are non-adjacent convex hull vertices</i>	19
3.1	<i>A set of vertical segments with a common transversal</i>	24
3.2	<i>A truncated set of vertical line segments</i>	25

3.3	<i>Convex Polygon that must be included by any polygon that intersects S</i>	27
3.4	<i>Intervals are invisible</i>	28
3.5	<i>Two independent optimization problems</i>	28
3.6	<i>Tangents to $hdc(S)$ and $luc(S)$ from a point in an interval on lL . . .</i>	29
4.1	<i>Input Points</i>	33
4.2	<i>Point 1, Farthest segment is a convex hull edge</i>	34
4.3	<i>Point 2, Farthest segment is a convex hull edge</i>	35
4.4	<i>Point 3, Farthest segment has one endpoint as an internal point . . .</i>	36
4.5	<i>Point 4, Farthest segment is a convex hull edge</i>	37
4.6	<i>Point 5, Farthest segment is a convex hull edge</i>	38
4.7	<i>Input Line Segments</i>	40
4.8	<i>Upper Hull of the Lower End Points and Lower Hull of Upper Endpoints</i>	41
4.9	<i>Searching for the minimum polygon in each interval</i>	42
4.10	<i>The Minimum Polygon</i>	43

Chapter 1

Introduction to Computational Geometry and Geometric Optimization

1.1 Introduction

This thesis aims at addressing problems falling in the class of *proximity problems* in the field of geometric optimization. In this class of problems, we attempt in finding out the maximum or minimum value of the objective function, and that too in an efficient manner. Our work aims at minimizing the objective function, being induced by geometric objects such as points and lines, hence they are geometric optimization problems.

Design and analysis of algorithms is a popular field of study for a long period of time. Study of computational geometry has been motivated from this field in the late 1970s (10). A large number of efficient researchers are working in this field and the subject has now its own identity through its own conferences and journals. The various challenging problems studied in this field is a reason why this field has become so popular as a research discipline. Also it is not confined to the books only, it

has substantial real world applications in computer graphics, Geographic Information Systems (GIS), robotics, medical science to name a few.

In the next two sections we try to give a panoramic view about computational geometry and geometric optimization.

1.2 Computational Geometry

Computational geometry is the study of algorithms aimed at solving problems in the field of geometry. Study of purely geometrical problems is also considered to be part of computational geometry. There are two main branches of computational geometry.

Combinatorial computational geometry, also called algorithmic geometry, mainly deals with developing efficient algorithms and data structures for solving problems stated in terms of basic geometrical objects like points, line segments, polygons etc. A typical combinatorial geometric algorithm is the minimum spanning circle problem, that is given n points in a plane, the problem is to find the circle of minimum area that contains these points. The simplest algorithm considers every circle defined by two or three of the n points, and finds the smallest of these circles that contains every point (Fig. 1.1). There exists $O(n^3)$ such circles, and each takes $O(n)$ time to check, for a total running time of $O(n^4)$. Designing of various optimization techniques for reducing this time complexity were later found, and now this problem can be solved in linear time by a technique called *Prune and Search* due to Megiddo (25). We will briefly introduce some of these techniques in the next section when we discuss Geometric Optimization.

Another field of computational geometry is Numerical geometry. It deals with geometric modeling. This area is concerned with problems such as curve and surface reconstructions, modeling and representation and has applications in shipbuilding, aircraft, and automotive industries.

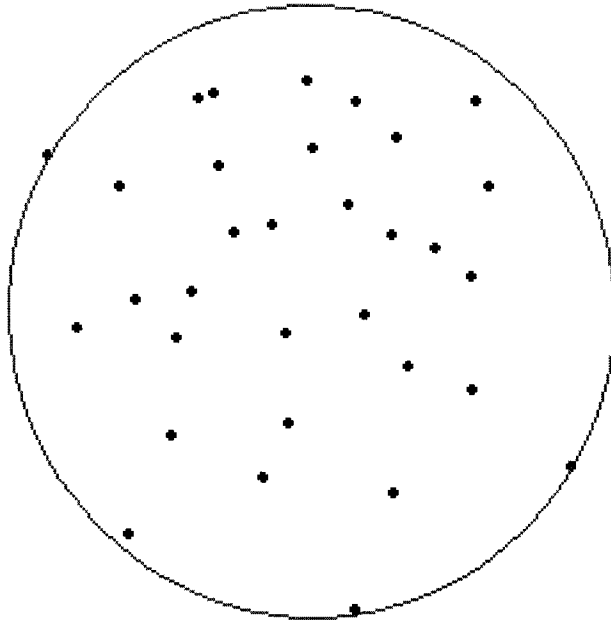


Figure 1.1: *Minimum Enclosing Circle for a set of points*

1.2.1 Limitations of Computational Geometry

One of the limitations of Computational Geometry is its discrete nature (27). In other words, when we solve a problem on a computer we express it in a discrete form. Some applications in real world deal with discrete approximation to continuous phenomenon. In Geographic Information Systems, road networks are discretized into collections of line segments.

Generally, researchers on computational geometry are originally experts in design and analysis of algorithms, but they do not deal much with core geometry (27). As a result, they intend to work on problems where they have to deal less with geometry and numerical computations. On one side it makes working in Computational Geometry fun by only allowing us to deal with combinatorial issues with really no requirement of substantial knowledge in analytical or differential geometry. But it also limits the scope of the applications of the field.

While most of the computational geometry problems deal with issues in 2-dimensions, our life may be made harder in higher dimensions. In 2 dimensions it

is easy to understand as we can visualize the problems without much difficulty. In higher dimensions, it is much more difficult to understand the problems and most of the applications require the study of problems in 3 or higher dimensions. Often the results obtained in 2-dimensions do not correspond to those obtained in three dimensions. For example, if we are given a set of n points in a plane and are asked to find the rectangle of minimum area that contains any k points among them, we will see that at least one edge of this rectangle will flush with one of the edges of the convex hull of these k points (9). But this result does not hold good in 3 dimensions.

1.3 Geometric Optimization

Geometric Optimization has emerged to be an important area in Computational Geometry. The study of this area has been motivated by applications in various fields like GIS, robot motion planning, optimal layout problems, etc.

The goal of an optimization problem is find the best of all possible solutions. A geometric optimization problem is one that is induced by a collection of geometric objects. The minimum enclosing circle problem mentioned in the last section is a typical example of a geometric optimization problem.

1.3.1 Optimization Techniques

There are quite a many optimization techniques available in literature. We briefly introduce some of these in the next section.

Parametric Search

Parametric Search is an optimization technique where we have a decision problem based on a real value, say λ , and the corresponding problem $P(\lambda)$ is monotonous on λ i.e. if $P(\lambda_0)$ is true, then λ is true for all $\lambda < \lambda_0$ (2) (32) (4). Our main purpose is to find λ^* , the maximum value for which $P(\lambda_0)$ is true. Several improvements of the parametric search technique have been suggested in (22) (23).

Linear Programming

In linear programming (LP) we have a set of linear inequalities, or *constraints*, which we can think of as defining a (may be empty or unbounded) polyhedron in space, called the *feasible region*, and we also have a linear *objective function*, which is to be minimized or maximized subject to the given constraints (27). A typical d -dimensional linear programming problem might be expressed as:

Maximize: $c^T x$

Subject to: $Ax \leq b$, where c and x are d -vectors, b is an n -vector and A is an $n \times d$ matrix.

From a geometric point of view, the feasible region is the intersection of half-spaces. Thus it is a (may be unbounded) convex polyhedron in d -space. The objective function can be viewed as a vector \vec{c} . So we search in the feasible region for a point that is farthest in the direction \vec{c} . We can call it the *optimal vertex*.

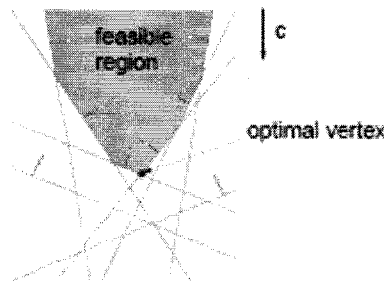


Figure 1.2: 2-dimensional linear programming (27)

Prune and Search

This technique is similar to parametric searching in the way that it too performs binary search implicitly over a given set of finite input values for λ^* (25) (24). But the novel of this method lies in the fact that while searching it tries to eliminate a constant fraction of the input objects, and this elimination does not affect the optimal

value λ^* . Thus after a logarithmic number of steps, the size of the problem becomes constant and then we can solve the problem by brute force method. The overall running time of the algorithm is proportional to the cost of a single pruning stage.

Randomized Algorithms

Input to a randomized algorithm is being guided by random numbers. Thus its run time complexity varies from one execution to another even with a fixed input. In the analysis of a randomized algorithm we establish bounds on the expected value of a the running time of the algorithm, (or any other performance measure) that are valid for every input, as against the worst case complexity for deterministic algorithms. It is an essential tool in computational geometry (3) (1). There are two major benefits of randomization: simplicity and speed. A randomized algorithm is the fastest algorithm available, or the simplest, or both for many applications.

The deterministic algorithmic methods make our life harder in generalizing to higher dimensions. This becomes much easy in the randomized frame. In two dimensions also, this leads to algorithms that are more efficient than the deterministic ones. Unfortunately, there are some deterministic algorithms that have no randomized counterparts.

Theoretically, we should have a truly random source (31). But in practice what we use is a pseudorandom generator, which we assume to be completely random. Quite naturally, these generators cannot guarantee the degree of randomness that may be required for good performance by these algorithms.

Now, an example of a pseudorandom generators is the linear congruential generator, or the LCG. The generators in the LCG class were known to exhibit strong and predictable regularities in most cases, until in 2000, Bernd Gartner (15) showed that it can produce misleading results in testing geometric algorithms that involve determinant computations.

Simulating a randomized algorithm in a deterministic fashion (31) is Derandomization. Derandomized algorithms clarifies the trade-off between randomness and determinism.

1.4 Some Application Areas of Geometric Optimization

We conclude our discussion on geometric optimization by mentioning some application areas.

1.4.1 Layered Manufacturing

Layered manufacturing is an important technology able to manufacture complex shapes. A multi-disciplinary project at Rutgers University (18) is aimed at developing advanced material delivery systems for layered manufacturing and rapid prototyping. We briefly describe their work below.

The CAD model of the 3D object is sliced using slice algorithms. The information on each slice is then sent to a manufacturing unit which consists of a material delivery or a curing system capable of tracing out the layer. Each layer has an associated thickness and the entire layer has the same cross-section. Once the current layer is ready, the computer sends the information about the next layer to the manufacturing system which builds it on the existing layers. In this way, the entire object is built layer-by-layer.

Majhi et al. (21) discuss some important issues arising in this area such as minimizing stair step errors on the surfaces of the manufactured surfaces under various formulations, minimizing the volume of so called support structures used, as well as minimizing the contact area between the supports and the manufactured object. They present efficient algorithms to address these issues by reducing these problems into geometric optimization ones such as halfplane range searching, constrained optimization, etc(Fig. 1.3).

1.4.2 Geographical Information Systems

A Geographical Information System is a collection of spatially referenced data (i.e. data that have locations attached to them) and the tools required to work with the data. One of the important issues in GIS is the map labeling problem (14). (Fig. 1.4) illustrates a general map labeling problem. This problem is concerned

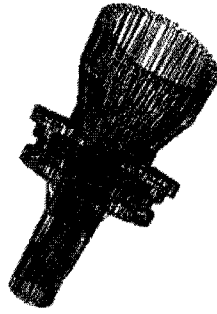


Figure 1.3: *Optimal orientation found by Majhi et al's weighted stair stepping algorithm for a speedometer component* (21)

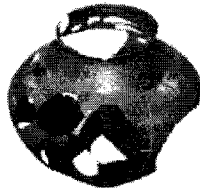
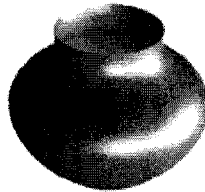


Figure 1.4: *Area labelling:labelling countries with curved labels*
(37)

with issues in labeling a set of sites (points, lines, regions); given a set of candidates (rectangles, circles, ellipses, irregularly shaped labels) for each site (37). A map can be a classical cartographical map, a diagram, a graph or any other figure that needs to be labeled. Map labeling is the problem of positioning labels on a map, maintaining some constraints such as the labels will not overlap and they should not cover important features of the map. Cartographers say that labeling a map manually consumes fifty percent of the time of actually drawing the map itself. So they seek some automated method to handle this problem. Finding such a labeling is NP-hard. Various geometric optimization techniques have come into play while addressing these issues.

1.4.3 Geometric Modeling and Industrial Geometry

Geometric Modeling and Industrial Geometry is a research unit at the Institute of Discrete Mathematics and Geometry, Vienna University of Technology (26). They are performing application oriented fundamental research and industrial research closely

Figure 1.5: *Input Object* (19)Figure 1.6: *Reconstructed Object* (19)

connected to geometry. One of the application areas they deal with is constrained optimization problems occurring in Geometric Computing. Ongoing research focuses on the computation of curves and surfaces constrained to surfaces and obstacle avoidance in motion design and curve and surface approximation.

Liu et al. (19) investigate 3D shape reconstruction from measurement data in the presence of constraints.

1.5 Organization of the thesis

In this thesis we will be investigating two interesting geometric optimization problems. The first problem is: Given n points in a plane, we report the farthest segment (from among the implicitly defined line segments defined by these n points) from each point. As we know, the number of line segments defined implicitly by these n points is in $O(n^2)$. A brute force approach to the solution of this problem will give rise to an $O(n^3)$ algorithm. Our goal was to design an efficient algorithm, thus trying to improve the time complexity of the brute force algorithm. We have been able to come up with the following result: If h be the number of vertices on the convex hull then our algorithm reports the farthest segment from each point in $O(nh + n \log n)$ time.

For the second problem we revisit the algorithm of Kumar et al. (30) for computing an area-optimal convex polygon intersecting a set of parallel line segments for which we provide a correctness proof and also an implementation. We also establish that after an initial step of computing convex hulls that is in $O(n \log n)$, the complexity of the rest of the steps is in $O(n)$. Study of the geometry of collection of parallel line segments was originally done by Goodrich et al. (16). Their work found applications in computer graphics, such as vectorizing scanned images, computing visibility for graphical display, and finding shortest paths for motion planning.

In the next two chapters we describe these two problems. The next chapter shows results shown by the implementation of our algorithms. We finally conclude with scope of future research that can be done on this thesis.

Chapter 2

On the All Farthest Segments

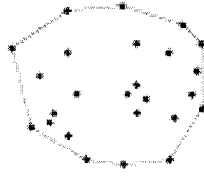
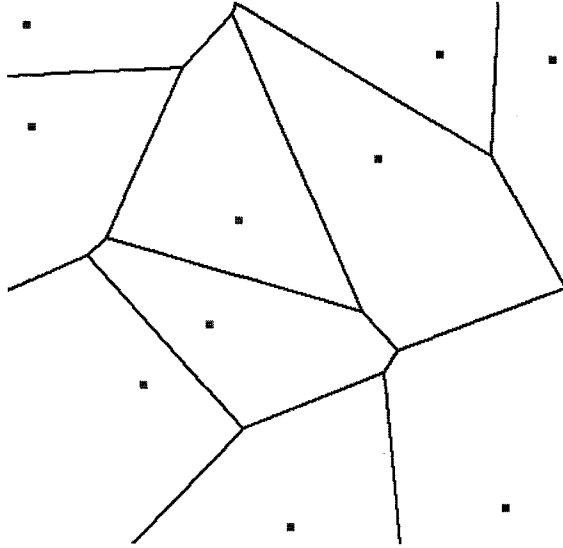
Problem for a Planar Set of Points

In this chapter, we outline a very simple algorithm for the following problem: Given a set S of n points $p_1, p_2, p_3, \dots, p_n$ in the plane, we have $O(n^2)$ segments implicitly defined on pairs of these n points. For each point p_i , find a segment from this set of implicitly defined segments that is farthest from p_i . We have $O(n^2)$ segments implicitly defined on pairs of these n points. The time complexity of our algorithm is in $O(nh + n \log n)$, where n is the number of input points, and h is the number of vertices on the convex hull of S .

2.1 Some Computational Geometry Concepts

2.1.1 Convex Hull

This is one of the very basic geometry structures. Given n points in a plane, we are interested to know the smallest convex polygon that contains these points. By the term convexity we mean that, if we join any two points in the convex hull, all the points in the line joining these two points will lie entirely within the convex hull. There are several algorithms for computing the convex hull of a set of points. We have used Graham Scan algorithm (27) here. Timothy Chan (6) proposed a very

Figure 2.1: *Convex Hull of a set of points (27)*Figure 2.2: *Voronoi Diagram*

efficient method of computing the convex hull.

2.2 Voronoi Diagrams

Given a set of points S in a plane Voronoi Diagram is a partition of a plane into regions, each region is the locus of points (x, y) , closer to a point of S than to any other point in S (33). Given two points, p_i and p_j , the set of points closer to p_i than to p_j is the half-plane containing p_i formed by the perpendicular bisector of $\overline{p_i p_j}$. If we denote this half-plane by $H(p_i, p_j)$, then the convex polygonal subdivision that describes the areas that are nearest to a set of the points in S is the intersection of such $N - 1$ halfplanes and has no more than $N - 1$ sides. Let us denote by $V(i) = \bigcap H(p_i, p_j), i \neq j$.

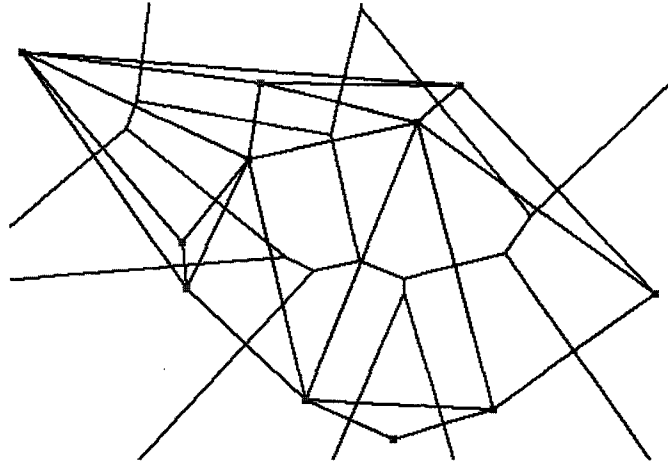


Figure 2.3: *Voronoi Diagram with its Delaunay triangulation*

This polygonal subdivision is called the *Voronoi diagram*(Fig. 2.2). The vertices of the diagram are called *Voronoi vertices* and the line segments are called *Voronoi edges*.

2.2.1 Some Properties of Voronoi Diagrams

We discuss some important properties of the Voronoi diagram (33) (27).

Voronoi Edges: For every vertex v of the Voronoi diagram, the circle centered at that point contains no other points of S .

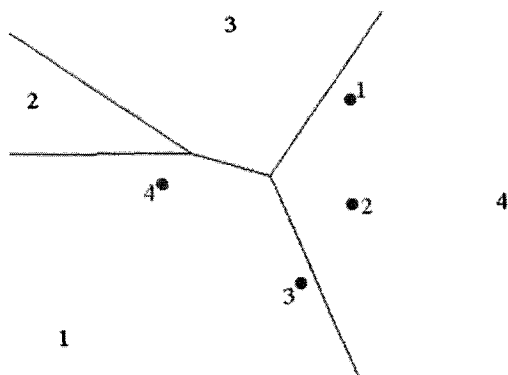
Voronoi vertices: Every vertex of the Voronoi diagram is the common intersection of exactly three edges of the diagram. Thus it is the center of the circle passing through these sites, and this circle contains no other sites in its interior.

Degree: Assuming no four points of S are cocircular, every Voronoi vertex has degree three.

Convex hull: Any cell $V(i)$ is unbounded iff p_i is a point on the boundary of the convex hull of the set S .

Triangulation: The straight-line dual of the Voronoi Diagram is a triangulation of S . This structure is called the Delaunay triangulation.

Size: The Voronoi diagram on N points has at most $2N - 5$ vertices and $3N - 6$ edges.

Figure 2.4: *Farthest Point Voronoi Diagram* (17)

There are various methods of computing Voronoi diagrams (33) (27). A detailed discussion of these methods is out of the scope of this thesis.

2.2.2 Farthest Point Voronoi Diagram

Farthest point Voronoi diagrams are the opposite of Voronoi diagrams (17). These identify the areas which have the greatest distance from the given points (Fig. 2.4). They have similar properties as the nearest version. A detailed algorithm for the construction of the farthest point Voronoi diagram can be found in (35).

2.3 Previous Work

The problem we study belongs to the class of proximity problems which has a long history in computational geometry. As for example, we can consider the *closest pair* problem. Given n points in a plane we are required to find the closest pair of points among them. An $O(n \log n)$ solution to this problem is by using Delaunay triangulation. In an arbitrary fixed dimension d , the first $O(n \log n)$ algorithm, based on divide-and-conquer, was described by Bentley and Shamos (5). They investigate a divide-and-conquer technique in multidimensional space which decomposes a geometric problem on n points in k dimensions into two problems on $n/2$ points in k dimensions plus a single problem on n points in $k - 1$ dimension. Special structure of the subproblems was also exploited to obtain an algorithm for finding the two closest

of n points in $O(n \log n)$ time in any dimension. Another $O(n \log n)$ algorithm of Vaidya (36) can actually find the nearest neighbor to each of the given points.

The more general problem of enumerating the k closest pairs (or enumerating the first k smallest distances) has also been looked into. Dickerson et al. (11) used the Delaunay triangulation to enumerate the k closest pairs. They present an $O(n \log n + k \log k)$ time and $O(n + k)$ space algorithm which takes as input a set of n points in the plane and enumerates the k smallest distances between pairs of points in nondecreasing order. They also present an $O(n \log n + kn \log k)$ solution to the problem of finding the k nearest neighbors for each of n points. Both of their algorithms are based on Delaunay triangulation. Timothy Chan in (7) revisits the problems of enumerating the k closest pairs and selecting the k -th smallest distance, given an n point-set. He presented randomized and deterministic algorithms with $O(n \log n + k)$ running time in any fixed-dimensional Euclidean space. For the selection problem, he describes an approach to obtaining k -sensitive time bounds. He also points out output-sensitive results for halfspace range counting that are of use in more general distance selection problems.

Our work is directly related to the following work done by Daescu and Luo(8). Given a set S of n points in a plane and another point q they give optimal $O(n \log n)$ time and $O(n)$ space algorithms for finding the closest and farthest line segments from q among those implicitly defined by points in S . They also suggest an $O(n \log n)$ time and $O(n)$ space algorithm to find the k -th closest line and also show a method to report these k closest lines in $O(n \log n + k)$ time and $O(n)$ space.

Duffy et al.(13) presented an algorithm that determines for every point $r \in S$ the closest distance between r and a line segment (p, q) implicitly defined in S . Their algorithm takes $O(n^2)$ time and $O(n)$ space. They also show that their algorithm is 3SUM-hard, and so it is unlikely that a better solution can be obtained.

While it is hard to provide any practical motivation for problems of this type that does not appear contrived, it is intriguing to know whether the *all-farthest* problem can be solved as efficiently as or faster than the *all-nearest* version.

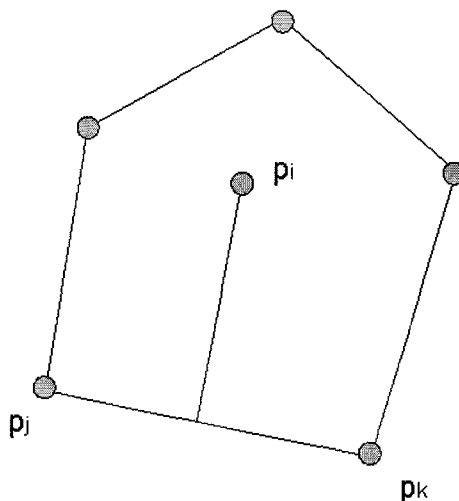


Figure 2.5: Farthest distance from p_i to segment (p_j, p_k) is to an intermediate point (Type A segment)

2.4 Characterization of a farthest segment

Let $\overline{p_j p_k}$ be a farthest segment of a point p_i . The farthest distance is obtained either by dropping a perpendicular from p_i to the segment $\overline{p_j p_k}$ (Fig. 2.5) or by joining p_i to the nearer one of the end points p_j and p_k (Fig. 2.6). We call these two types of farthest segments type *A* and type *B* respectively.

We design an algorithm by characterizing the two types of segments. To ensure the correctness of the arguments below, we shall assume that no three points of S are collinear.

Lemma 1. *If the segment $\overline{p_j p_k}$ is a type A farthest segment for a point p_i then $\overline{p_j p_k}$ is an edge on the convex hull of S .*

Proof: If the segment $\overline{p_j p_k}$ is not a convex hull edge, then there exists a point p_l of S in the open half-plane defined by the supporting line through p_j and p_k that does not contain p_i (Fig. 2.7). This gives a segment $\overline{p_j p_l}$ that is farther from p_i than $\overline{p_j p_k}$ since $\overline{p_i p_j}$ is the hypotenuse of the right-triangle formed by p_i , p_j and the foot of the perpendicular from p_i to $\overline{p_j p_k}$. This contradicts the assumption that $\overline{p_j p_k}$ is a farthest segment of p_i .

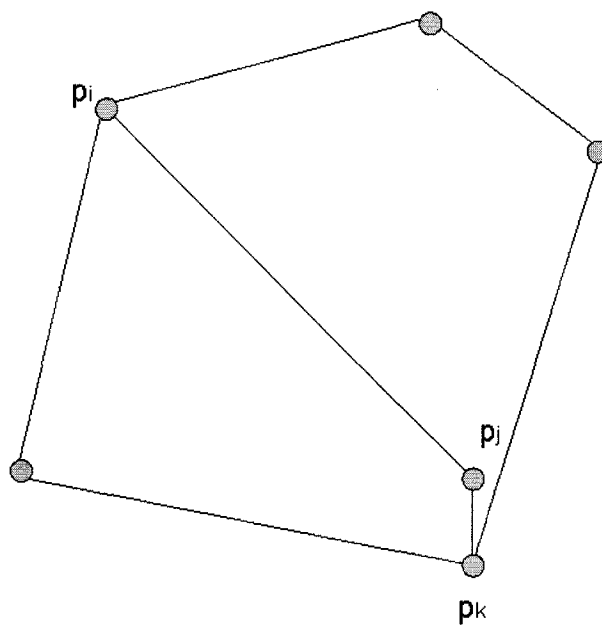


Figure 2.6: Farthest distance from p_i to segment (p_j, p_k) is to an endpoint (Type B segment)

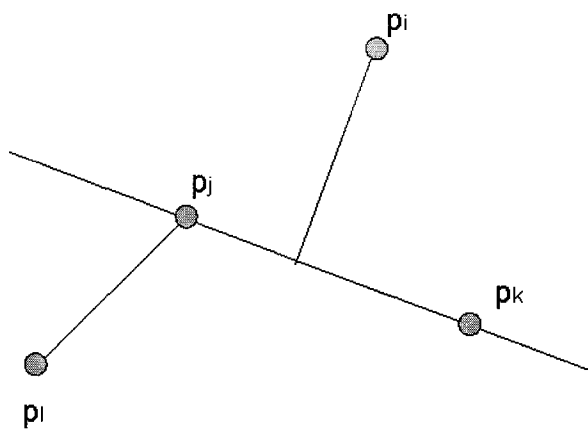


Figure 2.7: If a type A segment is not a convex hull edge

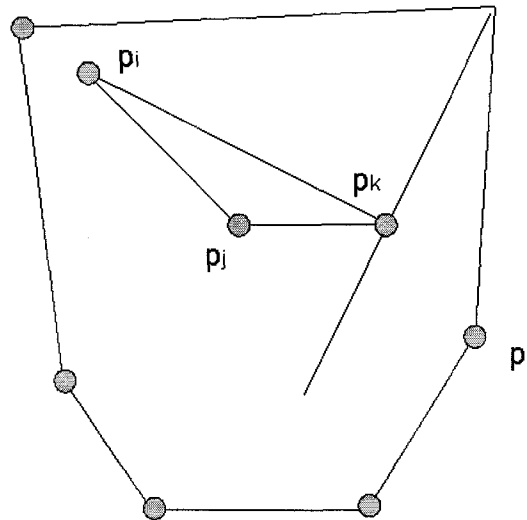


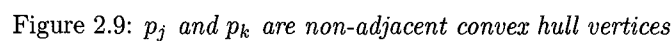
Figure 2.8: Illustrating the case when p_j and p_k are both internal vertices of the convex hull of S)

Lemma 2. *If the segment $\overline{p_j p_k}$ is a type B farthest segment for a point p_i , the farthest distance being the length of $\overline{p_i p_j}$, then either $\overline{p_j p_k}$ is an edge on the convex hull of S or p_j is farthest from p_i among all the points that are interior to the convex hull of the point set, while p_k is a convex hull vertex of the given point set.*

Proof: Let the farthest distance be realized by joining p_i to p_j . Our proof is in three parts, covering the mutually exclusive and exhaustive possibilities that the end points of $\overline{p_j p_k}$ are both points internal to the convex hull of S , are both convex hull vertices or one is an internal vertex while the other is a convex hull vertex. (1) Suppose p_j and p_k are both internal to the convex hull. If this were true, consider the half-plane defined by a line through p_k orthogonal to $\overline{p_i p_k}$ that does not contain p_i . This half plane must contain a vertex p_l of the convex hull of S , giving us a segment $\overline{p_k p_l}$ that is farther from p_i than $\overline{p_j p_k}$ and a contradiction. Hence this possibility is excluded (Fig 2.8).

(2) Suppose p_j and p_k are both vertices of the convex hull. We claim that in this case $\overline{p_j p_k}$ is a convex hull edge (Fig 2.9).

If otherwise, the segment $\overline{p_j p_k}$ divides the convex hull of S into two parts.



(3) p_k is a convex hull vertex and p_j is an internal vertex. We claim that in this case p_j is farthest from p_i among all internal vertices. Otherwise, let p_l be an internal point that is farther from p_i than p_j . There exists a point p_m that lies on the convex hull and is in the half-plane defined by a line through p_l orthogonal to $\overline{p_i p_l}$, not containing p_i . This gives us a segment $\overline{p_i p_m}$ farther from p_i than $\overline{p_j p_k}$ and a contradiction.

With these two lemmas, it is easy to design an efficient algorithm for solving this problem.

2.5 Algorithm

We first construct the convex hull of the point set; then the farthest-point Voronoi diagram of the interior points, if any. The time complexity of each of these two steps is in $O(n \log n)$, (33), (35). For each point p_i , we find the farthest segment as outlined in the following algorithm.

Algorithm All-Farthest-Segments

Input: A set of n points p_1, p_2, \dots, p_n

Output: The farthest segment $\overline{p_j p_k}$ for each p_i

for each p_i **do**

Step 1: Find the farthest segment among the edges of the precomputed convex hull; record the segment and the distance.

Step 2: Locate p_i in the precomputed farthest point Voronoi diagram of the points interior to the convex hull. Let p_j be its farthest neighbor and record the distance to it from p_i . If this distance is smaller than that computed in Step 1 report the segment found in Step 1 and quit, else continue.

Step 3: Draw a line orthogonal to the segment $\overline{p_i p_j}$; the other endpoint p_k is a convex hull vertex that lies in the halfplane not containing p_i . We find this by a linear search (we can afford this!) on the convex hull boundary. Report $\overline{p_j p_k}$ as the farthest segment.

od

2.6 Analysis

The time complexity of Step 1 is in $O(h)$; that of Step 2 is in $O(\log(n - h))$; while that of Step 3 is also in $O(h)$. Putting it all together, the time complexity of All-Farthest-Segments is in $O(nh + n \log n)$.

2.7 Conclusion

We have implemented the algorithm using JDK 1.4. One can view the software in (28). Clicking the mouse randomly on the screen to generates the points and then by clicking the button "Show Farthest Segments" one can see the furthest segments from each point. The software also shows how the segment is obtained, i.e. whether it is a convex hull edge or one of the endpoints is an internal point.

An improvement of this algorithm has been already done by R.L. Scot Drysdale and Asish Mukhopadhyay (12). The authors have used a farthest segment voronoi diagram for convex hull edges and have been able to solve the problem in $O(n \log n)$ time.

Chapter 3

On Intersecting a Set of Vertical Line Segments with a Convex Polygon of Minimum Area

We have revisited the problem of computing an area-optimal convex polygon stabbing a set of parallel line segments studied earlier by Kumar et al (30), and provided a correctness proof. We also establish that after an initial step of computing convex hulls that is in $O(n \log n)$, the complexity of the rest of the steps is in $O(n)$.

3.1 Previous Work

In (16) Goodrich and Snoeyink investigate the geometry of collections of parallel line segments. They look into the issue when a straight line or convex polygon can be fitted to such a collection. They define the *convex stabbing* problem in the following way: Let S be a collection of parallel line segments on a plane. A straight line is said to stab S if it intersects each line segment in S . They generalize this concept to convex polygons, where they redefine the term *stabbing* saying that a convex polygon stabs S if its boundary intersects each line segment in S . Thus they pose the convex stabbing problem: given a set S of parallel line segments in the plane, find a convex

polygon P that stabs S , if such a polygon exists, report failure otherwise.

The authors devise an algorithm that solves the convex stabbing problem for n parallel line segments in $O(n \log n)$ time and $O(n)$ space. They claim their solution to be optimal, as by reduction from sorting, any algorithm that outputs the stabbing polygon in clockwise order must take $O(n \log n)$ steps to find a stabber of n points on a circle. Also their work leads to finding a minimal perimeter or minimal area stabbing polygons in $O(n^2)$ time and $O(n)$ space.

Lyons et al. (20) presented an $O(n \log n)$ algorithm to compute a minimum-perimeter convex polygon that intersects a set of n isothetic line segments by reducing the problem to a shortest-path computation. David Rappaport (34) showed that a minimum perimeter polygon stabbing a set of line segments constrained to lie in a fixed number of orientations can be found in $O(n \log n)$ time. He also showed that if m denotes the number of orientations, then the complexity of the algorithm is given by $O(3^m n + \log n)$.

3.2 Definitions and Notations

Let S denote a set of n parallel line segments. Each line segment in S with endpoints p and q is denoted by \overline{pq} . The functions $\text{top}(\cdot)$ and $\text{bot}(\cdot)$ return the upper and lower end-points of a line segment.

3.3 Characterization

We first observe a trivial case. If all the line segments have a common transversal then the minimum area optimum polygon reduces to an arbitrary line segment. This is illustrated in (Fig. 3.1).

The question is, when can this case arise? Well, let us observe the y -values of the top and bottom end points. Let us denote the maximum value of the bottom end points by maxBot and the minimum value of the top end points by minTop . We observe that, this is indeed the case when the maxBot is of a smaller value than the

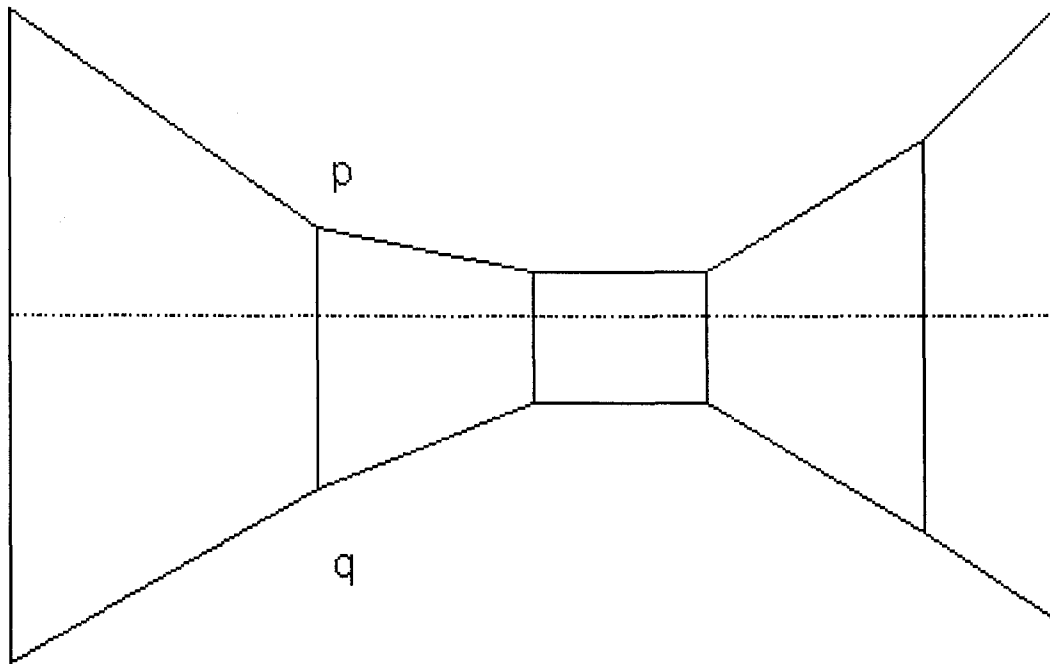


Figure 3.1: A set of vertical segments with a common transversal

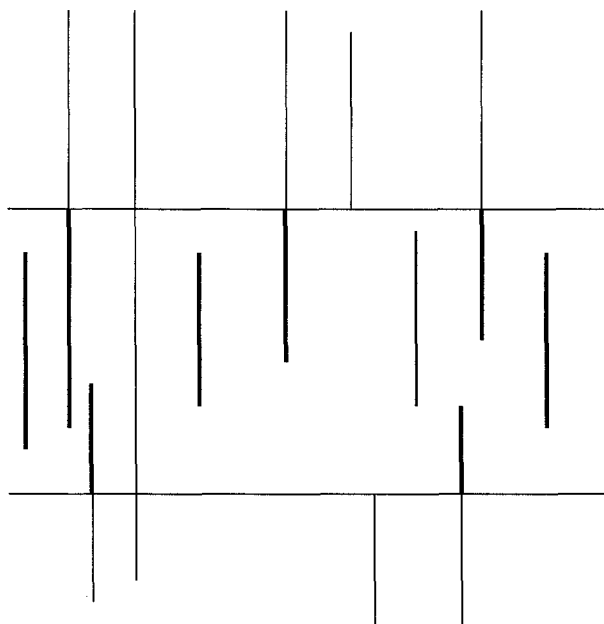


Figure 3.2: A truncated set of vertical line segments

minTop. We now deal with the situation when the case is otherwise.

We assume, without loss of generality that there is a unique leftmost line-segment \overline{lL} and a unique rightmost line segment \overline{rR} .

Lemma 3. *If the y-value of the maxBot is larger than that of minTop then the minimum area convex polygon will lie within a strip defined by horizontal lines through maxBot and minTop.*

Proof: The minimum area convex polygon will always have its vertices among the top and the bottom end points of the segments that lie between the leftmost and rightmost segments. Also, it will have a vertex on each of the extreme segments. Thus, the minimum convex polygon eventually lies within the horizontal strip defined by *maxBot* and *minTop*. Thus we redefine S to be consisting of the truncated line segments as defined in (Fig. 3.2).

Our main task is to determine the latter vertices. Before we do this, we try to characterize the minimum polygon. Let us define two functions, $\text{bot}(\cdot)$ and $\text{top}(\cdot)$ which return the bottom endpoint and the top endpoint of each line segment in S .

We construct the upper chain of the convex hull of the lower end-points of the line-segments in S . Going by the property of the convex hull, the $\text{bot}(s)$ of each line-segment s lies on or below this upper chain. Let us define a partial order relationship over the convex chains over a given range of x -values by defining a chain to be "less than or equal" to another if at every point of the range the corresponding y -value of the former is less than or equal to the corresponding y -value of the latter. Thus the upper hull of the lower end-points is the "smallest" one in the above partial order to have the above property. To denote this we denote this *lowest upward-convex chain* by $\text{luc}(S)$.

Similarly, the lower chain of the convex hull of the upper end-points is the "largest" among all convex chains which have $\text{top}(s)$ for each line segment s lying on or above it. We denote this *highest downward-convex chain* by $\text{hdc}(S)$.

Lemma 4. *If P be a convex polygon, lying between \overline{lL} and \overline{rR} the upper hull of P lies "on or above" $\text{luc}(S)$ and its lower hull lies "on or below" $\text{hdc}(S)$.*

Proof: Since P intersects each line segment s its bottom point cannot lie strictly above the upper chain of the convex hull of P . Thus the convex set consisting of the upper chain of the convex hull of P and the semi-infinite lines from the leftmost and rightmost vertices of P on \overline{lL} and \overline{rR} respectively to $-\infty$ contains the convex hull of the bottom end-points of all the s_i 's and thus, in particular, $\text{luc}(S)$. Similarly, the convex set consisting of the lower chain of the convex hull of P and the semi-infinite lines from the leftmost and rightmost vertices of P on \overline{lL} and \overline{rR} respectively to ∞ contain the convex hull of the top end-points of all the s_i 's and thus, in particular, $\text{hdc}(S)$. Hence the claim of the lemma is proved.

Thus any convex polygon P which intersects all the segments must include the area bounded by the polygon with thick edges as shown in (Fig. 3.3)

If P_{\min} is the minimum polygon, then the above holds true. We now prove the following lemma to further characterize P_{\min} . Let v_l and v_r be the leftmost vertex and rightmost vertex on \overline{lL} and \overline{rR} respectively.

Lemma 5. *P_{\min} is obtained by drawing tangents from v_l and v_r to $\text{hdc}(S)$ and $\text{luc}(S)$.*

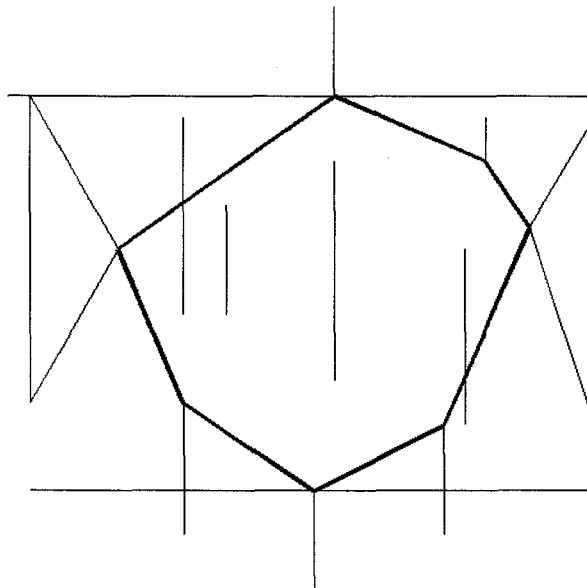


Figure 3.3: *Convex Polygon that must be included by any polygon that intersects S*

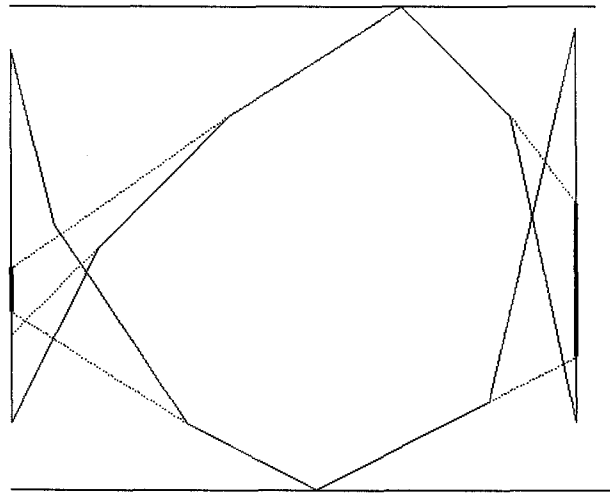
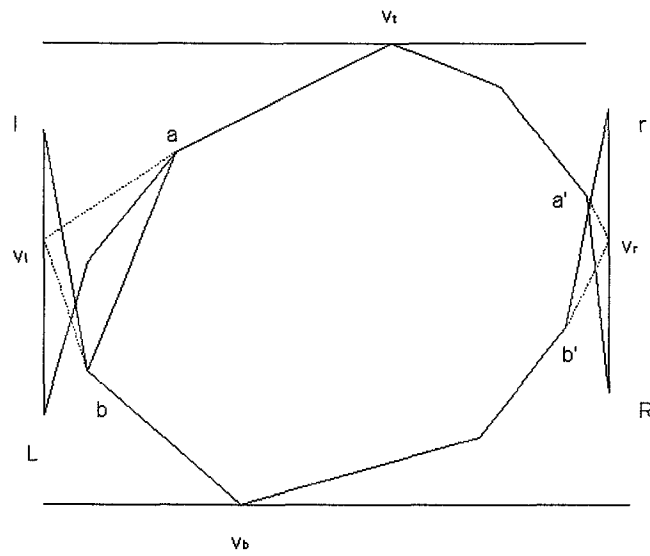
Proof: Let us consider the convex chains of P_{min} from v_l to v_b and v_t (Fig. 3.5). If these are disjoint from $hdc(S)$ and $luc(S)$ then we can obtain a convex polygon of smaller area than P_{min} , contradicting its minimality.

3.4 Solving the Optimization Problem

Now, as stated earlier the main problem here is to determine v_l and v_r . Before we do this, we prove the following lemma:

Lemma 6. *The determination of v_l and v_r can proceed independently. Each can be determined by local optimization problems.*

Proof: The edges that make up $hdc(S)$ and $luc(S)$ are extended to partition the leftmost segment \overline{lL} into intervals. Similarly the rightmost segment \overline{rR} is also partitioned. Now consider (Fig. 3.4). We see that the interval shown by the thick lines on the \overline{lL} is not visible by those on \overline{rR} . This gives rise to two independent optimization problems, each to be solved independently for the left and right part. Consider (Fig. 3.5). On the left we have to determine tangents from v_l to the convex chains

Figure 3.4: *Intervals are invisible*Figure 3.5: *Two independent optimization problems*

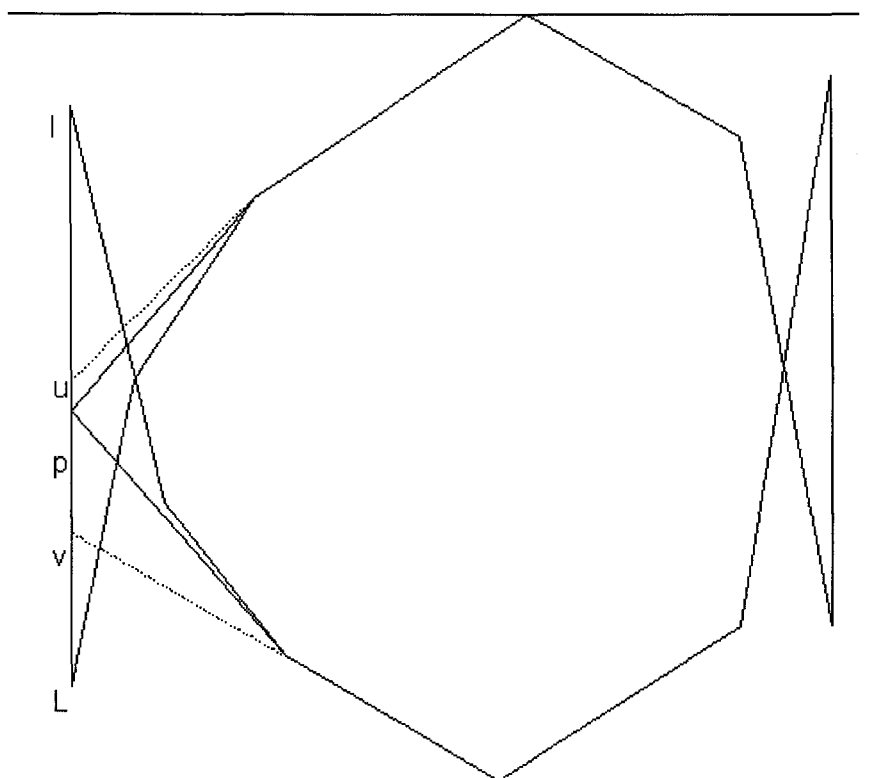


Figure 3.6: *Tangents to $hdc(S)$ and $luc(S)$ from a point in an interval on lL*

from L to v_t and from l to v_b so that the area of the $\triangle v_l ab$ is minimum. Similarly on the right we have to determine tangents from v_r to the convex chains from R to v_t and from r to v_b so that the area of the $\triangle v_r a' b'$ is a minimum.

We now discuss how to solve the optimization problem. We will explain how to solve the problem on the left, an exact similar kind will be done on the right. From each point in an interval on the two extreme line segments we can draw tangents to a vertex of $hdc(S)$ and to a vertex of $luc(S)$ as shown in (Fig. 3.6), where from the point p in the interval $[u, v]$ on \overline{LL} , tangents have been drawn to the convex chains $hdc(S)$ and $luc(S)$. The optimization for each interval goes as follows: the chosen point for which the area is a minimum will have to be an endpoint of the interval, determined by the skew of the segment joining the points of tangency with respect to \overline{LL} . We determine the area of the convex polygon to the left of $v_t v_l$ as a result of this optimization; the minimum area of all polygons obtained from each interval is $P_{minleft}$. Similarly, we determine the minimum convex polygon, $P_{minright}$ to the right of $v_t v_l$ and bounded by it. The area of P_{min} is the sum of two values. We formally describe the algorithm in the next section.

3.5 The Algorithm

Algorithm VerticalMinPolyStabber

Step 1. Compute the upper hull $luc(S)$ of the points $bot(s)$ and the lower hull $hdc(S)$ of the points $top(s)$.

Step 2. Extend the edges of these chains to partition $\overline{LL}(\overline{rR})$ into subintervals.

Step 3. For each subinterval on $\overline{LL}(\overline{rR})$ find the minimum triangle and compute the area of the left(right) polygon; update the current minimum on the left(right).

Step 4. Report the minimum area by summing the leftMinimum and the rightMinimum.

3.6 Analysis of the Algorithm

The complexity of Step 1 is bounded above by $O(n \log n)$. The time complexity of Step 2 is in $O(n)$. That of Step 3 is in $O(n)$ as the computation of the area of the convex polygon, say on the left, corresponding to the uppermost point on \overline{ll} , takes $O(n)$ time. The re-computation as we move to the vertex below this takes $O(1)$ time, and again for the vertex below that and so on, for a total time that is $O(n)$. Hence the VerticalMinPolyStabber is in $O(n \log n)$.

3.7 Conclusion

An implementation of the above algorithm has been done in Java using JDK 1.4. One can use the software going to (28).

As a future work of this algorithm, Mukhopadhyay (29) has proposed an $O(n^5)$ algorithm for the version of the problem where he has considered a set of isothetic line segments.

Chapter 4

Experimental Results

Both the algorithms have been developed in Visual Studio 2005. Programming language used is Java using JDK 1.4. The softwares have been developed in Windows XP platform.

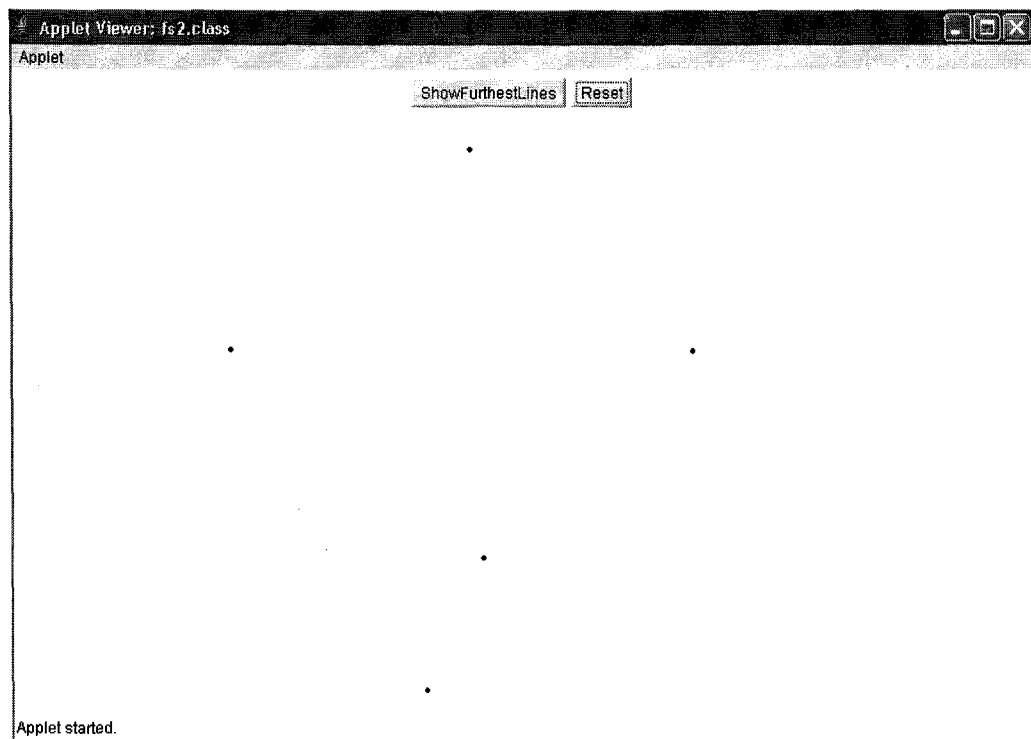
4.1 Results on the first problem

The following pages show some snapshots of the software developed for a set of points. How to use the software is described below.

Step 1: Open the Applet Window clicking on the *software* link on the homepage of Dr Asish Mukhopadhyay(<http://davinci.newcs.uwindsor.ca/~asishm>).

Step 2: The Applet Window appears. Click the mouse at different points on the applet window.

Step 3: When you think you have had enough points, and now want to see the farthest segment from each one of them, click the button "ShowFurthestLines". Go on clicking it, and you will be shown the farthest segment one by one. The concerned point whose farthest segment is being shown is marked with a red hollow box. The farthest segment is shown in red. Whether it is obtained by joining an endpoint, or by dropping a perpendicular, is being shown with a blue line.

Figure 4.1: *Input Points*

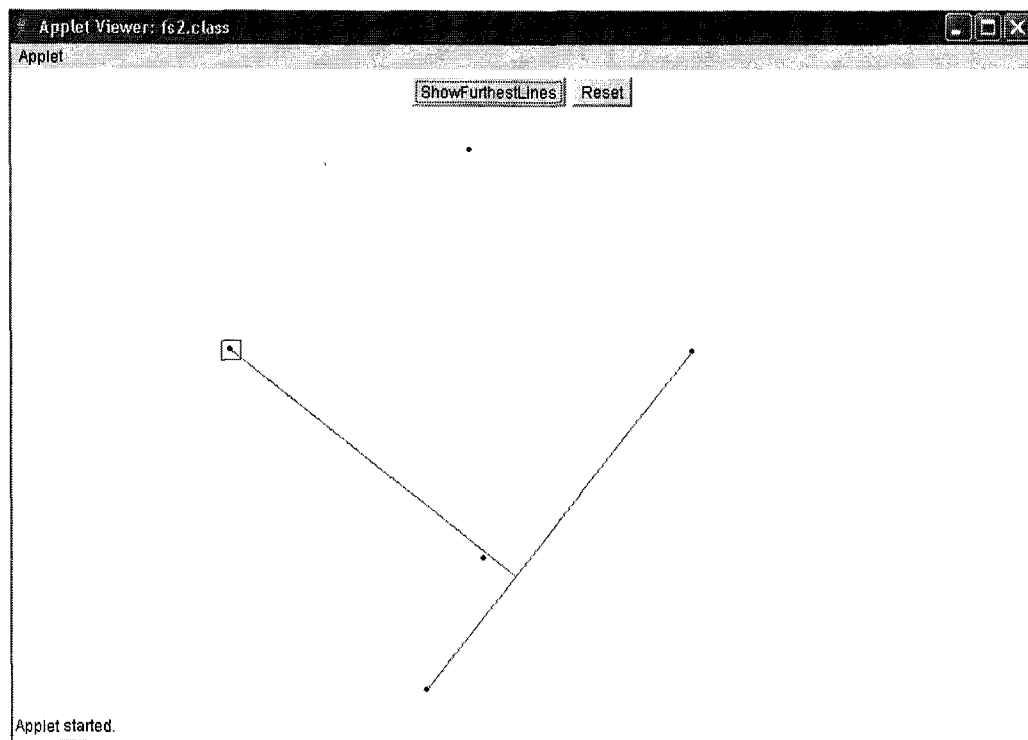


Figure 4.2: Point 1, Farthest segment is a convex hull edge

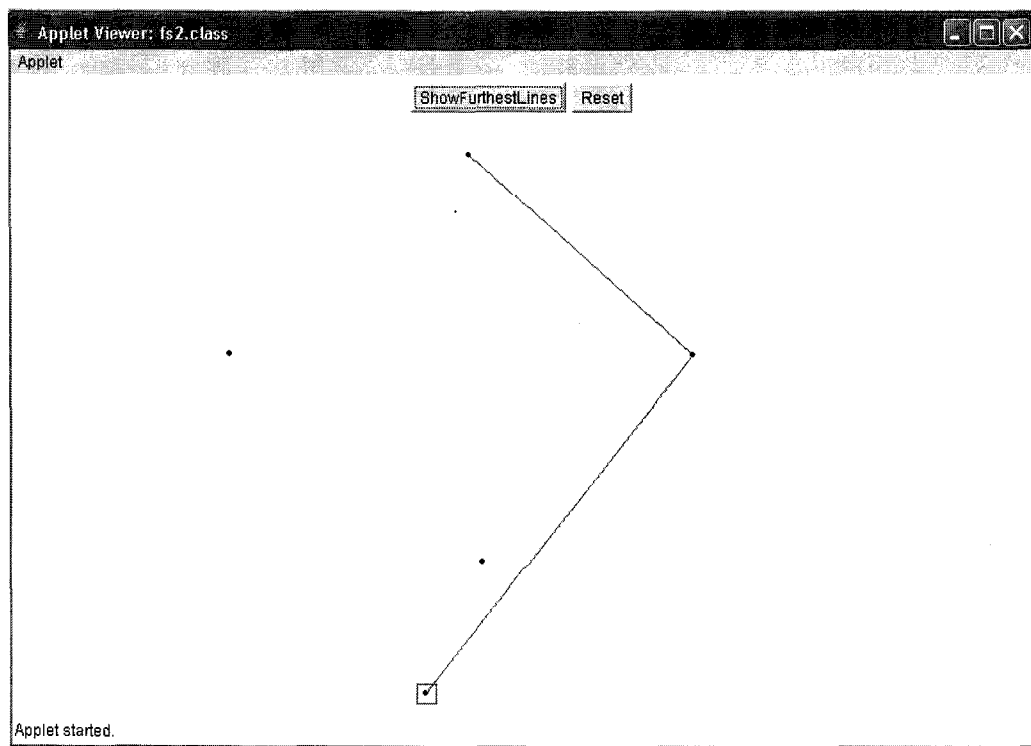


Figure 4.3: *Point 2, Farthest segment is a convex hull edge*

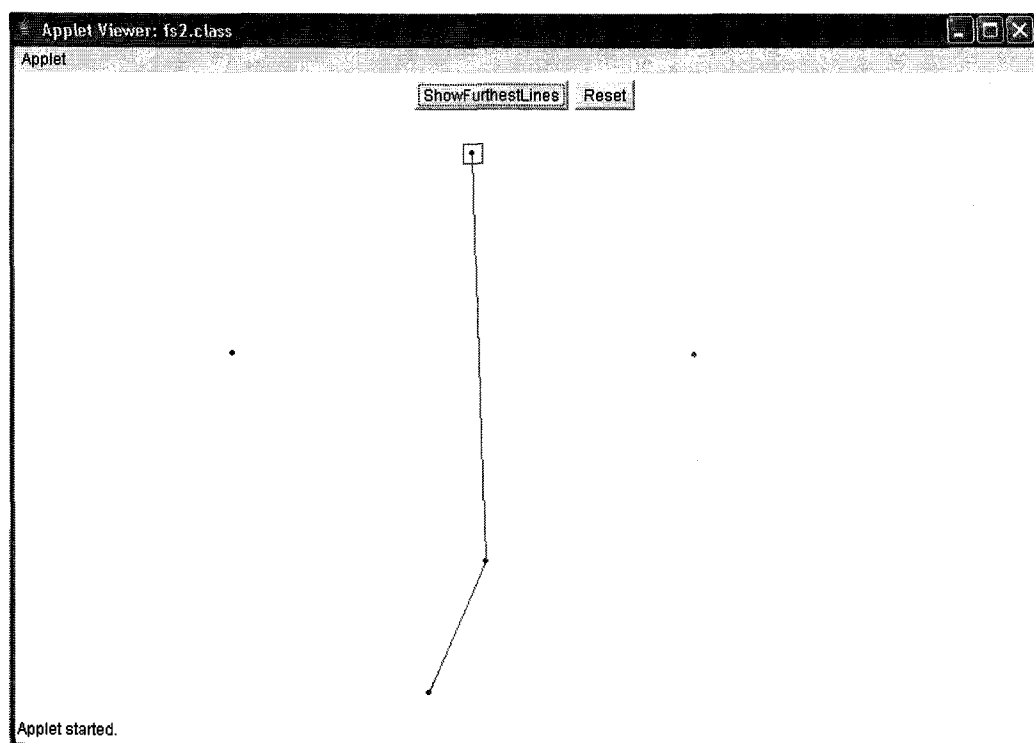


Figure 4.4: *Point 3, Farthest segment has one endpoint as an internal point*

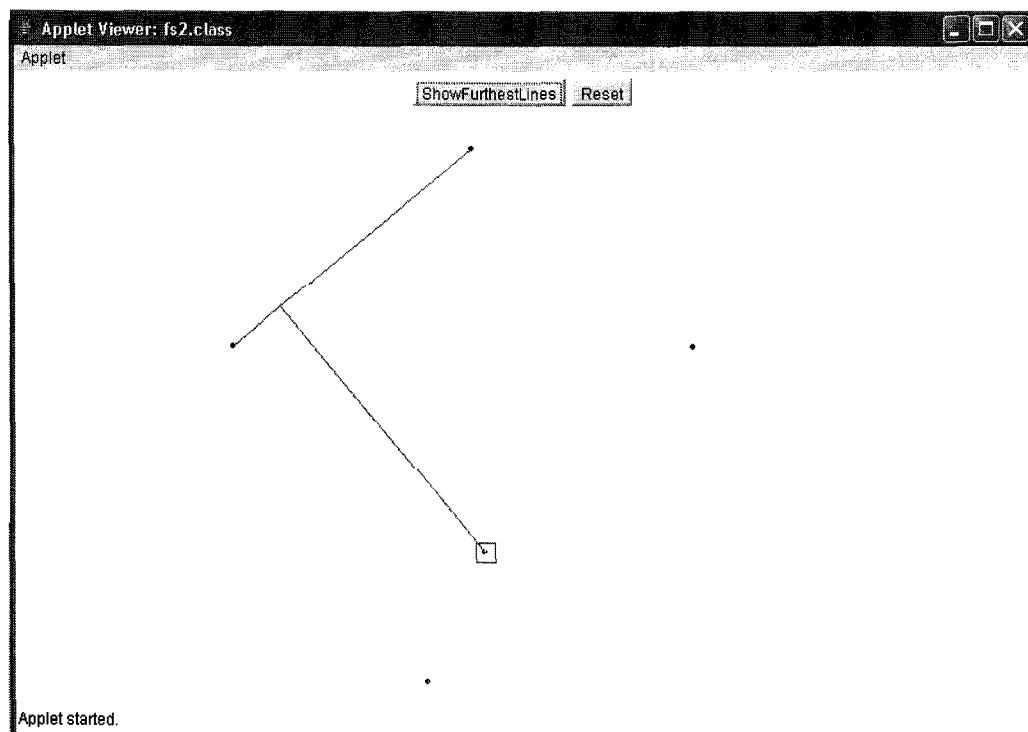


Figure 4.5: *Point 4, Farthest segment is a convex hull edge*

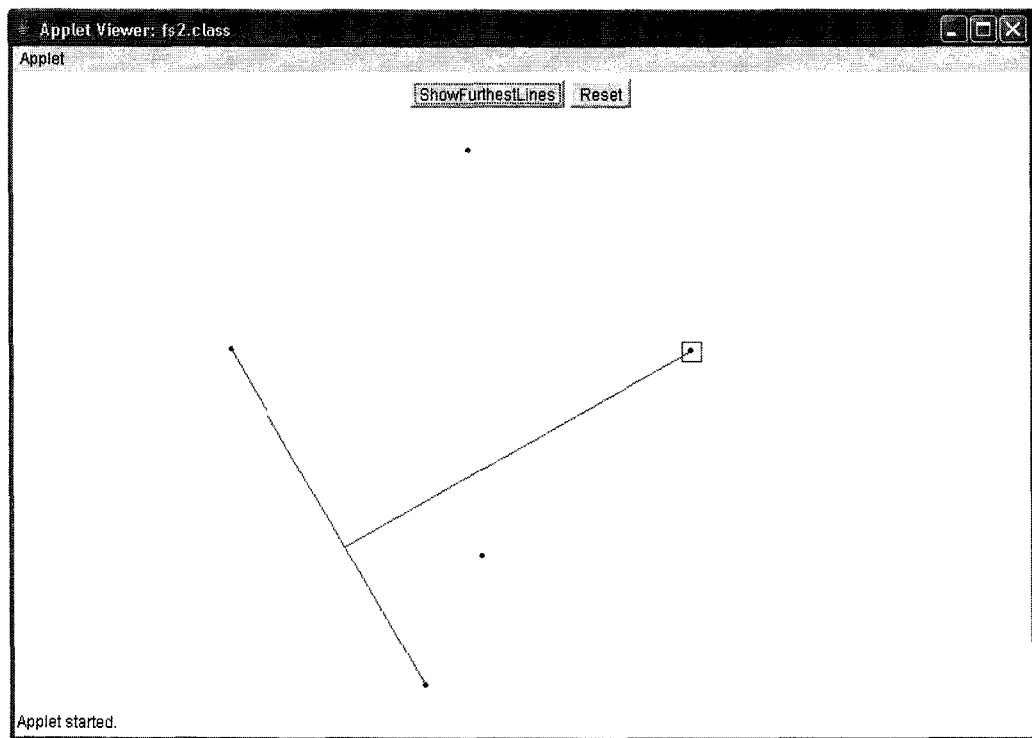


Figure 4.6: *Point 5, Farthest segment is a convex hull edge*

4.2 Results on the second problem

The following pages show some snapshots of the software developed for a set of vertical line segments. How to use the software is described below.

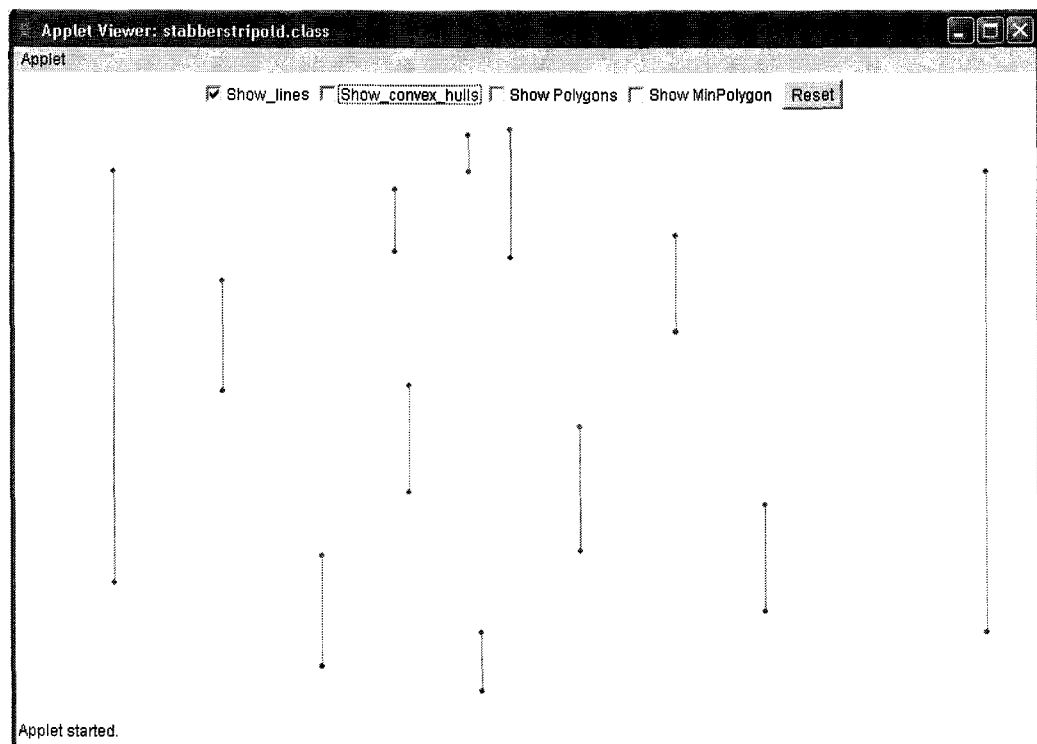
Step 1: Open the Applet Window clicking on the 'software' link on the homepage of Dr Asish Mukhopadhyay(<http://davinci.newcs.uwindsor.ca/~asishm>).

Step 2: The Applet Window appears. Click the checkbox "ShowLines". Now click the mouse at any place on the screen and drag it vertically downwards to some distance. Release the button and you have the first segment drawn. Even if due to personal error the drag is not exactly vertical, there is nothing to worry, it will automatically draw a vertical line from the starting point to the same y -value of the point where the mouse is released. Draw as many vertical segments as you can.

Step 3: Click on the checkbox "Show Convex Hulls". You will see the Upper Hull of the lower endpoints and Lower Hull of upper endpoints. Here you can add more line segments, the hulls will be dynamically updated.

Step 4: Click on the checkbox "Show Polygons". You will see the candidate polygons being shown. In the left you see the intervals showing all the candidate triangles, and a similar picture on the right. Here also, you may keep adding some segments, the figure will be dynamically updated.

Step 5: Click on the checkbox "Show Minimum Polygon". You see the final polygon of minimum area stabbing the line segments.

Figure 4.7: *Input Line Segments*

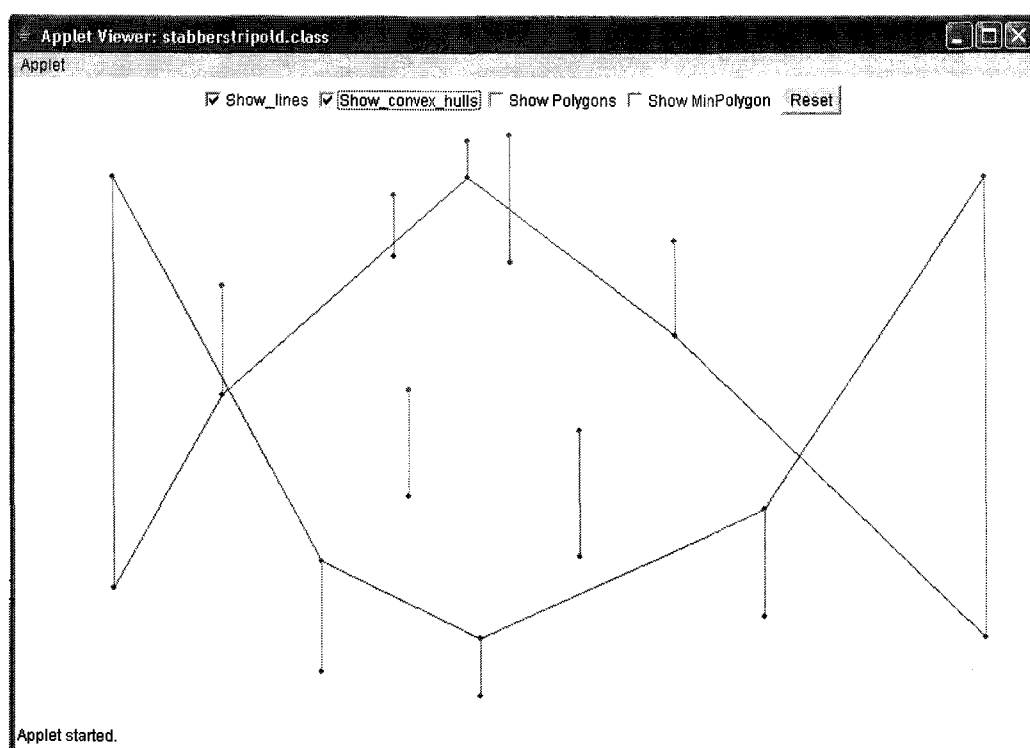


Figure 4.8: *Upper Hull of the Lower End Points and Lower Hull of Upper Endpoints*

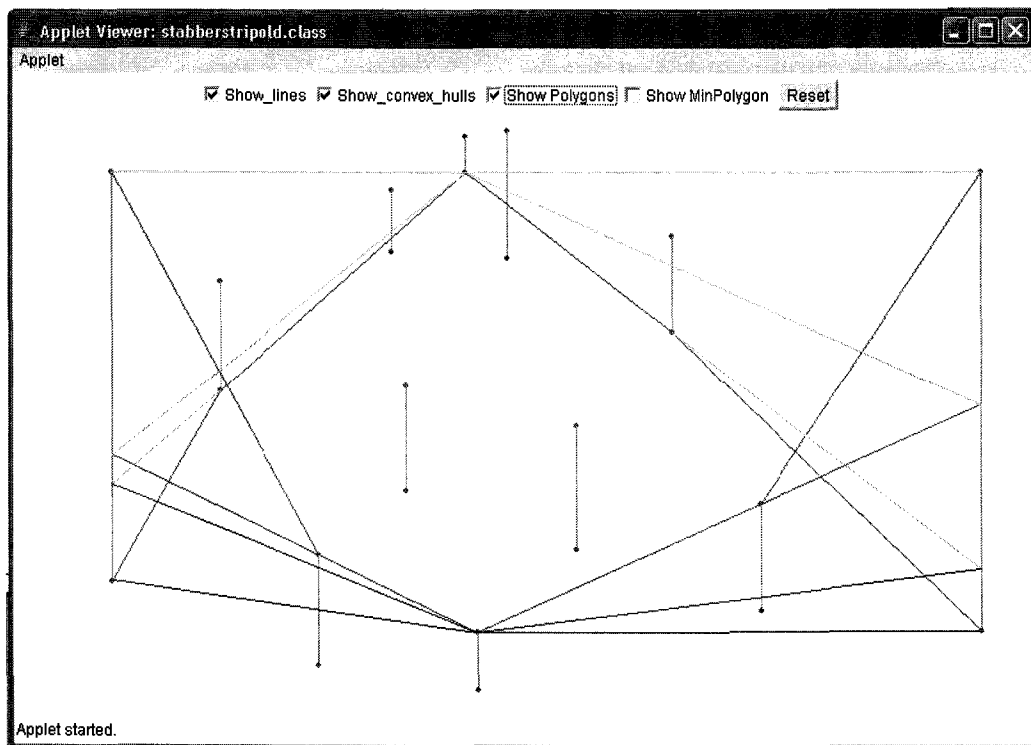
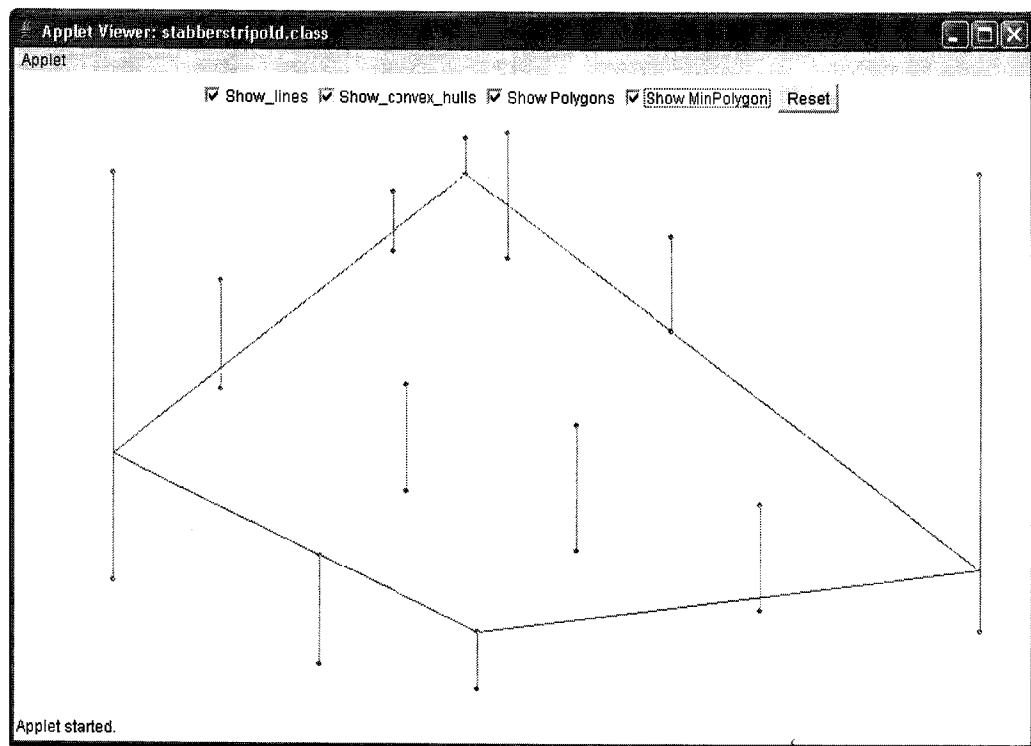


Figure 4.9: Searching for the minimum polygon in each interval

Figure 4.10: *The Minimum Polygon*

Chapter 5

Conclusions and Future Research

The first problem that we have worked had been accepted for presentation at the *European Workshop on Computational Geometry, 2006*(EWCG 2006). Dr Asish Mukhopadhyay, my advisor had delivered the talk. The paper has also been accepted for publication in a journal, namely *Information Processing Letters*. This work has some special significance due to two reasons. First of all, it is a completely new problem never been addressed before. After the paper had been presented at EWCG, Dr RL Scot Drysdale of Dartmouth College, USA opined that this algorithm can be improved. The improvement comes in where we search for the farthest convex hull segments. In this thesis we do that by a linear search, thus use $O(h)$ time, h being the number of vertices on the convex hull boundary. But this can be avoided, if we have a data structure (similar to the farthest point voronoi diagram) that can enable us to search for the farthest convex hull segments in $O(n \log n)$ time. Thus, our work has in fact led to the motivation of building a completely new data structure, called the *Farthest Segment Voronoi Diagram*, which was so long being unknown in literature. In (12), the authors used this data structure to improve the complexity to $O(n \log n)$.

One of the challenging open problems that this work poses is finding all k -th farthest segments for each point. One way of achieving this is to consider the k -th order voronoi diagram, instead of the farthest point voronoi diagram. If S is a set of

n points in a plane, any point in the k -th order voronoi diagram shares the same set of k closest points in S . It would be also interesting to see if a data structure similar to the one used in (12) can be designed, to address the problem of finding the k -th farthest convex hull segments in minimum possible time.

In the second problem, we have considered parallel line segments only (if we change the reference frame we can always have a set of parallel segments boil down to a set of vertical segments). The advantage we had here is that the intervals on the left and the rightmost segments are always invisible, due to the convex chains of the lower hull and the upper hull. Because of this, the global optimization problem of finding out the minimum polygon, eased down to solving two independent local optimization problems. But life will not be that easy, if we consider a set of isothetic line segments, that is when there are both vertical and horizontal lines. There, we will have two other extreme segments at the top as well at the bottom. So, we will have to consider four intervals at a time. It may happen that the interval on the top is visible to that on the left or right or both. Similar case may arise with the interval at the bottom. However, Dr Asish Mukhopadhyay as being stated earlier has considered all the cases carefully in (29) where he has proposed the $O(n^5)$ algorithm. The real finding of the work in this thesis is that the minimum polygon can only be obtained by partitioning the extreme line segments. This concept has also been used in the isothetic version of the problem. Partitioning the extreme line segments into intervals also enable us to see that the solution for the minimum local to an interval is at one of the endpoints. This finding holds good for the isothetic line segments version also.

But the real goal is still left to be addressed. If the line segments are of arbitrary orientation, it will be interesting to see whether our observations still hold fine, or things will change in that case. There are many things that need to be addressed. For example, in the very first step, one needs to see how the hulls can be drawn. In fact, whose hulls are needed to be drawn? Since they are of arbitrary orientation, we cannot characterize endpoints as bottom endpoints or top endpoints. It may be easier to first consider problems where the line segments may be of fixed orientation, and then move to the arbitrary version.

Bibliography

- [1] P.K. Agarwal and Sandeep Sen. Randomized algorithms for geometric optimization problems. *Handbook of Randomization*(P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim, eds.), Kluwer Academic Publishers, 2003.
- [2] P.K. Agarwal and M. Sharir. Algorithmic techniques for geometric optimization. *In Computer Science Today:Recent Trends and Developments,Lecture Notes in Computer Science*, 100(J.Van Leuween,ed.), 1995.
- [3] PK Agarwal and M Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete and Computational Geometry*, 16:317 – 337, 1996.
- [4] PK Agarwal and M Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30:412–458, 1998.
- [5] JL Bentley and MI Shamos. Divide and conquer in multidimensional space. In *Proc. of 8th ACM Symposium,Theory of Computation*, pages 220–230, 1976.
- [6] Timothy Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *GEOMETRY:Discrete and Computational Geometry*, 16:361–368, 1996.
- [7] Timothy Chan. On enumerating and selecting distances. *International Journal of Computational Geometry and Applications*, 11:291–304, 2001.
- [8] O. Daescu, J. Luo, and D.M. Mount. Proximity problems on line segments

- spanned by points. In *Proc. of 17th Canadian Conference on Computational Geometry*, pages 224–228, 2005.
- [9] Sandip Das, Partha P. Goswami, and Subhas C. Nandy. Smallest k point enclosing rectangle of arbitrary orientation. In *16th Canadian Conference on Computational Geometry*, pages 116–119, 2004.
- [10] Mark de Berg, Otfried Schwarzkopf, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd revised edition, 2000.
- [11] M. T. Dickerson, R. L. S. Drysdale, and J.-R. Sack. Simple algorithms for enumerating interpoint distances and finding l nearest neighbors. *Int. J. Comput. Geom. Appl.*, 2:221–239, 1992.
- [12] R.L. Scot Drysdale and Asish Mukhopadhyay. An $o(n \log n)$ algorithm for the all-farthest-segments problem for a planar set of points. Technical Report 06-011, School of Computer Science, University of Windsor, 2006.
- [13] K. Duffy, C. McAloney, H. Meijer, and D. Rappaport. Closest segments. In *Proc. of CCCG 2005*, pages 229–231, 2005.
- [14] Leila De Floriani, Enrico Puppo, and Paola Magillo. Applications of computational geometry in geographical information systems. *Chapter 7 in Handbook of Computational Geometry, Elsevier Science(J.R. Sack and J. Urrutia, eds.)*, pages 333–388, 1999.
- [15] B Gartner. Pitfalls in computing pseudorandom determinants. In *16th Annual Symposium on Computational Geometry*, pages 148–155, 2000.
- [16] M. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Computer vision, Graphics and Image Processing*, 49:152–170, 1990.
- [17] Paul Herron. <http://www.dma.fi.upm.es/mabellanas/tfcs/fvd/voronoi.html> , last retrieved on may 25, 2006.

- [18] N. Langrana. <http://www.caip.rutgers.edu/muri/>, last retrieved on may 25, 2006.
- [19] Y. Liu, H. Pottmann, and W Wang. Constrained 3d shape reconstruction using a combination of surface fitting and registration , <http://www.geometrie.tuwien.ac.at/ig/>, last retrieved on may 25, 2006. Technical Report 144, Vienna University of Technology, 2005.
- [20] Kelly A. Lyons, Henk Meijer, and David Rappaport. Minimum polygon stabbers of isothetic line segments. Department of Computing and Information Science, Queen's University, Ontario, Canada.
- [21] Jayanth Majhi, Ravi Janardan, Michiel Smid, and Prosenjit Gupta. On some geometric optimization problems in layered manufacturing. *Computational Geometry: Theory and Applications*, 12:219–239, 1999.
- [22] N Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.
- [23] N Megiddo. Applying parellel computaion algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.
- [24] N Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J.Comput.*, 12:759–756, 1983.
- [25] N Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31:114–127, 1984.
- [26] Geometric Modelling and Industrial Geometry. <http://www.geometrie.tuwien.ac.at/geom/fg4/>, last retrieved on may 25, 2006.
- [27] David M Mount. <http://www.cs.umd.edu/class/fall2005/cmsc754/lectures.shtml>, last retrieved on may 25, 2006.
- [28] Asish Mukhopadhyay. <http://davinci.newcs.uwindsor.ca/~asishm/software.html>.

- [29] Asish Mukhopadhyay. On intersecting a set of isothetic line segments with a convex polygon of minimum area. Technical Report 06-010, School of Computer Science, University of Windsor, 2006.
- [30] Asish Mukhopadhyay, Chanchal Kumar, and Binay Bhattacharya. Computing an area-optimal convex polygonal stabber of a set of parallel line segments. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, 1993.
- [31] Ketan Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice-Hall, 1994.
- [32] Rene Van Oostrum and Remco C. Veltkamp. Parametric search package:tutorial, 2005.
- [33] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
- [34] David Rappaport. Minimum polygonal transversals of line segments. *International Journal of Computational Geometry and Applications*, 5:243–256, 1995.
- [35] Sven Skyum. A simple algorithm for computing the smallest enclosing circle. *Information Processing Letters*, 37:121–125, 1991.
- [36] P. M. Vaidya. An $o(n \log n)$ algorithm for the all nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.
- [37] Alexander Wolff. <http://i11www.iti.uni-karlsruhe.de/~awolff/map-labeling/> , last retrieved on may 24, 2006.

Vita Auctoris

Samidh Chatterjee was born in 1978 in Kolkata, India. He graduated with a major in Statistics from St Xavier's College, Calcutta in 2000. He also obtained his Bachelor's degree in Computer Science and Engineering in 2003 from University of Calcutta. Samidh joined the University of Windsor in Fall 2004. He will be graduating in 2006. After his M.Sc., he plans to join Temple University, Philadelphia, USA for his doctoral degree. He would like to pursue research on computational biology during his PhD program at Temple University.