Optimal Enclosure Problems*

Esther M. Arkin**
Samir Khuller***
Joseph S. B. Mitchell****

TR 90-1111
March 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*A preliminary version of these results were presented at the 1st Canadian Conference on Computational Geometry, Montreal, Canada (Aug. '89).
**Partially support by NSF Grants DMC-8451984, ECSE-8857642, and DMS-8903304.
***Currently supported by an IBM Graduate Fellowship. Part of this research was done while this author was supported by NSF Grant DCR-85-52938 and PYI matching funds from AT&T Bell Labs. and Sun Microsystems.
****Partially supported by NSF Grants IRI-8710858, ECSE-8857642, and a grant from Hughes Research Labs.

Optimal Enclosure Problems *

Esther M. Arkin [†]
School of Operations Research
& Industrial Engineering
Cornell University
Ithaca, NY 14853

Samir Khuller [‡]
Computer Science Department
Cornell University
Ithaca, NY 14853

Joseph S.B. Mitchell [§]
School of Operations Research
& Industrial Engineering
Cornell University
Ithaca, NY 14853

Abstract

We consider the following "fence enclosure" problem: Given a set S of n points in the plane with values $v_i \geq 0$, we wish to enclose a subset of the points with a fence (a simple closed curve) so as to maximize the "value" of the enclosure. The value of the enclosure is defined to be the sum of the values of the enclosed points minus the cost of the fence. We consider various versions of the problem, such as allowing S to consist of points and/or simple polygons. Other versions of the problems are obtained by restricting the total amount of fence available and also allowing the enclosure to consist of up to K connected components. We show that the problem for a bounded length fence is NP-complete. Additionally we provide polynomial-time algorithms for many versions of the fence problem when an unrestricted amount of fence is available. When the set S consists of points and the fence is unrestricted in length we solve the problem via an $O(n^3)$ time sweep-line algorithm; we give an alternative $O(n^3)$ time algorithm based on finding shortest paths in directed graphs, and generalize it to handle various cases in which S is a set of polygons. When $K > 1$ components are permitted we obtain a polynomial-time algorithm (for constant K).

---

*A preliminary version of these results were presented at the 1^{st} Canadian Conference on Computational Geometry, Montreal, Canada (Aug '89).

†Partially supported by NSF Grants DMC–8451984, ECSE–8857642, and DMS–8903304.

‡Currently supported by an IBM Graduate Fellowship. Part of this research was done while this author was supported by NSF Grant DCR 85–52938 and PYI matching funds from AT&T Bell Labs. and Sun Microsystems.

§Partially supported by NSF Grants IRI–8710858, ECSE–8857642 and a grant from Hughes Research Labs.

1 Introduction

Problems that ask for an optimal polygon according to some objective function and subject to some set of constraints are prevalent in computational geometry. For example, [BDDG 85] looked at the problem of finding a maximum area/perimeter k-vertex convex polygon whose vertices come from a given set of n points. The "potato-peeling" problem of finding the maximum area convex polygon that fits inside a given simple polygon has been considered by [CY 88]. In this paper, we consider a class of problems that involves searching for an optimal polygon(s) in which the objective function is based on the "values" of objects (or points) that are enclosed.

We consider the following "fence enclosure" problem: Given a set $S = \{x_1, \ldots, x_n\}$ of n points in the plane with values v_i, we wish to enclose a subset of the points with a *fence* F (a simple closed curve) in such a way that the sum of the values of the points enclosed is maximized.

A more general statement of the enclosure problem is given in terms of the following parameters:

- The values v_i may be constrained to be nonnegative, or they may be unrestricted.
- There may be an upper bound $L \leq +\infty$ on the length of fence available.
- There may be an upper bound B on the number of bends allowed in the fence (i.e., on the number of "fenceposts").
- There may be a cost $c(\geq 0)$ per unit length of fence we use.
- We may be permitted to construct more than one enclosure. For example, we may be allowed to construct up to K (disjoint) enclosures, in which case the value of the enclosure is taken to be the sum of the values of the points inside all of the enclosures. In addition, there may be a fixed charge of E per enclosure.
- We may have constraints on the allowable shapes of the fence. For example, the shape may be unrestricted, convex, translates of a given shape, homothets of a given shape, etc.
- Rather than points with values, we may have more general (disjoint) regions with values; for example, S may be a set of (disjoint) simple polygons, each with a value.

In this paper, we concentrate on the two-dimensional case. The one-dimensional case is fairly straightforward, and we omit discussion of it here. Higher dimensional versions of the problems we address are still largely open.

We concentrate on the case in which $K = 1$. We suspect that problems in which K is part of the input are NP-hard. For constant K, we give polynomial time algorithms in Section 9.

Sections 2 and 3 contain some preliminary properties of optimal fences, and preprocessing algorithms that will be used later.

For the case in which S is a set of nonnegatively-weighted points in the plane (and fence costs per unit length), we solve the optimal enclosure problem in time $O(n^3)$ by a sweep-line algorithm (Section 4); we show how to formulate the problem as an all-pairs shortest path problem, yielding an alternative $O(n^3)$ algorithm (Section 5).

Furthermore, we show that if the length of fence is bounded above by $L < \infty$, then the problem is weakly NP-complete, and we provide a psuedo-polynomial algorithm for the case in which all the values v_i are integral (Section 6). We also provide an efficient algorithm that works well when L is small compared to the distances between the points themselves (and does not depend on the values of the points).

The case of general (unrestricted sign) weights is considered by Korte and Lovasz [KL 85], who develop an $O(n^5)$ algorithm for the case in which the fence (unrestricted in length) is restricted to be convex. When the fence is restricted to be translates or homothets of a given shape, standard "critical placement" arguments similar to [Ch 83], yield polynomial-time algorithms. Without constraints on the shape of the fence, the problem with unrestricted weights is seen to be NP-complete, by a reduction from the Euclidean Travelling Salesman Problem. Hence we will assume throughout the paper that the values of objects in S are all non-negative.

For the case in which S is a set of disjoint polygons that are considered to be obstacles (we cannot build a fence through a polygon), we again provide efficient algorithms based on shortest path constructions (Section 7). This algorithm takes $O(n^4)$ time and can also be implemented in NC. We also consider two other cases, when S consists of a set of disjoint polygons and we are permitted to build fences through some of the polygons: (1). The value of an obstacle is proportional to the fraction of the obstacle we include inside the fence; (2). The value of an obstacle is obtained only if the fence completely encloses it. Both these problems are discussed in Section 8.

Finally, we conclude with some open problems in Section 10.

2 Simple Properties of Optimal Fences

In this section we prove some simple properties about the shape of the optimal solution to the fence problem. We consider only the version of the problem in which S is a set of points having nonnegative values, and the fence costs c per unit length.

Lemma 1 *When all the values are positive, the optimum solution has a convex shape.*

Proof: Consider the fence boundary ∂S^* of the optimal solution S^*. If ∂S^* is not convex then consider the polygon S' which is the convex hull of the points enclosed by S^*. The enclosure S' contains all the points enclosed by S^* (possibly along with some others) and has a smaller perimeter contradicting the fact that S^* was an optimal enclosure.

□

Remark: Note that in three dimensions, the *convexity* property does not hold ! Consider four non-coplanar points in $\Re^3$; the optimal solution (minimum surface area "sheet") is not convex.

Definition 1 *Given points pq, let $H_{p,q}$ denote the closed half-plane left of the oriented line $\vec{pq}$.*

We often refer to the set of points in the closed half-plane $H_{p,q}$ as $H_{p,q}$.

Definition 2 *By $F_{p,q}(S)$ we denote the best enclosure from point set $S \cap H_{p,q}$ with the segment $\overline{pq}$ an edge in the enclosure. The best "supporting fence" to the left of oriented line segment $\vec{pq}$ is $F_{p,q}(S)$ (see Fig. 1).*

Lemma 2 *Consider an optimal solution S^* to the enclosure problem for a set S of points. For any pair of points $p, q \in \partial S^*$ the optimum supporting fence on $\vec{pq}$ is given by $\partial(S^* \cap H_{p,q})$.*

Proof: Consider the pair of points (p, q). Draw a line through p and q defining the half-plane $H_{p,q}$. Consider the optimal solution $S'_{p,q}$ for the point set $H_{p,q}$ with $\overline{pq}$ as a boundary edge. Assume $S'_{p,q}$ is different from $(S^* \cap H_{p,q})$ (see Fig. 2). It is easy to see that that the convex hull of the points enclosed by $S'_{p,q}$ and $S^* \cap (S - H_{p,q})$ yields a better solution than S^*, contradicting the fact that it was the optimum solution (see Fig 3).

□

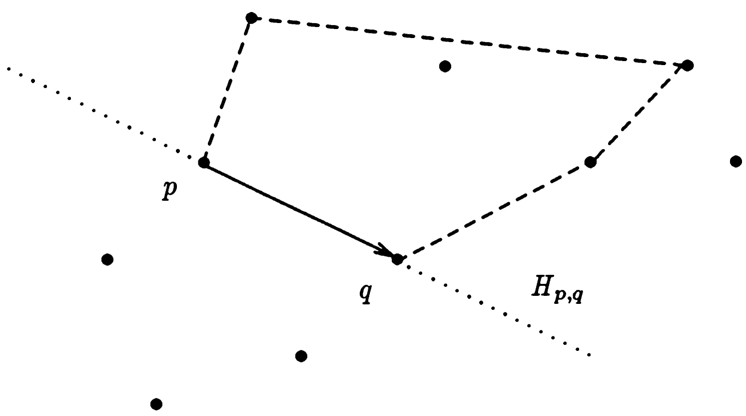


Figure 1: The best supporting fence for $\vec{pq}$

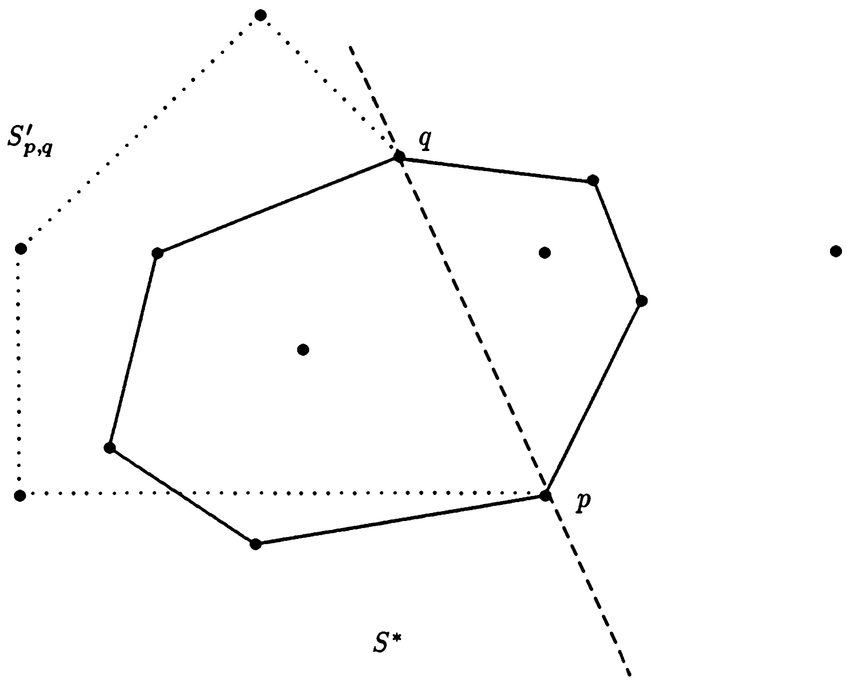


Figure 2: Solution for $F_{p,q}(H_{p,q})$ different from $S^* \cap H_{p,q}$

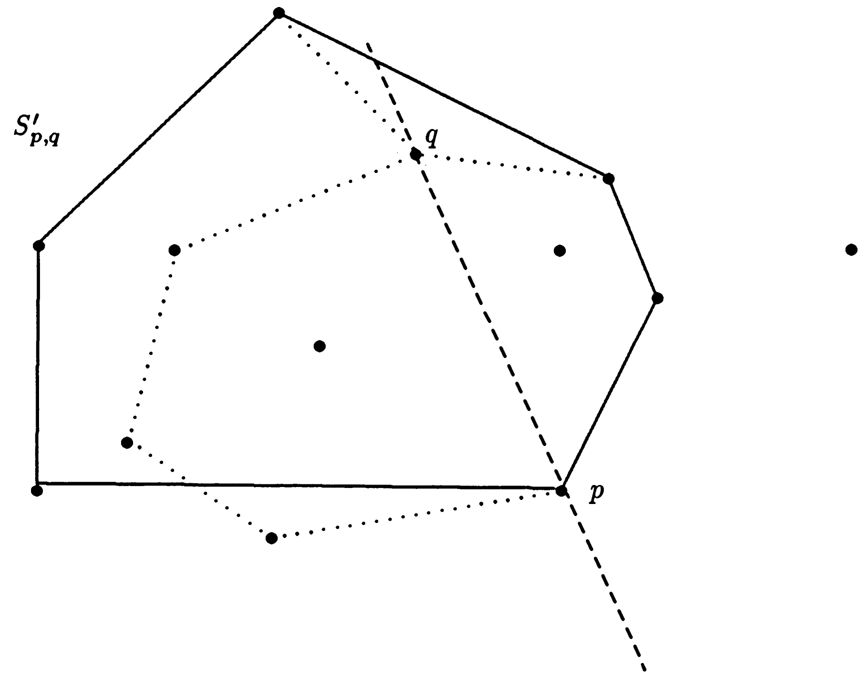


Figure 3: Combining $S'_{p,q}$ and $S^* \cap (S - H_{p,q})$

3 Triangle Queries

In this section we consider the following problem: Given a set S of n points in the plane, with values $v_1, v_2, ..., v_n$ we wish to preprocess the points so that we can efficiently answer queries of the following form:

Triangle query: Given a triple of points $(p, q, r)(p, q, r \in S)$ compute the sum of the values of the points in Δpqr.

We give an algorithm which does the preprocessing in time $O(n^2)$ and uses $O(n^2)$ space so that the queries can be answered in $O(1)$ time. We first provide a preprocessing algorithm which answers the following "wedge" and "half-plane" queries in $O(1)$ time; and then show how to use these queries to answer the triangle queries in $O(1)$ time. The triangle queries will be useful later on to develop algorithms to solve the optimum fence enclosure problem.

Wedge query: Given a triple of points (p, q, r), compute the sum of the values of the points enclosed in the wedge formed by the line segments $\vec{pq}$ and $\vec{pr}$ (going around clockwise from q to r).

Half-plane query: Given a pair of points (p, q), compute the sum of the values of the points in each of the two half-planes defined by the line passing through p and q.

In all the queries we find the values of the points strictly enclosed by either the triangle, wedge or half-plane (the value of the points p, q and r is ignored). To simplify the discussion we will also assume that no three points are collinear.

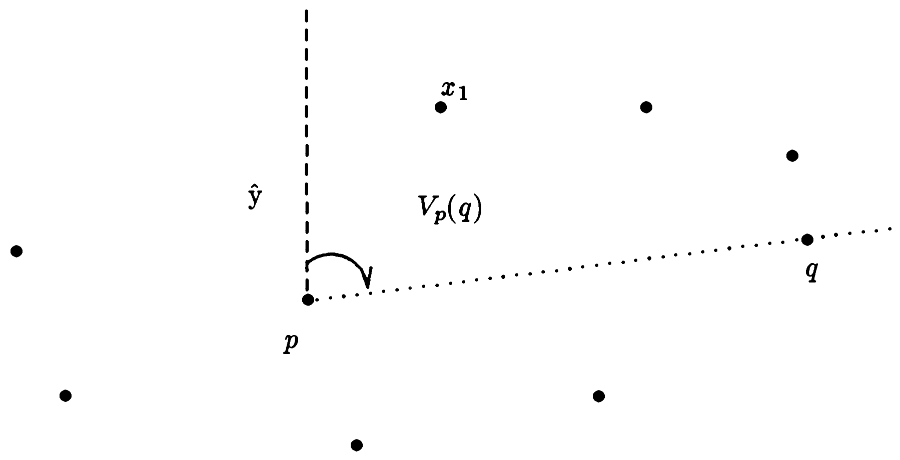


Figure 4: Points angularly sorted about point p

Preprocessing Algorithm for wedge queries

Step 1: Sort the points angularly about each point.

Step 2: For each point p do the following:

> Draw a line $\hat{y}$ from p vertically upwards (see Fig. 4) and walk around the points sorted about p in clock-wise order at each step computing the values $V_p(q)$ (which is the total value of the points in the open wedge formed by the segments $\hat{y}$ and $\vec{pq}$. This value is easily computed by simply keeping track of the sum of the values of the points as we walk around the points in order.

It is easy to see that the algorithm runs in $O(n^2)$ time. For Step 1 we use the algorithm in [LC 85] which solves the problem by mapping the problem to its dual. Step 2 takes $O(n)$ time for each point once the sorted list of points is available for each point p.

Wedge queries $V(p, q, r)$ are answered as follows: Depending on the angles of the segments, there are two cases to consider, which are similar. The cases to consider are (see Fig. 5 and Fig. 6):

1. $V(p, q, r) = V_p(r) - V_p(q) - v_q$
2. $V(p, q, r) = \sum_{i \neq p} v_i - V_p(q) - v_q + V_p(r)$

We use the values of $V_p(r)$ and $V_p(q)$ computed by the algorithm given above. Thus we conclude:

Lemma 3 *We can answer wedge queries of the type $V(p, q, r)$ to compute the sum of the values of the points in the wedge defined by p, q, r in $O(1)$ time. The preprocessing takes $O(n^2)$ time and uses $O(n^2)$ space.*

The algorithm which does the preprocessing for the half-plane queries is very similar in spirit to the above algorithm.

Algorithm for half-plane queries

Step 1: Sort the points angularly about every point.

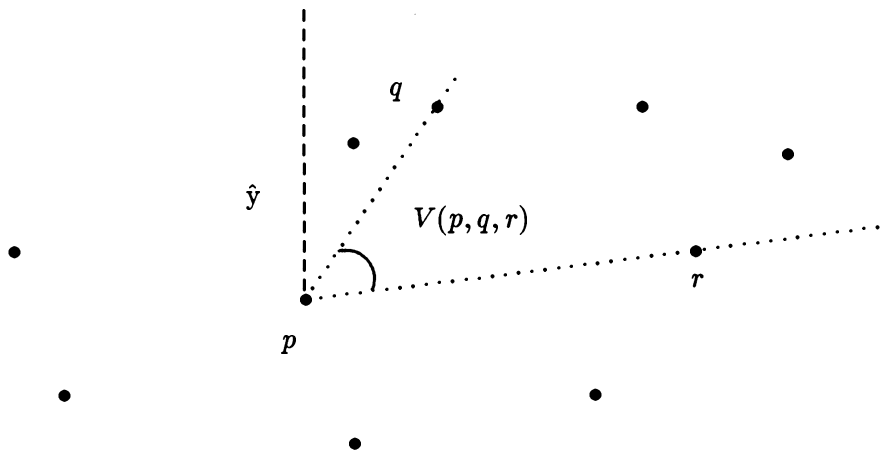


Figure 5: Computing wedge query $V(p,q,r)$

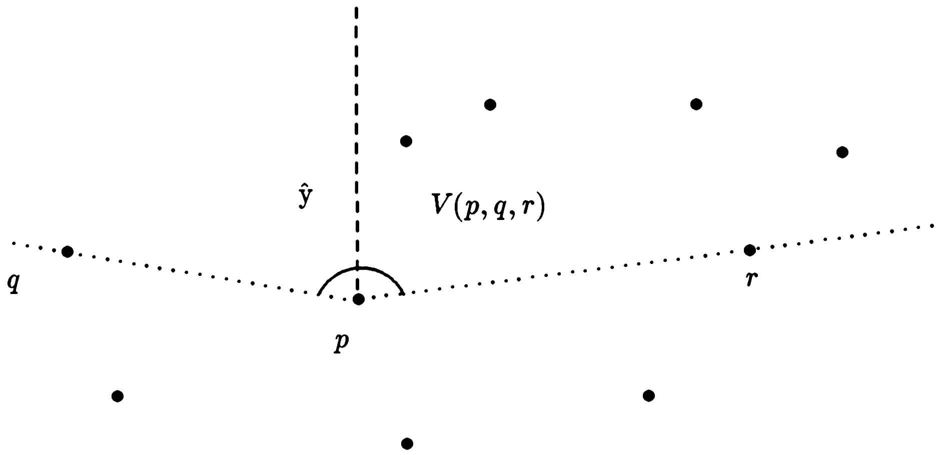


Figure 6: Computing wedge query $V(p,q,r)$

Step 2: For each point p do the following:

> Draw a line $\hat{y}$ from p vertically upwards (see Fig. 4) and for the first point in clockwise order (x_1) we compute the sum of the values of the points in each half plane formed by x_1 and p. Now we rotate a line segment around p and as each point moves from one half-plane to the other we compute the sum of the points for each half-plane with one of the points defining it as point p.

This algorithm computes the required information precisely, so that the half-plane queries can be answered directly. Thus we conclude:

Lemma 4 *We can answer half-plane queries of the type (p, q) to compute the sum of the values of the points in the half-planes defined by the pair of points (p, q) in $O(1)$ time . The preprocessing takes $O(n^2)$ time and uses $O(n^2)$ space.*

We now show that the triangle queries can be formulated as a sequence of wedge and half-plane queries as follows.

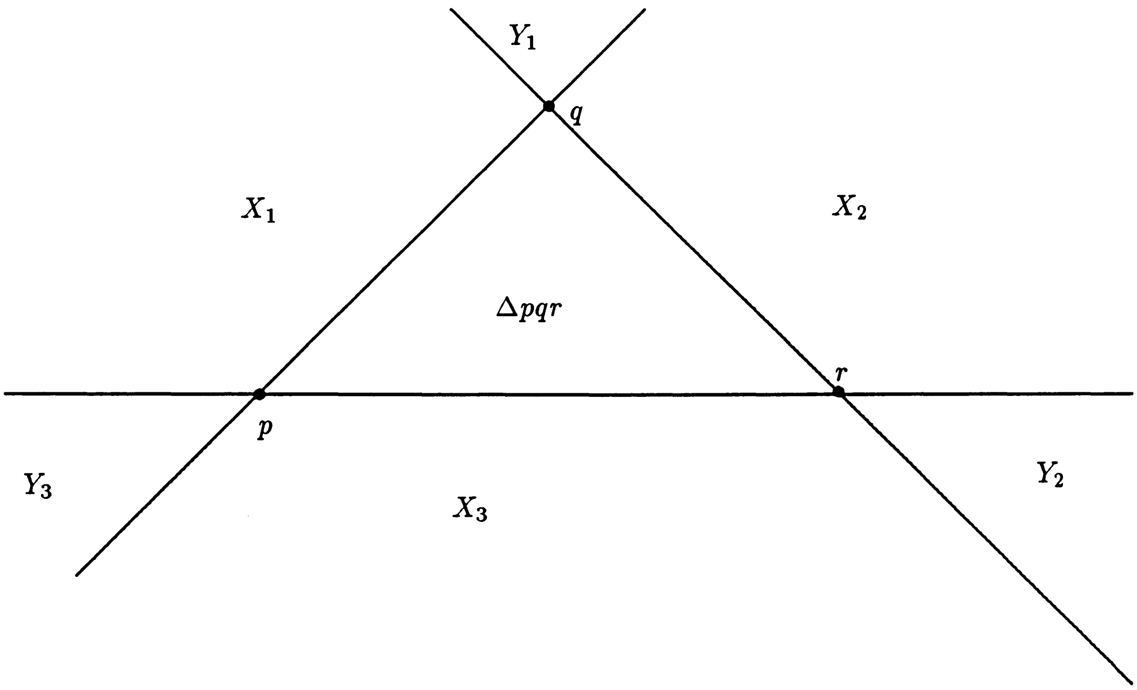


Figure 7: Solving the triangle query

Let the triangle Δpqr be as shown in Fig. 7. Define $V_p(q, r)$ to be the sum of the values of the points contained in the half-plane defined by the pair (q, r) and containing p. This is easy to compute given the values of both the half-planes defined by the pair (q, r).
Define

$$V_1 = V(p, q, r) + V(q, r, p) + V(r, p, q)$$

$$V_2 = V_p(q, r) + V_q(p, r) + V_r(p, q)$$

$$V_3 = v_p + v_q + v_r$$

$$V = \sum_{i=1}^{n} v_i$$

It is easy to see that the following are true:

$$3 \times \Delta pqr + X_1 + X_2 + X_3 = V_1 \tag{1}$$

$$3 \times \Delta pqr + 2(X_1 + X_2 + X_3) + Y_1 + Y_2 + Y_3 = V_2 - V_3 \tag{2}$$

$$\Delta pqr + X_1 + X_2 + X_3 + Y_1 + Y_2 + Y_3 + V_3 = V \tag{3}$$

The X_i's and Y_j's are defined to be the sum of the weights of the points in the regions as shown in Fig. 7. Solving the above equations we get

$$\Delta pqr = V_1 - V_2 + V.$$

Theorem 1 *We can answer triangle queries of the type Δpqr to compute the sum of the values of the points in the triangle defined by p, q, r in $O(1)$ time. The preprocessing takes $O(n^2)$ time and uses $O(n^2)$ space.*

4 Sweep-line algorithm

In this section we consider the version of the problem in which the fence is unrestricted in length and costs c per unit length. We wish to maximize the "value" of the enclosure (which is the value of the points enclosed minus the cost of the fence). We solve the problem via an $O(n^3)$ time sweep line algorithm, which we outline below.

The main idea is to sweep the point set from left to right, and at each step k, maintain information about the best supporting fences on each line segment formed by a pair of points.

Define $S_k = \{x_i \mid 1 \leq i \leq k\}$ ($S_k = \emptyset$ when $k = 0$) assuming that the points are sorted in their x-coordinates.

When the sweep-line is at point x_k, for each pair of points x_j, x_k ($j < k$) we obtain $F_{jk}(S_k)$ and $F_{kj}(S_k)$. By $F_{ij}(S)$ we denote $F_{x_i,x_j}(S)$. The sweep line algorithm actually computes $F_{jk}(S_k)(\forall j < k)$, when the sweep line passes through point x_k. Similarly we compute $(\forall j < k)F_{kj}(S_k)$.

As we move the sweep line over a new point x_k in the set S we need to compute the new supporting fences for each line segment formed with x_k as the right endpoint. We will assume that all the supporting fences for the line segments already swept have been computed.

Definition: Let $V(j,k) =$ Optimum value of supporting fence for oriented pair (x_j, x_k). We assume that $j < k$, and the value of the pair is a chain from j to k on the left side of the line segment $\overline{x_j x_k}$. The values $V(k,j)$ for the oriented pair (x_k, x_j) can similarly be computed. We define l_{ij} to be the euclidean distance between the points x_i and x_j. We define Δijk to be the sum of the values of points strictly enclosed by the triangle defined by x_i, x_j and x_k.

Sweep-line algorithm

Step 1: Sort the points in their x-coordinates.

Step 2: Assume that the sweep-line currently is at point x_k (initially $k = 1$). For point x_k we do the following:

1. Compute $\forall j < k$

$$V(j,k) = \max_{i \in S_k \cap H_{x_j,x_k}} [V(j,i) + \Delta ijk + v_k - c(l_{ik} + l_{jk} - l_{ij})]$$

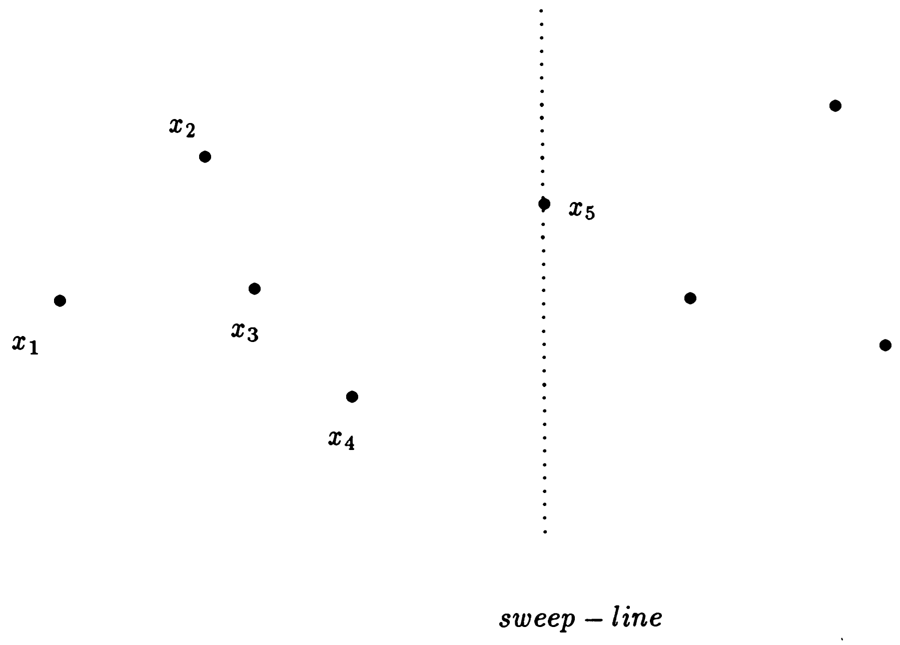


Figure 8: Sweep line algorithm

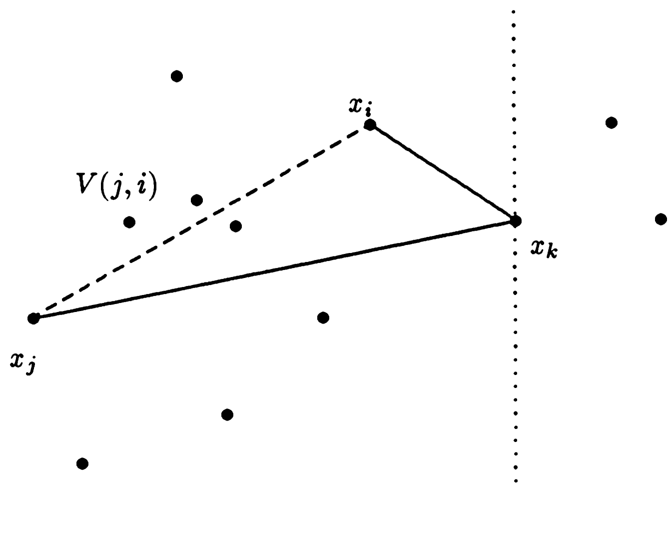


Figure 9: Sweep line at point x_k

2. $V(j,k) \leftarrow \max(v_j + v_k - 2cl_{jk}, V(j,k))$

In Step 2.1, we assumed that i is to the left of $\vec{jk}$, when that is not the case the equation can be modified by replacing $V(j,i)$ by $V(i,j)$. The invariant is that when the sweep-line passes through point x_k, $V(j,k)$ denotes the optimum value of $F_{jk}(S_k)$. The above algorithm produces a set of enclosures, which all include atleast two points in them. To compute the best solution (which may be just a singleton point) we compare each v_i with the following set of values: $\{V(i,j) \mid \forall i, \forall j\}$ and choose the best one as the fence. The following theorem proves that we do indeed obtain the best enclosure possible:

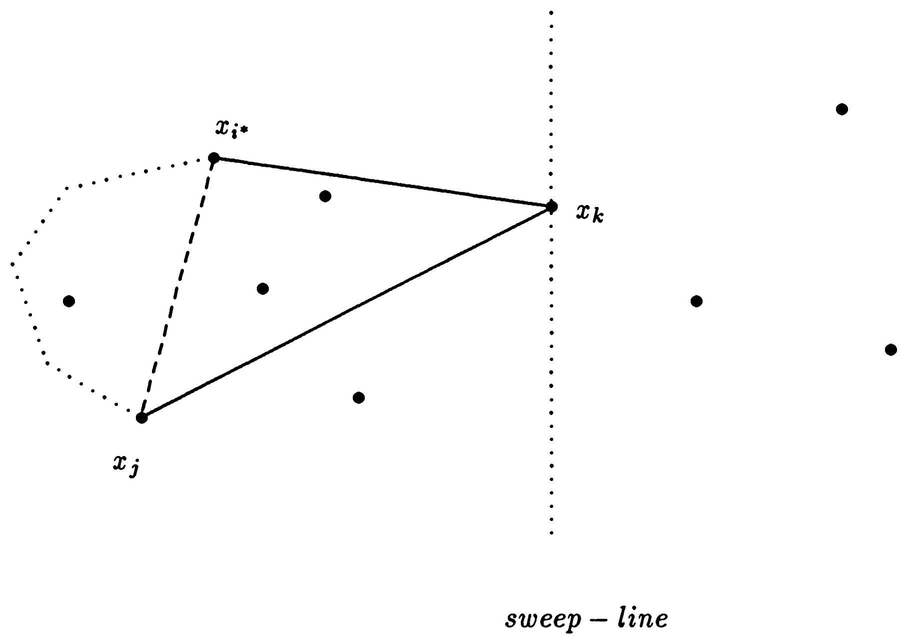


Figure 10: Computation of $V(j,k)$

Theorem 2 *$V(j,k)$ is computed correctly at each step (i.e., = value of $F_{jk}(S_k)$).*

Proof: Proof by induction on k. We assume that $V(i,l)$ has been computed correctly $\forall l < k$ and $\forall i < l$. Consider the optimal solution $F_{jk}(S_k)$. There are two cases to consider

1. The optimal solution encloses only the points x_j and x_k.

2. The optimal solution has a third point.

Case 1 is considered in Step 2.2 of the algorithm, for case 2 we consider (of all boundary points in the solution) the point i^* with the largest x-coordinate (see Fig. 10). In our maximization step clearly we consider $i = i^*$, and assuming $F_{ji^*}(S^*)$ had been computed correctly, we obtain the optimal solution to $F_{jk}(S_k)$.

□

Theorem 3 *There exist i, j such that $V(i,j)$ = value of S^*.*

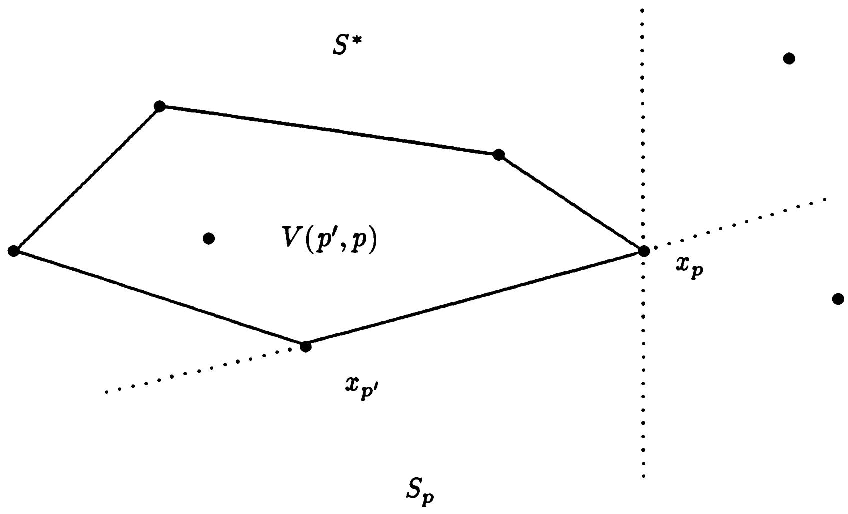


Figure 11: Optimal solution S^*

Proof: Let us consider the optimum enclosure S^* and show that the sweep-line algorithm would consider it as one of the solutions. Let the point with the rightmost coordinate be point x_p. Let $x_{p'}$ be the neighbour of x_p, on the lower chain of ∂S^* (see Fig. 11). Clearly, $V(p',p)$ is the optimal solution (and equals the value of S^*).

□

Analysis: Sorting the points by their x-coordinates requires $O(n \; log \; n)$ time. Let us consider the amount of work done by the algorithm for each point swept by the sweep-line. The sweep-line algorithm does $O(n)$ work for the maximization step, assuming that the triangle queries can be answered in $O(1)$ time. This is done for all values of i, resulting in $O(n^2)$ work each time Step 2 is executed. Thus the algorithm runs in $O(n^3)$ time.

This algorithm computes the value of the optimum enclosure, and not the enclosure itself. It is easy to modify the algorithm to compute the actual enclosure as follows: at each step, we keep track of the index that achieves the maximum value. Now, by a simple backtracking strategy we can compute the actual enclosure having the optimum value.

5 Shortest Path formulation

In this section we develop another algorithm for the problem considered in Section 4 by showing how to reduce the optimal enclosure problem to an *all-pairs shortest path* problem in directed graphs. This yields an $O(n^3)$ time sequential algorithm to solve the problem. Since shortest paths in directed graphs can also be obtained in NC, this also yields a parallel algorithm to solve the fence problem.

Given the set S of n points x_i in the plane (we will assume that the points are sorted by their x-coordinates), define the complete weighted directed graph $G(V, E)$ as follows:

$$V = \{i \mid x_i \in S\}$$

$$E = \{(i, j)\}$$

Definition 3 *Let* $U_{i,j} = \{x_k \mid (x_k \text{ "above" } \overline{x_i x_j}) \wedge (i < k < j)\}$ *and* $L_{i,j} = \{x_k \mid (x_k \text{ "below" } \overline{x_i x_j}) \wedge (i < k < j)\}$.

Definition 4 *Let* $L_i = \{x_k \mid k < i\}$ *and* $R_i = \{x_k \mid k > i\}$.

Definition 5 *Let* $cost(L_i) = \sum_{k \in L_i} v_k$. *We define* $cost(R_i)$ *similarly.*

We define

$$cost(i,j) = \begin{cases} cl_{ij} + \sum_{k \in U_{i,j}} v_k & \text{if } i < j \\ cl_{ij} + \sum_{k \in L_{i,j}} v_k & \text{if } i > j \end{cases}$$

Now compute the shortest paths $sp(i,j)$ between all pairs of vertices i, j. The optimum solution is obtained as follows (assuming $i < j$):

$$V^* = \min_{i,j}[sp(i,j) + sp(j,i) + cost(L_i) + cost(R_j)].$$

$$\text{Value of } S^* = \sum_{k \in S} v_k - V^*$$

The term in the minimization is essentially summing the cost of the enclosure (in terms of fence length) together with the sum of the values of the points *not* enclosed by the fence (see Fig. 12).

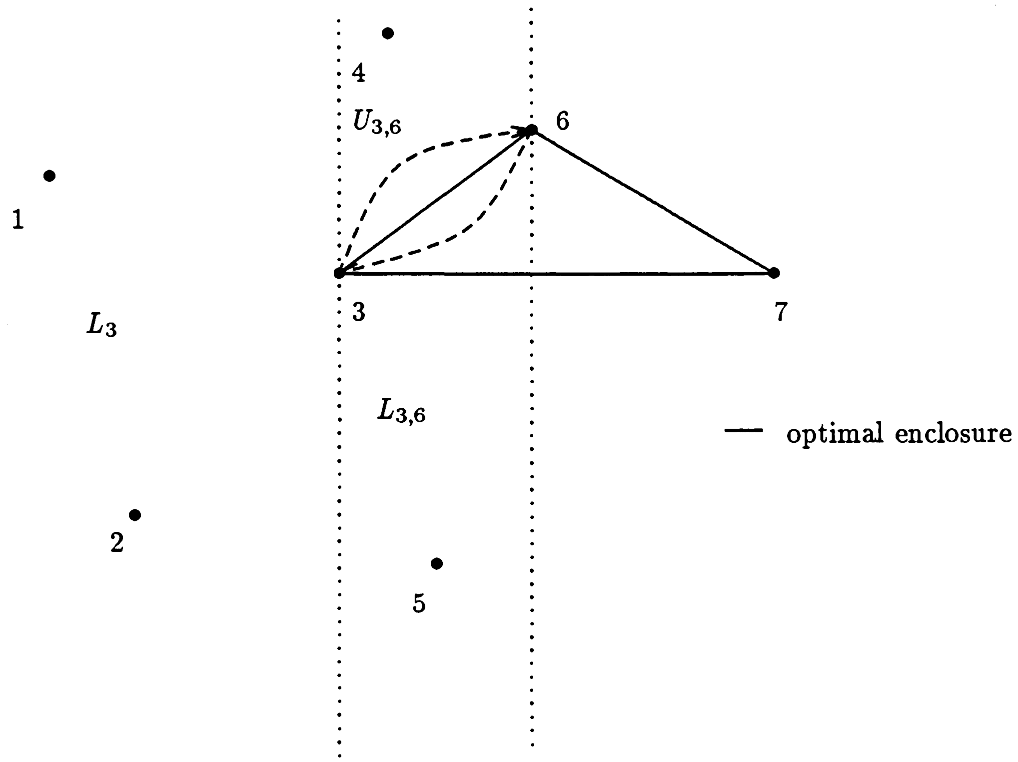


Figure 12: Optimal solution corresponding to S^*

Theorem 4 *The algorithm correctly obtains the optimum enclosure.*

Proof: Let S^* be an enclosure with maximum value. Let the points with the smallest (largest) x-coordinates on ∂S^* be i^* and j^* respectively. Then, there is a term in the minimization (i^*, j^*) with value $(\sum_{k \in S} v_k - S^*)$. If the minimum is achieved for any other pair (i', j') then we can obtain an enclosure with a higher value than S^*.

□

This algorithm computes the best enclosure with at least two points in it. We can compare the obtained solution with all the enclosures containing only a singleton point.

Analysis: The construction of the graph can be done in $O(n^3)$ time very simply. The shortest path computation also takes $O(n^3)$ time (for an all-pairs shortest path computation) [PS 82]. The minimization is over $O(n^2)$ terms, and thus the entire algorithm runs in $O(n^3)$ time. If the shortest paths are also computed along with the the lengths, we can obtain the actual enclosure easily.

6 Bounded Length Fence

We prove that the following version of the two-dimensional fence enclosure problem is NP-hard by showing a polynomial time reduction from the *Knapsack* problem, which is known to be NP-complete [GJ 78].

Fixed Length Fence Enclosure: Given a set S of n points $x_i \in \Re^2$, each point x_i is associated with a value v_i, and real numbers V and L. Is there a subset S' of the points such that $\sum_{j \in S'} v_j \geq V$ and the perimeter of the convex hull of the points in S' is $\leq L$?

Knapsack Instance: Given a set U, and for each element $i \in U$, a weight w_i, and a value v_i, and positive integers W and C. Is there a subset $U' \subseteq U$ such that $\sum_{j \in U'} w_j \leq W$ and $\sum_{j \in U'} v_j \geq C$?

Theorem 5 *The Fixed Length Enclosure problem is NP-hard.*

Proof: Assume that the set U has cardinality n (in the instance of the knapsack problem). We create a set S of $2n$ points in $\Re^2$ as follows ($S = A \cup B$): Set A consists of n points placed on the vertices of a regular convex n-gon. Assume that the points are labeled $p_1, p_2, ..., p_n$ in clockwise order on the boundary of the n-gon. The value of each point $p_i \in A$ is K (which is a very large constant). Assigning very large values to points in A ensures that all of them are chosen to be in the enclosure (provided a sufficient amount of fence is available). These points effectively "fix" the general outline of the optimal enclosure. The value of L is such that we can include all the points of set A in the enclosure.

The points in set B (of cardinality n) correspond to items in set U. Point $q_i \in B$ (corresponding to $i \in U$) is placed off the side $\overline{p_i p_{i+1}}$ of the n-gon (see Fig. 13). We draw the bisector of the angle $\angle p_i c p_{i+1}$ (where c is the center of the n-gon), and extend it beyond the intersection with segment $\overline{p_i p_{i+1}}$. The distance of q_i from c is determined by w_i (described later). The value of q_i is taken to be the value of the item i, i.e., v_i.

We would like the items that are chosen to be in the Knapsack in the optimal solution to correspond exactly to the items of B that are chosen to be in the optimal enclosure. To achieve that we set $V = nK + C$. (The value is obtained by choosing the n points in set A and a subset of the other points of value at least C.) We set $L = \text{perim}(\text{ConvHull}(A)) + sW$ (where s is an appropriate scale factor).

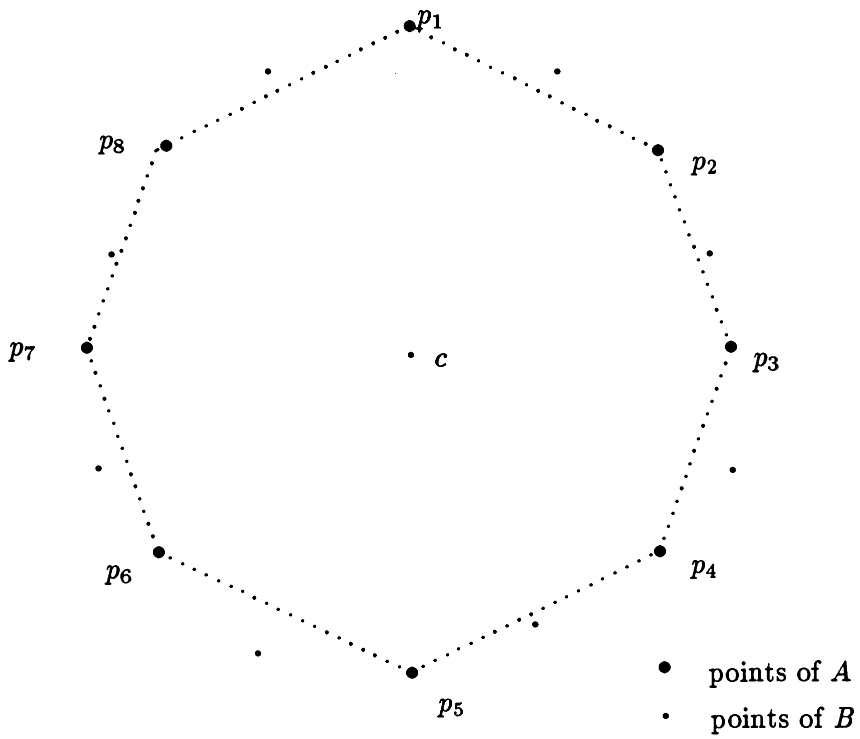
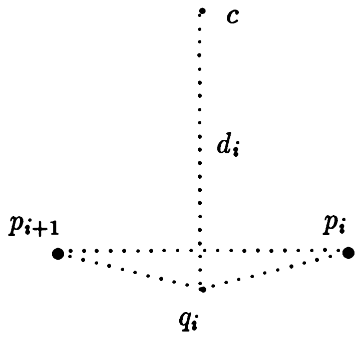


Figure 13: Positioning of points in set S



$$dist(q_i p_{i+1}) + dist(q_i p_i) - dist(p_i p_{i+1}) = sw_i$$

Figure 14: Placement of point q_i

The points in set B are placed so that the boundary of the enclosure (and not the interior) contains all the points in A; the scale factor s is chosen to ensure this property. The points q_i are placed at a distance d_i from the side $\overline{p_i p_{i+1}}$, so that the extra fence required to include q_i is sw_i (see Fig. 14).

Notice that the coordinates of some of the points may be irrational. It is easy to see that the points can be perturbed slightly to place them at rational coordinates. This is done by perturbing them all away from the center and providing a small amount of extra fence to take into account the perturbation. It is easy to see that the reduction can be performed in polynomial time.

□

The above reduction shows that the problem is NP-hard. We now provide a pseudo-polynomial time algorithm to solve the problem.

Pseudo-Polynomial time Algorithm

We present two algorithms to solve the version of the problem when the fence is of a bounded length L. The first algorithm runs in polynomial time when the values of the points are small integers. The second algorithm performs well when L is not very large compared to the smallest distance between two points (the values of the points may be large).

Theorem 6 *There is a pseudo-polynomial time algorithm to solve the fixed length enclosure problem.*

Proof: The algorithm is based on dynamic programming and is outlined below. We wish to compute $f(x_j, x_k, K)$ which is the least length of fence required to enclose at least value K with a "supporting fence" to the left of $\overline{x_j x_k}$. We assume that the values of x_j and x_k are included in the supporting fence. We also assume that x_i is a point in the half-plane left of $\overline{x_j x_k}$.
$f(x_j, x_k, K)$ is computed as follows:

$$f(x_j, x_k, K) = \begin{cases} l_{jk} & \text{if } v_j + v_k \geq K \\ \min_{x_i}\{f(x_i, x_k, K - (\Delta ijk + v_j)) + l_{ij}\} & \text{for all } x_i \text{ to the "left" of } \overline{x_j x_k} \\ \infty & \text{otherwise} \end{cases}$$

Notice that to compute any $f(p, q, K)$ we need the value of $f(*, *, K')$ where $K' < K$. Thus the function $f(p, q, K)$ can be computed in increasing values of K for all pairs of points p and q. The first values to be computed are $f(p, q, 1)$, for each pair of points p, q.

To compute the optimal solution we wish to compute the largest value of K, such that for any pair of points x_j, x_k the following holds:

$$(l_{jk} + f(x_j, x_k, K)) \leq L.$$

The optimal solution could also be a singleton point, and thus we compare K with each singleton value to obtain the optimal solution. We call this value K^* and it represents the optimal solution to the problem.

Complexity: The algorithm computes a table of size $O(Kn^2)$ (where K represents the sum of the values of all the points). It costs us $O(n)$ time to compute each entry in the table, and thus the running time of the algorithm is $O(Kn^3)$. The space complexity is $O(Kn^2)$.

□

We present an alternative algorithm to solve the problem. This algorithm has the property that it works well for lengths of rope that are small compared to the distances between the points. It may take exponential time in the worst case.

By $f(x_j, x_k, X)$ we denote the largest value that can be enclosed with a "supporting fence" to the left of $\overline{x_j x_k}$ with a perimeter less than X. We assume that the values of x_j and x_k are included in the value of the "supporting" enclosure.

$f(x_j, x_k, X)$ is computed as follows:

$$f(x_j, x_k, X) = \begin{cases} -\infty & \text{if } X < l_{jk} \\ \max_{x_i}(f(x_i, x_k, X - l_{ij}) + v_j + \Delta ijk) & \text{for all } x_i \text{ to the "left" of } \overline{x_j x_k} \\ v_j + v_k & \text{if previous term yields - } \infty \text{ for all } x_i \end{cases}$$

To compute the optimal solution, we compute the values of $f(p_j, p_k, L - l_{jk})$ for all pairs of points p_j, p_k. The computation runs in a demand driven fashion (rather than constructing a table) and essentially by making recursive calls to f. Notice, that in each successive recursive call the value of the parameter X is decreasing. For small values of L the depth of the recursion will be small. Again, the optimal solution can be a single point (if L is too small). We can compare the value of the most valuable singleton with the value computed by the above algorithm, to obtain the optimal solution.

7 Optimum enclosure with obstacles

We now consider a more general version of the fence enclosure problem with obstacles in the plane. A formal description of the problem is given as follows: Given a set S of disjoint simple polygons (obstacles), with obstacle O_i having value v_i, we wish to find an enclosure that maximizes "value"; where "value" is again defined to be the sum of the values of the obstacles enclosed minus the cost of the fence (where fence costs c per unit length). We do not permit the fence to be in the interior of any simple polygon (obstacle) in the set S. Thus, an obstacle is either entirely included or, entirely excluded from the enclosure. All points with values will be treated as obstacles for convenience. Hence, S consists entirely of obstacles, and we use $x_1, x_2, \ldots, x_n$ to denote the vertices corresponding to the points defining the obstacle boundaries. The optimum enclosure may not have a convex shape due to the presence of obstacles (see Fig. 15).

We can show that the shape of the optimum solution is geodesically convex (i.e., the shortest path between any two points in the region enclosed by the optimum solution, is also entirely inside the region). We provide an algorithm to solve the fence enclosure problem in the presence of obstacles, which is based on a reduction to an *all-pairs shortest path* problem in directed graphs. This yields an $O(n^4)$ time sequential algorithm to solve the problem. (We let n denote the size of the input, i.e., the total number of points describing the boundaries of the polygons.) Since shortest paths in directed graphs can also be obtained in NC, this yields a parallel algorithm to solve the fence problem for the two-dimensional version of the problem in the presence of obstacles.

Lemma 5 *An optimum solution is geodesically convex.*

Proof: Suppose that for points v_1 and v_2, belonging to an optimum enclosure S^*, the shortest path P, leaves the boundary of S^*. We can obtain a better solution by replacing a segment of δS^*, by the segment of P outside S^*. (Since, we shorten the boundary and also enclose more obstacles.) This is a contradiction to the assumption that S^* was an optimal solution.

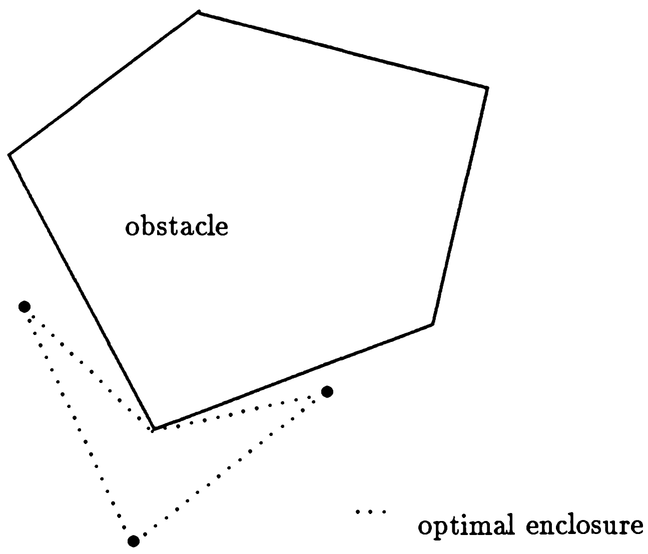


Figure 15: Optimal solution may not be convex

□

Outline of Algorithm:

Assume that point x^* is on the boundary of an optimum enclosure. The algorithm begins by finding shortest geodesic paths P_i from each vertex x_i (belonging to an obstacle in S) to x^*. Now consider all the vertices in angularly sorted clockwise order about x^*. We sort them in the angular order of the geodesic paths P_i emanating from x^*. We define $x_i \prec x_j$ iff x_i is strictly before x_j in the angular order $\mathcal{P}$ (if one of the geodesic paths, say P_i, is a subpath of P_j, then there is no ordering between x_i and x_j, and they are said to coincide). For example, in Fig. 16, the path $P_0 \prec P_1, P_0 \prec P_2, P_0 \prec P_3$, and $P_1 \prec P_3$. But there is no ordering between P_1 and P_2, and also between P_2 and P_3 (and we say that they coincide).

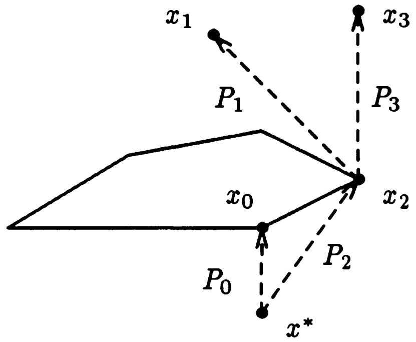


Figure 16: Example to ilustrate the ordering $\mathcal{P}$

We will now show that the vertices on the boundary of the optimum enclosure S^* (considered in clockwise order) never "back up", in the order $\mathcal{P}$.

Lemma 6 *Consider the optimum solution S^*, with point x^* on the boundary ∂S^* (of the enclosure). The clockwise ordering of the vertices on ∂S^* is consistent with the ordering $\mathcal{P}$.*

Proof: Assume that there are vertices x_i and x_j on ∂S^*, with x_i before x_j, and that $x_j \prec x_i$ (see Fig. 17). It is easy to see, that in either case we can shorten the boundary of S^*, increasing the region enclosed by the fence. This is a contradiction to the assumption that ∂S^* is an optimal enclosure. In the first case, we can replace the subpath $\partial S(x^*, x_j)$ by $P_j(x^*, x_j)$. In the second case, let u be the point of intersection of $\partial S(x_j, x_k)$ with P_i. Then $P_i(x_i, u)$ is shorter than $\partial S(x_i, u)$, and replacing the segment of the boundary by the segment of the P_i path also increases the region enclosed by S.

□

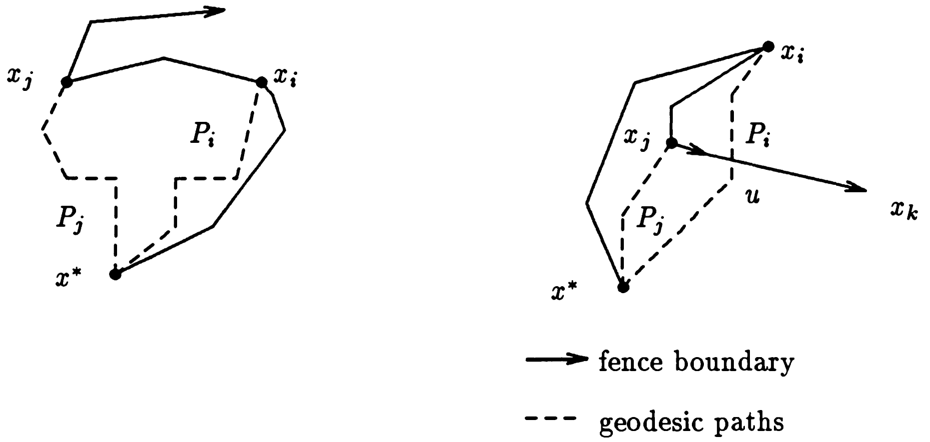


Figure 17: Boundary always progresses in $\mathcal{P}$

We now build a directed graph $G(V, E)$ as follows:

$$V = \{i \mid x_i \text{ is a vertex of an obstacle in } S\}$$

$$E = \{(i,j) \mid x_i \text{ is "visible" from } x_j, \text{ and it is not the case that } x_j \prec x_i\}$$

In other words, the edge set consists of all edges in the visibility graph of the set of points. The edges are directed in the following manner: edge (i, j) is directed from i to j if either $x_i \prec x_j$ in $\mathcal{P}$, or they coincide in $\mathcal{P}$.

We now show how to assign costs to the edges in such a way, that the cost of a cycle in G corresponds precisely to the sum of the values of the obstacles left out of the enclosure, together with the cost of constructing the fence. The shortest cycle in G corresponds to the optimal enclosure with x^* on the boundary of the enclosure.

We can assume that the boundary of the enclosure has at least three vertices, and it goes through x^* in a clockwise manner. Let (x^*, x^O) be the first outgoing edge from x^*, and (x^I, x^*) be the (last) incoming edge to x^*. For an illustration of the incoming and outgoing edges from x^* see Fig. 18.

For each obstacle O_i (which is not a point obstacle) with value v_i, we associate its value with a point q_i on its boundary (q_i is chosen to be a vertex on the boundary of the obstacle).

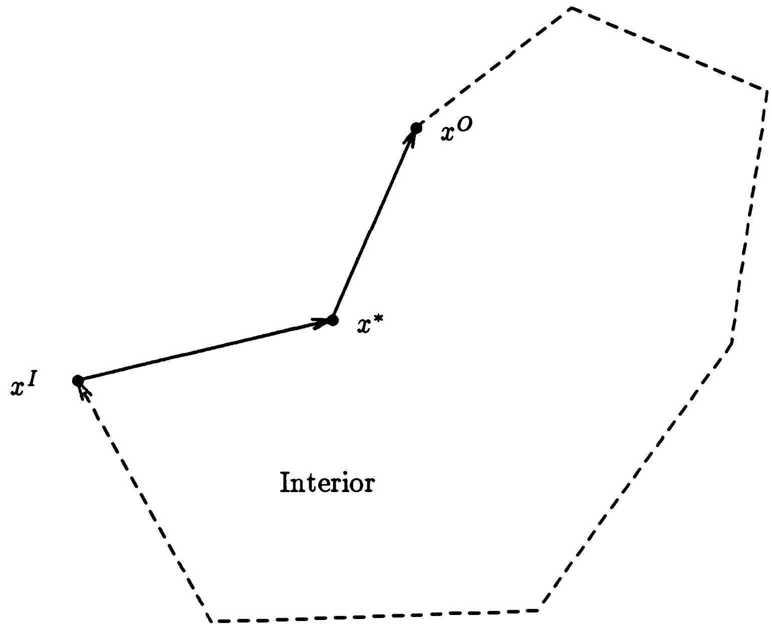


Figure 18: Incoming and Outgoing edges from x^*

The costs on the edges are assigned as follows: Consider an edge (x_i, x_j) such that x_j is "visible" from x_i. Consider the "triangle" (more precisely, geodesic triangle) formed by P_i, P_j and the line segment $\overline{x_i x_j}$. If $x_i \prec x_j$, then we define $Out(i,j)$ to be the set of obstacles O_k, such that the path from q_k to x^* goes through the segment $[x_i, x_j)$. Note that the interval is closed at one endpoint of the segment, to take into account the obstacles to whom the shortest paths go through x_i. The obstacles in $Out(i,j)$ are all excluded from the optimal enclosure.

$$c(i,j) = cl_{ij} + \sum_{O_k \in Out(i,j)} v_k$$

In other words, the cost of the edge corresponds to the length of the edge together with the cost of the obstacles left out due to this edge. If x_i and x_j coincide, then the cost of the edge $c(i,j) = cl_{ij}$ (this is the case when P_i is a subpath of P_j, or vice versa).

Define the set $O(x^I, x^O)$ to be the set of obstacles O_k, such that the path from q_k to x^*, is angularly strictly inbetween $\overline{x^I x^*}$ and $\overline{x^* x^O}$. The obstacles whose shortest paths contain x^I are also included in $O(x^I, x^O)$. (Note that the obstacles whose shortest paths contain x^O will be counted in the shortest path from x^O to x^I.) We associate a cost $c(x^O, x^I)$ with the pair x^O, x^I where

$$c(x^O, x^I) = c(l_{x^I x^*} + l_{x^* x^O}) + \sum_{O_k \in O(x^I, x^O)} v_k$$

Let X^O be the set of vertices that are "visible" from x^*, and it is not the case that $x^O \prec x^*$. We now find the shortest path tree in G, from each vertex $x^O \in X^O$ to all the vertices x^I, such that x^I is "visible" from x^* and it is not the case that $x^* \prec x^I$ (i.e., either x^I is "behind" x^* in $\mathcal{P}$ or coincides with x^*). The edges $(x^I, x^*), (x^*, x^O)$ together with the shortest path $sp(X^O, x^I)$ forms a cycle $C(x^*, x^O, x^I)$.

For a fixed x^* and x^O, we compute the folowing:

$$V(x^*, x^O) = \min_{x^I}[c(x^O, x^I) + sp(x^O, x^I)]$$

Taking the minimum over all $x^O \in X^O$ gives us the shortest cycle through x^*. This gives us the optimal enclosure passing through vertex x^*. Let $C(x^*)$ be the shortest cycle passing through x^*.

Theorem 7 *The cost of cycle $C(x^*)$ exactly equals the cost of constructing the fence, together with the values of the obstacles left out.*

Proof: To prove this theorem, we need to show that the value of an obstacle not enclosed by the fence is charged to exactly one edge in the cycle. Consider the shortest path of any obstacle O_k that is not inside the enclosure, to the vertex x^*. Either, the path cuts the boundary of the enclosure and enters the interior of $C(x^*)$ to reach x^*, or reaches x^* without ever entering the interior. In the latter case, it is easy to see that the value of obstacle O_k is charged to $c(X^O, x^I)$. In the former case, we need to show that the path from q_k to x^* crosses the boundary of the fence exactly once. If it does cut the boundary more than once, it is easy to see that the fence must "backup" in the order $\mathcal{P}$, which is a contradiction to Lemma 6. The actual cost of the fence is charged separately to each edge of the cycle.

□

Complexity: For a fixed x^* and x^O the cost of computing the shortest path tree is $O(n^2)$. Once we enumerate over all all such pairs the running time of the algorithm is $O(En^2)$, where E is the number of edges in the visibility graph. The worst case running time of the algorithm is $O(n^4)$.

8 Optimal Enclosure of Non-Obstructing Segments

In this section we consider the case in which S is a set of *non-intersecting* line segments with values, as in the previous section. In this case however the fence may cross line segments in S. There are two possible ways in which we can count the value of a fence that partially encloses a segment.

- The value obtained is proportional to the fraction of the line segment enclosed. Namely, if the fence encloses half of a line segment, the value obtained is half the value of the line segment. It is not hard to see how the ideas of previous sections can be generalized to yield an efficient algorithm for this case, and we omit the discussion here.
- An "all or nothing" scheme for charging value: if a line segment is totally enclosed its value is obtained, but if the line segment is only partially enclosed we do not count its value towards the value of the fence.

In this section we describe an $O(n^6)$ algorithm to find an optimal fence among n line segments under the "all or nothing" charging scheme.

We state some facts about an optimal fence in the "all or nothing" case:

- The optimal fence will be convex.
- The optimal fence will be a simple polygon with vertices at the endpoints of line segments.

- The optimal fence will cross a line segment zero, one, or two times.

The difficulty of the problem stems from the line segments that are crossed twice.

Our algorithm is a shortest path approach similar to the one we used in Section 5 for points. For convenience we will think of the fence as being directed and going clockwise. We build a graph whose vertices correspond to the endpoints of line segments and connect them with arcs (directed edges) whose cost will be the Euclidean length of the edge (times c) plus the value of line segments this arc "leaves out", partially or completely. Then, as before, we will find a shortest cycle in this graph (using for example, the Floyd-Warshall algorithm [PS 82]). The "tricky" part of this approach is which line segments do we charge to which edges. Ideally, we could find a charging scheme that would ensure that each line segment not completely enclosed by a fence will be charged to exactly one arc of the fence. Unfortunately, there doesn't seem to be an easy way of achieving this. Thus our algorithm will have two phases. In the first phase we will assign a cost to each arc and show that our charging scheme is correct unless the fence crosses a line segment twice, in which case it will get charged to exactly two arcs. In the second phase we separately treat line segments that are crossed twice, to "fix" our charging scheme by updating the costs of some arcs.

An end point i, of a line segment, gives rise to two nodes in our graph t_i and b_i corresponding to whether the node is used on the top part of the fence (the part of the fence going from its left most point to its right most) or the bottom part of the fence. Now we give the costs on arcs on the top part of the fence, say between end points i and j (where i is left of j) which will be the cost on arc (t_i, t_j). The cost on arc (b_j, b_i) is defined similarly. We will assume that no two endpoints have the same x coordinate. We will also treat all the line segments as being directed from left to right.

Consider the semi infinite trapezoid generated by points i, j: a line segment joining them and vertical upward rays from i and j, to infinity. We call this trapezoid *the upper slab of i and j*. A line segment is charged to the arc (t_i, t_j) if:

1. Either, the left end point of the segment is in the upper slab of i and j.

2. or, the line segment crosses a line segment joining point i and j "from the inside to the outside".

In Fig. 19, we show an arc (t_i, t_j) and 12 possible line segments which a convex fence using the arc (t_i, t_j) would not completely enclose. (Not all these segments are present simultaneously, because there are no intersections among line segments in S. We draw them here to illustrate all possible kinds of segments.)

The line segments that will be charged to the arc (t_i, t_j) are: 1, 3, 5, 6, 7, 8, 9, 11. Segments 2 and 4 are not charged because they do not cross the arc (t_i, t_j) and their left endpoint is not in the upper slab of i, j. Segment 10 and 12 are not charged because their left end point is not in the upper slab of i, j, and although they cross the arc (t_i, t_j) they do so from the outside to the inside.

We conclude our description by assigning costs to the arcs (t_i, b_i) and (b_i, t_i). To arc (t_i, b_i) ((b_i, t_i)) we charge all line segments whose left endpoint is to the right (left) of endpoint i. The intuition is that if we use the arc (t_i, b_i) ((b_i, t_i)) then endpoint i is the right most (left most) point of the fence. We are now ready to prove the following lemma.

Lemma 7 *For an optimal fence (in fact for any convex fence) every segment that is not completely enclosed by the fence will be charged to one, or two arcs of the fence. The only segments that are charged to two arcs are crossed by the fence twice. No segment that is completely enclosed by the fence is charged to an arc. In other words our charging scheme is correct except for segments that are double crossed, and these are double charged.*

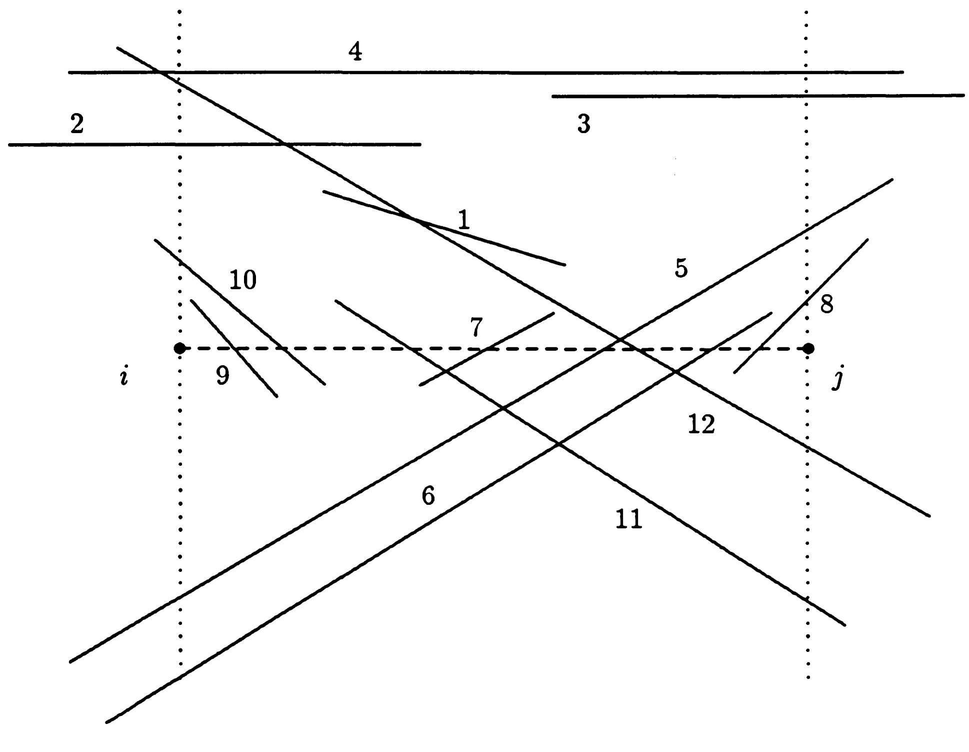


Figure 19: Example to show how different line segments may be oriented with respect to a single edge (i, j)

Proof: Consider a segment that is completely enclosed by the fence. Since neither rule (1) nor rule (2) for charging ever applies, it will not be charged to any arc. Now consider a segment that is not completely enclosed by the fence and the left end point of this segment. If the left endpoint is right of the right most fence point (left of the left most end point) it will be charged to an arc (t_i, b_i) $((b_i, t_i))$. Otherwise, there is an arc for which this segment is of type 1, 3, 7, 8, 9, or 11 and will get charged to that arc. It remains to be shown that if a segment is charged to two arcs then it is crossed by the fence twice. Consider one of the arcs that the segment was charged to, say arc (t_i, t_j). The only types of segments that might be charged again are 5, 6, and 11 by the convexity of the fence. But these three types cross the fence once between end points i and j, and if they are ever charged again there must be another crossing. For types 5 and 6, they may be charged again by rule (1) but that means there must have been an other crossing for the left end point to be above an arc. For type 11 if it is ever charged again it will be by rule (2) which again would imply an additional crossing.

□

We now turn to "fixing" the charging scheme for segments that are crossed twice. Note that if a segment is crossed twice by an optimal fence, there are four endpoints of segments that are "responsible" for these crossings. (See Fig. 20 below.)

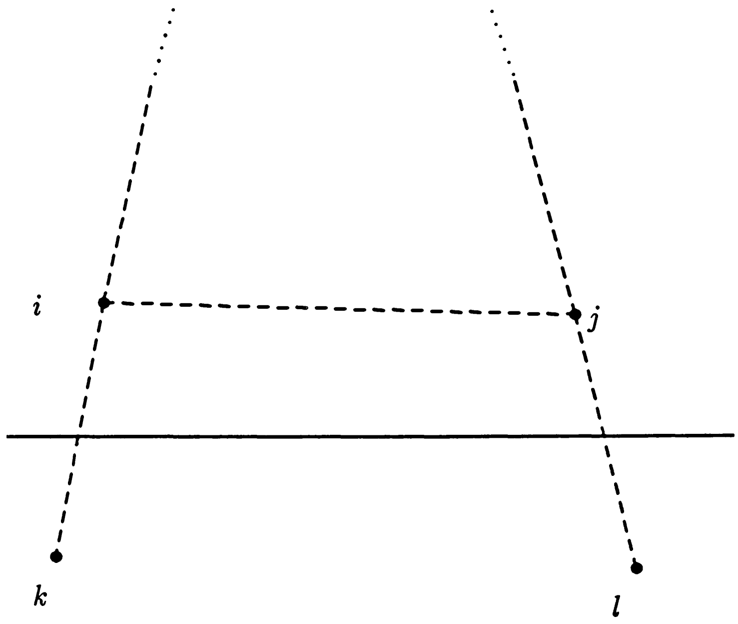


Figure 20: Configuration when a line segment is crossed twice

The idea is to find a shortest path in the graph between t_k and t_l whose cost is correct. We do so, by assuming that somehow we already know the true costs on all arcs that we will need for this computation. We compute the shortest path between t_i and t_j, and set the cost on arc (t_k, t_l) to be the minimum of its current cost and the cost of (t_i, t_j) plus the Euclidean lengths of (k, i) and (j, l) plus the value of segments that are crossed by either (i, k) or (j, l) or both (unless this segment is already counted for one of the arcs in the path from i to j). Thus our procedure will be recursive, correcting costs on arcs, where we always keep the best cost we found so far for each arc.

In order to "somehow already know the true costs on all arcs that we will need" we must do these computations in a particular order. For example, if the best fence from i to j crosses some segment twice we should already have the correct cost on the arc (t_i, t_j) otherwise we wouldn't get the correct cost for (t_k, t_l). What is this particular order? It should be clear that if we count possible crossings and do the corrections in order of increasing possible crossings we achieve such an order. It is at this point that we assume that the line segments are non intersecting and can be ordered in a "nested" fashion (the line segments that cross the boundary twice). Our algorithm thus starts by finding all possible crossings. These are found by connecting endpoints of segments by an arc and checks whether this arc crosses any (and which) segments. Then for each quadruple of points of the form i, k, j, l such as in Fig. 20 we compute the number of possible crossings in the semi infinite wedge given by the ray from i in direction k, i, the ray from j in direction l, j and the arc (i, j). See the shaded region in Fig. 20. With a little care this step can be done in time $O(n^6)$.

To summarize we state the algorithm and compute its complexity below:

Step 1: For every quadruple of points i, k, j, l compute the number of possible crossings in the wedge. Use these numbers to determine an order of the quadruples in which the corrections will be made.

Step 2: Compute original costs on all arcs.

Step 3: Compute corrected costs in the order determined in step 1.

Step 4: Use correct costs to compute shortest cycle which is the optimal fence.

The bottlenecks of the algorithms are steps one and three. As mentioned it is easy to see that Step 1 takes time $O(n^6)$. Step 3 can be done by a single run of Dijkstra's algorithm for each quadruple thus it also take time $O(n^6)$, which dominates the time required to accomplish steps 2 and 4. Thus we have shown the following theorem:

Theorem 8 *An optimal fence can be found in time $O(n^6)$ for the case of n non intersecting line segments which can be crossed by a fence, with an all or nothing value of enclosed segments.*

Corollary 1 *Given a set S of n disjoint line segments in the plane, one can find a convex set enclosing at least k segments that has minimum perimeter (area) in time $O(n^6)$ time.*

9 Multiple fences

In this section we consider the version of the problem in which multiple fences are permitted in the enclosure. Suppose we are allowed to use up to K fences to build the enclosure. It is easy to see that we wil use all K (assuming $n \geq K$). The optimal solution will consist of convex fences, all of which can be separated from each other by straight lines passing through two of the points in set S. Fences F_i and F_j are separated from each other by line l_{ij}. In fact all these separator lines come from the $O(n^2)$ lines defining the arrangement of the lines passing through a pair of points in S.

Thus the entire set of separators has size $K(K-1)/2$. We can choose $K-1$ lines from the lines defining the arrangement. These lines are our "guess" for the actual separators for fence F_1, from F_2 through F_k. These lines define a set of half planes that we intersect to obtain a single region that contains one fence of the enclosure. Now we find a single optimal fence for this point set, define this to be the fence F_1. We discard all the points in F_1 and continue the same procedure to successively find the other $K-1$ fences. Clearly, this algorithm runs in polynomial time for fixed K. In fact, a naive implementation has running time $O(n^{K^2-K+3})$.

A faster algorithm can be obtained using the fact that the fences can be separated by a planar graph. This is due to a lemma in [ERS 87] (see also [RW 89]) that states:

Lemma 8 *Let $F_1,\ldots,F_k$ be k convex, compact and pairwise disjoint sets in the plane. Then there is a planar graph $G(V,E)$ that has vertices corresponding to the K given sets, with the following properties:*

- *For each edge $(F_i,F_j) \in E$, there is a line which cuts the plane into two open half-planes H_{ij} and H_{ji}, such that F_i is contained in H_{ij}, and F_j is contained in H_{ji}.*
- *For each F_i:*
$$F_i \subseteq F_i' := \bigcap_{F_i,F_j \in E} H_{ij}.$$
- *The regions F_i' are disjoint.*

As in the naive algorithm, we can choose these lines from the set of $O(n^2)$ lines defining the arrangement of lines passing through a pair of points in S. Let the number of (non-isomorphic) planar graphs with K vertices be $f(K)$. Note that $f(K)$ is not a function of n. Thus we need only consider all planar graphs on K vertices. For each graph we generate possible edges from the set of $O(n^2)$ lines and compute the optimal fence within each region F_i'. This algorithm has complexity $O(f(K)n^3n^{ck})$, or $O(n^{ck+3})$ for constant K. In [RW 89] it is claimed that one can speed up the algorithm further, to $O(n^{c\sqrt{k}})$, by the use of the planar separator theorem.

10 Conclusions

We presented an $O(n^3)$ time sequential algorithm to obtain an optimal solution to solve the fence enclosure problem. We leave open the question of obtaining an algorithm that takes $o(n^3)$ time.

Another very interesting question is to study the complexity of the enclosure problem in higher dimensions.

As already noted, when the points have positive and negative weights it can be show that the problem becomes NP-hard by a reduction from the *Euclidean Travelling Salesman Problem*. We believe, however, that the following problem may have a polynomial-time algorithm: Given a set S of points with positive and negative values, find a maximum-value convex k-gon, where the value of the k-gon is defined to be the sum of the weights of the enclosed points of S.

Acknowledgements: We would like to thank Robin Roundy for providing the key idea for the NP-hardness proof for the case when the fence has bounded length.

References

[BDDG 85] J.E. Boyce, D.P. Dobkin, R.L. Drysdale and L.J. Guibas, "Finding extremal polygons", *SIAM Journal on Computing* 14 (1985), pp. 134–147.

[CY 88] J.S. Chang and C.K. Yap, "A polynomial solution for the potato peeling problem", *Discrete and Computational Geometry* 1 (1988), pp. 155–182.

[Ch 83] B. Chazelle, "The polygon containment problem", *Advances in Computing Research*, Jai Press (1983), pp.1–33.

[ERS 87] H. Edelsbrunner, A.D. Robison and X. Shen, "Covering convex sets with non-overlapping polygons", *Technical Report UIUCDCS-R-87-1364*, Dept. of Computer Science, University of Illinois at Urbana-Champaign.

[GJ 78] M.R. Garey and D.S. Johnson, "Computers and Intractability: A guide to the theory of NP-completeness", *Freeman, San Francisco.*

[KL 85] B. Korte and L. Lovasz, "Polyhedral results on antimatroids", *to appear* in Proc. of the NY Combinatorics Conference.

[LC 85] D.T. Lee and Y.T. Ching, "The power of geometric duality revisited", *Information Processing Letters* 21 (1985), pp. 117–122.

[ORW 89] M.H. Overmars, G. Rote and G. Woeginger, "Finding minimum area k-gons", *Technical Report RUU-CS-89-7*, University of Utrecht, Utrecht.

[PS 82] C.H. Papadimitriou and K.Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", *Prentice-Hall, New Jersey.*

[RW 89] G. Rote and G. Woeginger, "Geometric clusterings", *Technical Report B-89-04*, Frie University, Berlin.