1978

Geometric Problems with Application to Hashing

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Michael J. O'Donnell

Report Number:
79-303

GEOMETRIC PROBLEMS WITH APPLICATION TO HASHING

Douglas Comer
Michael J. O'Donnell

Computer Science Department
Purdue University
West Lafayette, IN 47907

CSD - TR 303

April 1979

1. Introduction:

Recently, researchers have found efficient algorithms for several geometric problems [GRAH72, SHAM75, SHAM75a, JARV73, PREP77]. These include convex hull in 2 and 3 dimensions as well as nearest neighbor problems. This paper presents algorithms for two new geometric problems. Both problems have hashing applications; generalizations of one apply to graphics.

The first problem is motivated by Sprugnoli [SPRU77], who studies perfect hashing functions for a static set of keys, proposing solutions which seem to require large amounts of space. No analysis of the space requirements is given, however, so a quantitative assessment is not available. Sprugnoli's work suggests the following problem: given a fixed set of keys S and two functions H_1, H_2 which map S to Z^+, find constants c_1, c_2, and c_3 such that the hash function $H(x) = \lfloor c_1 \cdot H_1(x) + c_2 \cdot H_2(x) + c_3 \rfloor$ produces a minimum table size perfect hashing of S.

A related, practically motivated problem raised in [COME79] also concerns finding an optimum linear combination of two hashing functions. In this problem the hashing function need not be perfect, however, we allow only k buckets in the hash table (fixed k) and minimize cost based on the bucket sizes. Typical cost functions might measure the maximum bucket size, the number of empty buckets, or the uniformity of distribution.

Section 2 defines the geometric problems underlying the above hashing problems, section 3 and 4 present the basic algorithms and their analysis, and section 5 discusses the applications in greater detail. Finally section 6 concludes with a summary of open problems.

2. Definitions:

Def: Let $S = \{(x_i,y_i) | x_i,y_i \in R,\ 1 \leq i \leq n\}$, and let $\theta \in [o,\Pi)$ be called an angle of projection. The projection of S at angle θ is

$$P_\theta^S = \{\ x \cdot \cos\theta + y \cdot \sin\theta \mid (x,y) \in S\ \}$$

where S is understood, we write P_θ^S as P_θ. The span of projection P_θ is

$$\text{span}\ (P_\theta) = \max_{u,v \in P_\theta} (|u-v|)$$

the resolution of projection P_θ is

$$\text{res}\ (P_\theta) = \min_{u,v \in P_\theta} (|u-v|)$$

and the length of projection P_θ is

$$\text{len}\ (P_\theta) = \text{span}(P_\theta)/\text{res}(P_\theta)$$ □

Intuitively, we think of S as a set of n points in 2-dimensional space projected onto a line at angle θ. The resolution of a projection gives the minimum distance between projected points along the line; the length gives the distance between endpoints after the resolution has been normalized to unity.

Problem 1: Given S, a finite subset of R x R, find $\theta \in [0,\Pi)$ which minimizes

$$\text{len}(P_\theta)$$ □

In the second problem we think of a finite set of points in 2-dimensional space projected onto a line. Using the minimum and maximum projected points to determine a line segment, mark off k equal size buckets 0,1, ... k-1, such that bucket 0 starts with the minimum projected value and bucket k-1 ends within e past the maximum projected value, where $0<e<<1$. Some number of projected points lie within each bucket; this number is denoted by "size" in the definition. The problem, then, is to find an angle of projection which minimizes the cost of

the resulting distribution according to the cost function C. For example, a typical cost function might be the maximum number of objects in a bucket or the number of nonzero buckets. In any case, T(k) denotes the complexity of computing the cost function given the k bucket sizes. Usually, T(k) is small.

Def: Let S be a finite subset of R x R, let $\Theta \in [0,\Pi)$ be an angle of projection, let $k \in Z^+$, and let $e \in R$ where $0<e<<1$. Using $\min(P_\Theta)$ and $\max(P_\Theta)$ to denote the minimum and maximum elements in P_Θ^S, the scale of P_Θ with k buckets is

$$\text{scale}(P_\Theta,k) = (\max(P_\Theta) + e - \min(P_\Theta))/k$$

The size of bucket i under projection P_Θ scaled to k is $\text{size}(P_\Theta,k,i) = |\{s \mid s \in P_\Theta \text{ and } s \in [\min(P_\Theta) + \text{scale}(P_\Theta,k)\, i,\ \min(P_\Theta) + \text{scale}(P_\Theta,k)\cdot(i+1))\}|$

Note that $\text{size}(P_\Theta,k,i)>0$ only if $0 \le i < k$. The distribution of projection P_Θ into k buckets is

$$\text{distr}(P_\Theta,k) = \{\text{size}(P_\Theta,k,i) \mid 0 \le i < k\}$$

Let $C:Z^k \to Z^+$ be a cost function. Then the cost of a distribution is

$$\text{cost}(C,P_\Theta,k) = C(\text{distr}(P_\Theta,k))$$

By convention, T(k) will denote the time complexity of computing $C(\text{distr}(P_\Theta,k))$ given that $\text{distr}(P_\Theta,k)$ has been computed. □

Problem 2: Given S a finite subset of R x T, $k \in Z^+$, $c \in R$ where $0<e<<1$, and a cost function $C:Z^k \to Z^+$, find $\Theta \in [0,\Pi)$ which minimizes $\text{cost}(C,P_\Theta,k)$.

□

3. Finding Minimum Length Projections:

This section presents an efficient algorithm for problem 1. First we give an overview of the algorithm and data structures. Then, we discuss each piece in more detail. Finally, we show a simple lemma needed for correctness, and conclude with a discussion of the algorithms complexity.

Basically, our algorithm forms two lists, SPANLIST and RESLIST, corresponding to $span(P_\Theta)$ and $res(P_\Theta)$. Elements in these lists, ordered by increasing angle Θ, consist of an angular interval $[\alpha,\beta)$[1] and a pair s_1, $s_2 \in S$, with the interpretation that for $\Theta \in [\alpha,\beta)$ the projections of s_1 and s_2 determine $span(P_\Theta)$ and $res(P_\Theta)$, respectively. From SPANLIST and RESLIST the algorithm forms a single list, LENLIST, which is again ordered by angle. LENLIST contains enough information to compute $span(P_\Theta)$ and $res(P_\Theta)$ for each angle $\Theta \in [0,\Pi)$, from which $len(P_\Theta)$ can be determined.

Throughout the development we note the time complexity of each step, explaining the analysis later, and summarizing at the end.

Algorithm 1:

input: S, a finite subset of R x R
output: $\Theta \in [0,\Pi)$ and $len(P_\Theta)$ such that $len(P_\Theta)$ is minimum
method:

1.	compute SPANLIST	$O(n\log n)$
2.	compute RESLIST	$O(n^2\log n)$
3.	merge SPANLIST and RESLIST to form LENLIST	$O(n^2)$
4.	find an element of LENLIST for which $len(P_\Theta)$ is minimum and output it.	$O(n^2)$

SPANLIST can be formed by first extracting a convex hull and ordering the points of the hull counterclockwise, with respect to an interior point. Starting from an arbitrary point, search for another point of maximum separation to

---

[1] the interval $[\alpha,\beta)$ is taken clockwise from α to β. Since $len(P_\Theta) = len(P_{\Theta+\Pi})$ we consider angles in $[0,\Pi)$ instead of the usual $[0,2\Pi)$, and all angle arithmetic is performed mod Π.

get two points which appear in SPANLIST. From these first two points walk the hull counterclockwise to determine the angle over which each pair of points dominates. Finding the hull and ordering it requires O(nlogn) time [GRAH72]; walking takes O(n) time and produces a SPANLIST of O(n) entries.

Finding RESLIST is a bit more tricky and requires some explanation. The key to understanding the algorithm lies in the following observation. Let L be the set of all possible unordered pairs of points from S, and consider a particular pair $(s_1,s_2) \in L$. For some angle $\Theta \in [0,\Pi)$, s_1 and s_2 project to the same point as shown in Figure 1. We call the angle for which s_1 and s_2 project to the same point a zero angle for the pair.

If one thinks of the distance between projections of two points s_1 and s_2 as the angle of projection increases from 0 to Π as shown in Figure 2, we see that it is essentially a reflected sine wave of period Π and amplitude equal to the distance from s_1 to s_2. When another pair of points $(\bar{s}_1, \bar{s}_2)$ with larger separation is added, the distance of their projection forms a reflected sine wave with period Π, greater amplitude, and different phase. The distance between projections of $\bar{s}_1$ and $\bar{s}_2$ will be less than the distance between projector's of s_1 and s_2 only around the zero angle of $(\bar{s}_1,\bar{s}_2)$. This fact, expressed in Lemma 1, is the basis for computing RESLIST.

LEMMA 1: Let s_1, s_2 be elements of S such that $||s_1-s_2||$ is maximum, and let Θ_z be the zero angle for (s_1,s_2). Then the angles Θ for which $res(P_\Theta^{\{s_1,s_2\}})$ is strictly minimum lie in an open interval (α,β) containing Θ_z, with $\beta - \alpha \leq \Pi/2$.

PROOF:

First, let (s_1,s_2) and (s_3,s_4) be two pairs of points with $d_{12} = ||s_1 - s_2|| \geq ||s_3 - s_4|| = d_{34}$. Let Θ_{12}, Θ_{34} be the angles of the segments $\overline{s_1s_2}$, $\overline{s_3s_4}$ respectively. Then

$$res(P_\theta^{\{s_1,s_2\}}) = |d_{12}\cdot\cos(\theta - \theta_{12})|$$

$$res(P_\theta^{\{s_3\ s_4\}} = |d_{34}\cdot\cos(\theta - \theta_{34})|$$

Since the lengths of the projections of $\overline{s_1s_2}$ and $\overline{s_3s_4}$ depend only on the lengths and directions of these segments, not their positions, we need only consider projections of the vectors $\overrightarrow{s_1s_2}$ and $\overrightarrow{s_3s_4}$ positioned at the origin. The angles for which $res(P_\theta^{\{s_1,s_2\}} = res(P_\theta^{\{s_3,s_4\}})$ are the angles $\theta_e,\theta'_e \in [0,\Pi)$ perpendicular to the vectors $\overrightarrow{s_1s_2} - \overrightarrow{s_3s_4}$ and $\overrightarrow{s_1s_2} - \overrightarrow{s_4s_3}$ (see Figure 3).

Order θ_e,θ'_e so that $res(P_\theta^{\{s_1,s_2\}}) < res(P_\theta^{\{s_3,s_4\}}$ for $\theta \in (\theta_e, \theta'_e)$. Of course, $\theta_z \in (\theta_e, \theta'_e)$. $\theta'_e - \theta_e \leq \Pi/2$ may be proved geometrically from Figure 3. Intuitively, the longer segment $\overline{s_1s_2}$ must produce the longer projection for at least one half of the angles in $[0,\Pi)$.

Now, the region over which $res(P_\theta^{\{s_1,s_2\}})$ is minimum is the intersection of all the intervals $[\theta_e,\theta'_e]$ for all choices of (s_3,s_4). The intersection of open intervals containing θ_2 of length $\leq \Pi/2$ must itself be such an interval □ .

From Lemma 1 we can form a procedure for computing RESLIST. Order the set L of all pairs of points in S by increasing distance between the points in a pair. To initialize, select a minimum distance pair, making its zero angle the origin for measuring angles. Place the pair on RESLIST with interval $[0,\Pi)$.

Then, insert each pair (s_1,s_2) from L into RESLIST by locating the interval(s) which include the zero angle of (s_1,s_2) and updating RESLIST. Observe that each new pair of points adds at most two intervals to RESLIST, and possibly subsumes existing intervals. Thus RESLIST contains at most $O(n^2)$ entries corresponding to n^2 pairs on L. By keeping the entries in the leaves of a balanced tree during insertions, and linking the leaves in a list, one can find an interval including a zero angle in O(log n) time. Then, moving right and left in the list, one can determine how many existing entries to delete. Note that while deletion costs O(log n), each entry will be added and deleted at most once, so

we charge it both the cost of its insertion and deletion. Therefore, the running time is $O(\log n^2)$ per entry.

procedure compute RESLIST

1.	Generate L, a list of all unordered pairs of points From S ordered by increasing distance of separation between points in the pair	$O(n^2 \log n)$
2.	for each pair $(s_1,s_2) \in L$ do {	$O(n^2)$ iterations
3.	find interval in RESLIST containing the zero angle of (s_1,s_2) using a balanced tree	$O(\log n)$/iteration
4.	update RESLIST possibly removing old intervals that are subsumed and updating the balanced tree. }	(see note in text)

Performing the merge of SPANLIST and RESLIST is straightforward and requires at most $O(n^2)$ time (since there is at most n^2 entries in RESLIST and $O(n)$ entries in SPANLIST). The above analysis, combined with Lemma 1 allows us to conclude the correctness and time complexity of Algorithm 1.

Thm 1: Algorithm 1 solves Problem 1 in $O(n^2 \log n)$ time.

Proof: Immediate from the above discussion.

4. Finding Minimum Cost Distribution into Buckets:

This section presents an efficient algorithm for finding an angle of projection which minimizes the cost of a distribution. It relies heavily on the reader's knowledge and intuition from the previous section, concentrating on differences between the two algorithms. As before, we present an overview of the solution first followed by a more detailed discussion of each piece. Also as before, we note the complexity of each section as we present, it justifying the claims later.

Our algorithm for finding a minimum cost distribution begins like Algorithm 1 by computing SPANLIST. Recall that SPANLIST, ordered by angle, contains angular intervals $[\alpha,\beta)$ and pairs of points $s_1,s_2 \in S$ that define $\min(P_\theta)$ and $\max(P_\theta)$ for $\theta \in [\alpha,\beta)$. In terms of the distribution, s_1 is the start of bucket 0 and $s_2 + e$ is the end of bucket k-1 for $\theta \in [\alpha,\beta)$. Assuming the special case of colinear points has been taken care of, the algorithm proceeds as follows:

Algorithm 2 $O(kn^2T(k) + kn^2\log(kn))$

input: S, a finite subset of R x R, an integer $k > 0$, $e \in R$ where $0 < e << 1$, and a cost function $C : Z^k \to Z^+$.

output: an angle of projection $\theta \in [0,\Pi)$ and $cost(C,P_\theta,k)$ such that $cost(C,P_\theta,k)$ is minimum.

method:

1.	form SPANLIST	O(n log n)
2.	for each $([\alpha,\beta),s_1,s_2) \in$ SPANLIST do {	O(n) iterations
3.	mincost ← $cost(C,P_\alpha,k)$	O(T(k))/iteration
4.	find all distributions in $[\alpha,\beta)$.	O(kn log(kn))/iteration
5.	for each distribution in $[\alpha,\beta)$ do	O(kn) times/iteration
6.	mincost ← $\min(\text{mincost},cost(C,P_\theta,k))$	O(T(k))kn times/iteration
	}	
7.	output mincost and angle giving that cost	O(1)

Steps 4-6 each require further explanation; we begin with step 4.

The key to finding all distributions in an interval $[\alpha,\beta)$ lies in thinking of a line of projection with k buckets marked off rotating from α to β as shown in Figure 4. One can easily construct such a line at angle α by projecting all points and marking off k equally spaced intervals between the smallest and largest.

Think of k interval marks as the projection of a set of k equally spaced "dummy" points along a line from s_1 to s_2. As the imaginary line of projection rotates from α to β, the distribution changes whenever the projection of an element from S crosses a projection of one of the k dummy points (i.e. a bucket mark). Thus, if D denotes the set of k dummy points, a crossing corresponds to a zero angle of a pair from D x S. The algorithm simply forms a list, CROSSLIST, consisting of kn triples (θ,d_i,s_i) where θ is the zero angle between $d_i \in D$ and $s_i \in S$. Of course, only those triples with $\theta \in [\alpha,\beta)$ are saved. Sorting CROSSLIST requires $O(n \cdot k \cdot \log(nk))$ time, and produces a list of all in $[\alpha,\beta)$ where the distribution changes, as well as a record of which point changes buckets at that angle.

We summarize the procedure for step 4:

procedure step 4 find all distribution in $[\alpha,\beta)$

1.	Form D, a set of equally spaced "dummy" points on the line s_1 to s_2 where s_1 and s_2 determine $\min(P_\theta)$ and $\max(P_\theta)$ for $\theta \in [\alpha,\beta)$	O(k)
2.	Find size (P_α,k,i) for $0 \leq i < k$	O(n)
3.	Form CROSSLIST by finding all zero angles for pairs in D x S and sorting	O(nk log(nk))

Given CROSSLIST steps 5-6 become simple: remove the next element (θ,d_i,s_i) from CROSSLIST, update the appropriate bucket counts, compute the cost of the new distribution, and record it in case the new cost is lower than the minimum

found so far. The $O(n \cdot k)$ elements on CROSSLIST each require $O(T(k)$ time to process yielding a bound of $O(n \cdot k \cdot T(k))$. The loop in step 2 iterates steps 4-6 for each of the $O(n)$ items in SPANLIST, however, so the total processing in steps 5-6 requires $O(n^2 \cdot k \cdot T(k))$. Similarly, step 4 requires a total of $O(n^2 k \log(nk))$ because it is iterated $O(n)$ times. Thus, the total running time of Algorithm 2 is $O(n^2 k \log(nk) + n^2 k\, T(k))$

Thrm 2: Algorithm 2 solves Problem 2 in $O(n^2 k \log(nk) + n^2 k T(k))$ time.

Proof: Immediate from the above discussion.

5. Applications to Hashing:

This section describes how Algorithms 1and 2 apply to the hashing problems mentioned in the introduction. The first problem, concerned with finding minimum table size perfect hashing can be defined as:

Problem H1: Given K a set of n keys and two functions H_1 and H_2 which map k to Z^+, find constants c_1, c_2, and $c_3 \in R$ such that for $H(x) = \lfloor c_1 \cdot H_1(x) + c_2 \cdot H_2(x) + c_3 \rfloor$ the following holds:

(1) $\forall k_1, k_2 \in K$, $H(k_1) = H(k_2)$ iff $k_1 = k_2$

(2) $\min_{k \in k}(H(k)) = 0$

(3) $\max_{k \in k}(H(k))$ is minimized

Property 1 guarantees that H is a perfect hashing, while properties 2 and 3 assure a minimum table size.

Problem H1 translates to the geometric problem of finding an angle of projection such that the length of projection is minimized when projected points are placed in different cells. Note, however, that Algorithm 1 does not always produce such a minimum projection. Instead, it minimizes the length of projection while simultaneously placing the closest pair of projected points distance 1 apart.

To see the difference between the unit distance stipulation imposed by Algorithm 1 and the distinct cell stipulation given in the problem statement, consider three colinear points as shown in Figure 5. Trouble arises when an integer separates two projected points p_1 and p_2. Using real arithmetic, one could squeeze p_1 and p_2 arbitrarily close together, forcing p_3 to move close to p_2. Clearly, the optimum solution requires only 3 cells in a hash table. Algorithm 1 which places p_1 and p_2 unit distance apart, requires more than 3 cells. Even using floating point hardware to approximate the real number solution may still lead to anomolies like the one in Figure 4 if the floating point

approximations for s_1 and s_2 happen to lie on either side of an integer.

We take the view that while Algorithm 1 may not produce an optimum solution for problem H_1, it provides a sufficiently accurate approximation for most applications. First, if one chooses c_3 to translate the minimum projected point to 0, special cases of an integer falling between two very close projections may disappear. Such translation seems reasonable, even desirable, in practice. Second, while different floating point representations may yield different minimum solutions, Algorithm 1 is optimum in the sense that it produces a smallest solution which is valid independent of floating point representation.

We can define the second hashing problem as:

Problem H2: Given K a set of n keys, $m \in Z^+$, $e \in R$ such that $0 < e << 1$, H_1 and H_2 which map K to Z^+, find constants c_1, c_2, and $c_3 \in R$ such that for $H(x) = [c_1 \cdot H_1(x) + c_2 \cdot H_2(x) + c_3]$ the following holds:

(1) $\min_{k \in K} H(k) = 0$

(2) $\max_{k \in K} H(k) = m - 1 - e$

(3) Let C(k) be a cast function s.t. $C: Z^k \rightarrow Z^+$, C(H(k)) is minimum.

Because of its more restrictive definition, Algorithm 2 applies better to H2 than Algorithm 1 applies to H1. In particular, the problem formulated in [COME79] requires finding a projection that distributed keys as uniformly as possible into buckets. In the particular problem described, the cost of searching a bucket with t entries is log t-1. Assuming that the complexity of computing log i for small integer i is constant, this yields a complexity of n $n^2k \log(nk) + nk^2$ for finding an optimum projection.

6. Conclusions and further research:

In order to be used in graphics applications, the algorithms of this paper must be adapted to find optimal projections from 3 dimensions to 2 dimensions, instead of 2 to 1. A major component of the adapted algorithm, a solution to the 3-dimensional convex hull problem, is already known [PREP77]. The major remaining difficulty is to find a higher-dimensional analogue to the search tree representation of RESLIST.

Given two hashing functions h_1 and h_2 and a fixed set of keys, the algorithm of this paper may be used to choose a good hashing function of the form $(n) = a \cdot {}_1(n) + b \cdot {}_2(n) + c$. The question of how good is the best hashing function in such a class is open. Very little is known about hashing with predetermined, static sets of keys; the only treatment of this problem seems to be Sprugnoli's [SPRU77]. A combinatorial analysis should at least settle the question of how large a class of hash functions is needed to guarantee a given level of performance for any set of keys.

[COME79] D. Comer and V.Y. Shen, "Hash-Binary Search. A Fast Technique for Searching an English Spelling Dictionary," Technical Report TR-CSD-304, Dept. of Computer Science, Purdue University, West Lafayette, IN, 1979.

[GRAH72] R.L. Graham, "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set," Infor. Proc. Letters 1, (1972) 132-133.

[JARV73] R. Jarvis, "On the Identification of the Convex Hull of a Finite Set of Points in the Plane," Infor. Proc. Letters 2, (1973) 18-21.

[PREP77] F. Preparata and S. Hong, "Convex Hulls of Finite Sets of Points in Two and Three Dimensions," Comm. ACM 20:2 (Feb.1977), 87-93.

[SHAM75] M. Shamos, "Geometric Complexity," in Proc. 7th Annual ACM Symposium on Theory of Computing (1975) ACM NY, 224-233.

[SHAM75a] M. Shamos and D. Hoey, "Closest Point Problems," Proc 16th Ann IEEE Symp. on Foundations of Computer Science, IEEE, NY, (1975) 151-162.

[SPRU77] R. Sprugnoli, "Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets," Comm. ACM 20:11 (November 1977), 841-850.

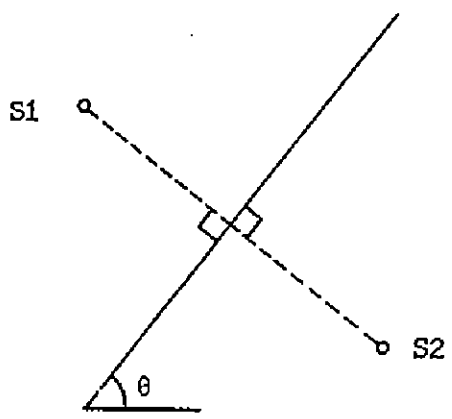
Figure 1. The angle Θ such that the projections of S_1 and S_2 coincide. There is such an angle for each pair of points.

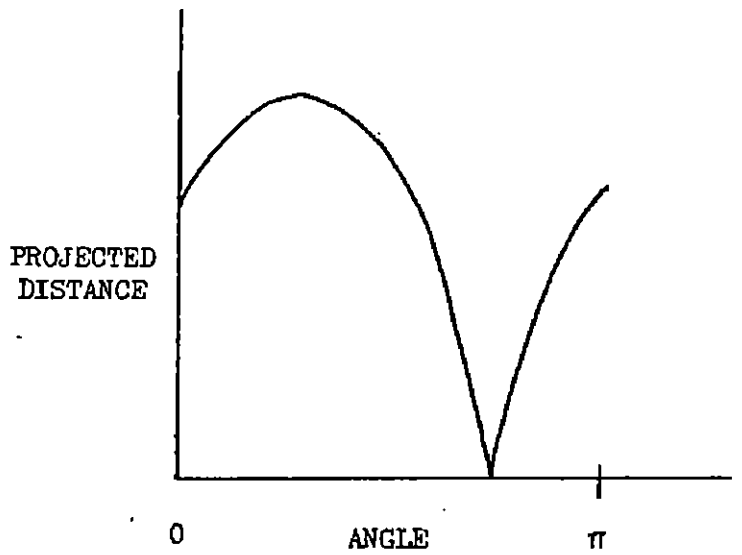
Figure 2. The distance between the projection of two points as a function of the angle of projection. The curve is a reflected sine wave of period π and amplitude $||S_1 - S_2||$.

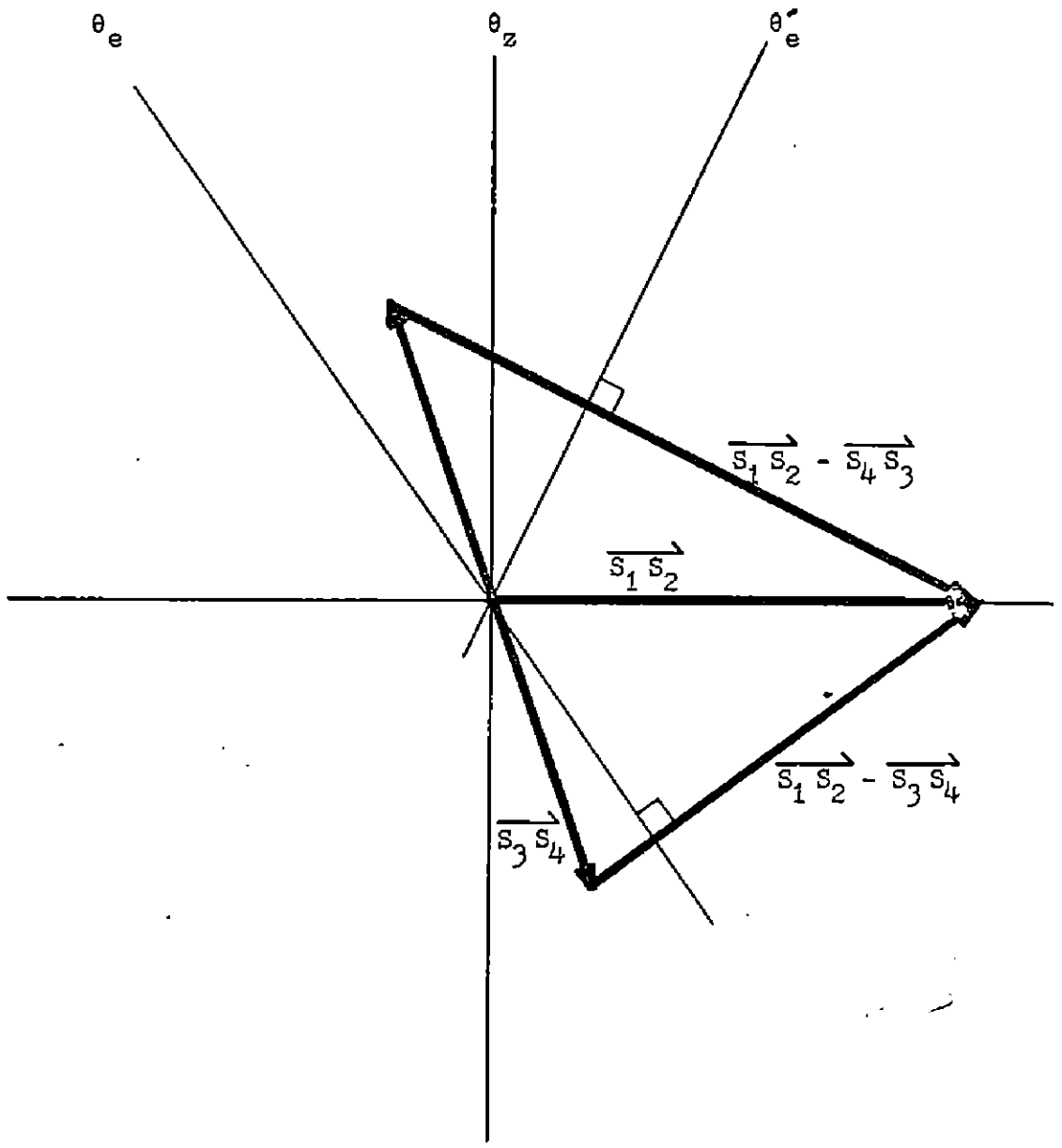
Figure 3. Angles Θ_e and Θ_e' where $\overline{S_1\ S_2}$ and $\overline{S_3\ S_4}$ project to equal lengths. Θ_z is the zero angle for (S_1, S_2)

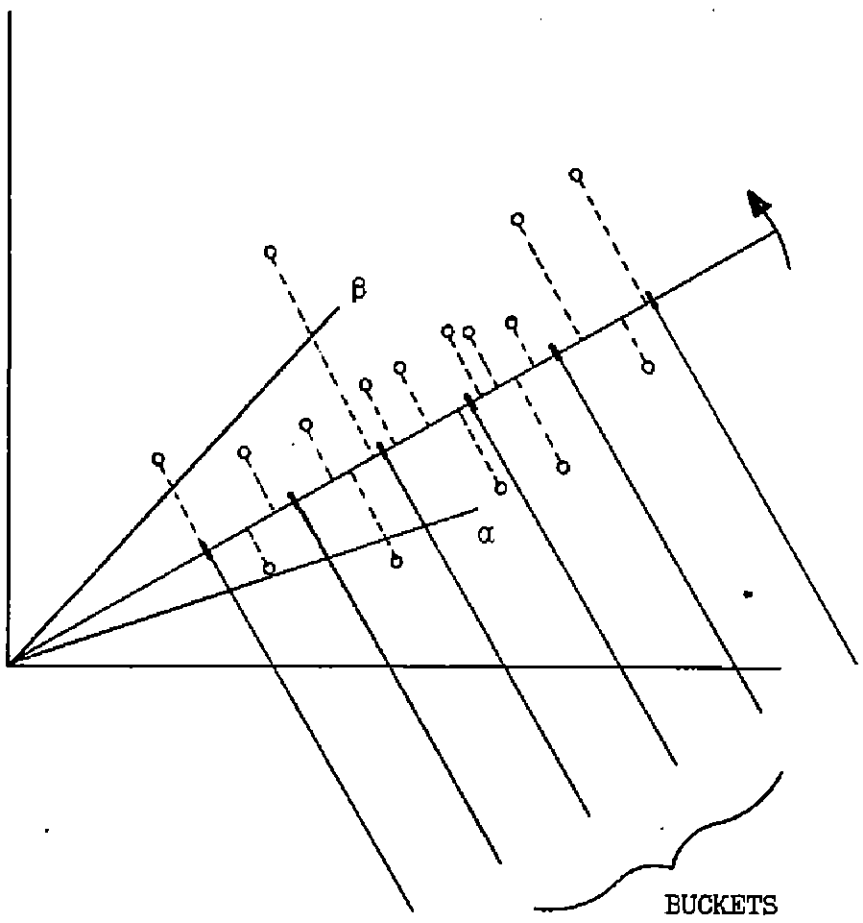
Figure 4. A lire ℓ rotating from angle α to β with k buckets marked off.

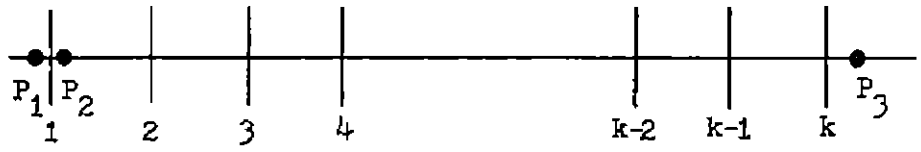
Figure 5. A case where Algorithm 1 can produce a projection that is arbitrarily far from the optimum hash table size. The points P_1, P_2, and P_3 are colinear.


S1
S2
θ


PROJECTED
DISTANCE
0
ANGLE
π


θ_e
θ_z
θ'_e
$\overrightarrow{S_1 S_2} - \overrightarrow{S_4 S_3}$
$\overrightarrow{S_1 S_2}$
$\overrightarrow{S_1 S_2} - \overrightarrow{S_3 S_4}$
$\overrightarrow{S_3 S_4}$


β
α
BUCKETS


P1
P2
1
2
3
4
k-2
k-1
k
P3