DIAMETER PARTITIONING

David Avis

School of Computer Science

McGill University

805 Sherbrooke St. W.

Montreal, Canada, H3A 2K6

ABSTRACT

We discuss the problem of partitioning a set of points into two subsets with certain conditions on the diameters of the subsets and on their cardinalities. For example, we give an $O(n^2 \log n)$ algorithm to find the smallest t such that the set can be split into two equal cardinality subsets each of which has diameter at most t. We also give an algorithm that takes two pairs of points (x, y) and (s, t) and decides whether the set can be partitioned into two subsets with the respective pairs of points as diameters.

Key Words. Computational Geometry, Partitioning, Diameter

1. Introduction

Consider a set of communications posts on a large plane. Each post is equipped with a transmitter that can reach some distance t from the post. If all the posts are within distance t of each other, all can communicate without difficulty. Suppose, however, that some posts are further than t units apart. We would like to know if the posts can be split into two groups, so that within each group each pair of posts may communicate. In order to conserve energy, we wish to reduce the power of the transmitters as much as possible. Accordingly, what is the smallest t for which such a partition into two groups is possible? In addition, we may wish to impose criteria on the cardinality of the subsets: for example, we may wish to make the subsets have equal cardinality.

The above problem is typical of a large class of problems that have been studied recently, that concern the partitioning of point sets. A closely related problem is the m-centers problem. In this case, it is required to cover the point set by a set of m circles of minimum radius[9] [13] [15]. Cardinality criteria have also been much studied, motivated by the generalized range search problem in data retrieval. Various results have been obtained on the problem of splitting a set of points in d dimensions by a set of intersecting hyperplanes, such that each open region contains an equal or near equal fraction of the points[16] [17] [5] [2] [3]. A related problem is that of finding subsets of cardinality k that can be separated from the point set by a hyperplane. Even for $d = 2$ when the hyperplane becomes a line, the problem of finding the number of such subsets appears very difficult[8] [7] [12].

In this paper, we combine cardinality criteria with criteria on the diameter of the subsets, giving the problem mentionned in the first paragraph. Two major cases arise depending on whether or not we require the subsets to be linearly separable. Two subsets S and T of a planar point set P are linearly *separable* if there is a line such that each subset lies in opposite half spaces bounded by the line. We call a partition into S and T *balanced* if $|S| = \lfloor \frac{|P|}{2} \rfloor$. As shown by the example in Figure 1, the minimum diameter balanced partition of a point set may not be separable. In this figure, both circles have unit radius. The only minimum balanced partition is $\{p_1, p_2, p_3, p_4\}$, $\{p_5, p_6, p_7, p_8\}$, and it is not separable. If we restrict ourselves to separable partitions, we can find a minimum diameter balanced partition in polynomial time by brute force. This follows from the fact that any separating line can be rotated until it hits at least one point in S and one point in T. Therefore we can effectively enumerate all separable subsets by choosing all pairs of points in the set and constructing the separating line through each of them. This yields an $O(n^3 \log n)$ algorithm, since it takes $O(n \log n)$ time to find the diameters of the two subsets[14]. For the non-separable case, no obvious brute force approach suggests itself.

The main results of this paper are contained in the next three sections. In Section 2 we present an efficient algorithm for finding the minimum diameter balanced partition in the case where the subsets need not be separable. In Section 3 we consider the additional restriction that the subsets must be separable. Finally we consider a slightly different type of problem in Section 4. Here we are given two candidate pairs of points for the diameters, and the problem is to decide whether the point set can indeed be partitioned with these pairs as the diameters of the subsets. The techniques throughout are related: an appropriate graph is defined and the geometric problem is transformed into an easily solvable graph problem.

2. Minimum Diameter Balanced Partitions

Let $P = \{p_1, \ldots, p_n\}$ be a set of points in Euclidean space. Let $d(p_i, p_j)$ be the distance between points p_i and p_j. In this section we make no restrictions on the distance function. For the application mentionned in the introduction Euclidean distance is assumed. The diameter of the set is denoted $diam(P)$ and is defined by

$$diam(P) = \max_{1 \le i < j \le n} d(p_i, p_j).$$

For any real number t we consider the following predicate $Q_0(t)$:

P can be partitioned into subsets S and T such that max $(diam(S), diam(T)) \le t$.

Clearly $Q_0(t)$ is false if there exist three points in P which are all more than t units apart. This suggests defining an auxiliary graph $G_t = (P, E_t)$ on the point set P, where (p_i, p_j) is an edge in E_t if and only if $d(p_i, p_j) > t$. Such a construction is shown in Figure 2. The following easy lemma gives a graph theoretic answer to $Q_0(t)$.

Lemma 2.1. *$Q_0(t)$ is true if and only if G_t is bipartite.*

Proof: Suppose $Q_0(t)$ is true, and let S and T be the corresponding subsets. Since $\max(diam(S), diam(T)) \le t$, S and T are independent sets in G_t, that is, no edge has both its endpoints in the same subset. Therefore G_t is bipartite. Conversely, if G_t is bipartite, let S and T be the two parts. Since they are independent, $Q_0(t)$ holds.

We are interested in finding the smallest value of t such that $Q_0(t)$ is true. Clearly if $Q_0(t)$ is true, then so is $Q_0(t')$ for any $t' \geq t$. Furthermore, the minimum value of t for which $Q_0(t)$ is true is a distance $d(p_i, p_j)$ for some i and j. (The diameter of the larger subset give such a pair, for example.) This suggests the following algorithm, which will later be extended to solve several problems. The algorithm employs a binary search on the sorted set of distances between the n points.

Algorithm $MIN - DIAM$ (P)

1. Sort the distances $d(p_i, p_j)$ into increasing order: $d_1 \leq d_2 \leq \cdots \leq d_{\binom{n}{2}}$.

2. Set $u = n$; $l := 1$;

   while $u \geq l$ **do**

   begin

   Set $m = \lfloor \frac{u+l}{2} \rfloor$; $t = d_m$;

   Construct G_t;

   if $TEST - GRAPH - 1$ (G_t) **then** $u = m - 1$ **else** $l = m + 1$

   end;

   if not $TEST - GRAPH - 1$ (G_t) **then** $t = d_{m+1}$;

   return (t).

Algorithm $MIN - DIAM$ is a straight forward binary search to find the minimum diameter partition. The procedure $TEST - GRAPH - 1$ is a boolean procedure that returns true if the graph G_t is bipartite. Such a procedure is easily implemented by depth first search[1] in time proportional to the number of edges in the graph. Step 1 of $MIN - DIAM$ takes $O(n^2 \log n)$ time. The while loop in step 2 is performed at most $O(log\ n)$ times. Each call to $TEST - GRAPH - 1$ takes at most $O(n^2)$ time. Therefore the overall complexity of $MIN - DIAM$ is $O(n^2 \log n)$ time.

The subsets produced by algorithm $MIN - DIAM$ will not in general be balanced. For real t and integer k we define the predicate $Q(t, k)$:

P can be split into sets S and T such that (i) $\max(diam(S), diam(T)) \le t$ and (ii) $|S| = k$.

Of particular interest is the case $k = \lfloor \frac{n}{2} \rfloor$. Along the lines of Lemma 2.1 we have the following lemma.

Lemma 2.2. *$Q(t, k)$ is true if and only if G_t has a bipartition (S, T) with $|S| = k$.*

If G_t is connected then there is a unique bipartition, but if it is disconnected, there may be exponentially many different bipartitions. Suppose that G_t has m connected components. Each of these components is bipartite and we will denote each by the bipartition (S_i, T_i), $i = 1, \ldots, m$. Note that $S_i = \phi$ or $T_i = \phi$ is possible. Figure 2 illustrates the case $m = 2$. Here $S_1 = \{p_1\}$, $T_1 = \{p_4, p_5\}$ and $S_2 = \{p_6\}$, $T_2 = \{p_2, p_3\}$. It can be seen that $Q(t, k)$ is true if there are binary values $x_1, \ldots, x_m$ so that

$$\sum_{i=1}^{m} (x_i\ |S_i| + (1 - x_i)\ |T_i|) = k. \tag{1}$$

In general, it is an NP-complete problem to solve equations of type (1)[10]. However, in our case, all of the data are bounded by n and so (1) can be solved in $O(n^2)$ time by a so called pseudo-polynomial time algorithm. We outline the procedure below in $TEST - GRAPH - 2\ (G_t, k)$. The idea of this algorithm is as follows. We consider which sums of form (1) are possible using components $(S_1, T_1), (S_2, T_2), \ldots, (S_i, T_i)$, for each $i = 1, 2, \ldots, m$.This is done sequentially. For $i = 1$ the only sums possible are for $k = s_1$ or $k = t_1$. If a sum of value j can be obtained using the first i components, then sums of value $j + s_{i+1}$ and $j + t_{i+1}$ are obtainable using the first $i + 1$ components. The sums are recorded using the array w. After the *ith* iteration, $w[j] = 1$ if and only if the sum j can be obtained by a partition of the first i components. Clearly we do not need to record $w[j]$ for j greater than k. When updating the array w, we update in decreasing order by subscript so as not to destroy information about the previous iteration until it is no longer of value. The array *flag* is used to record the composition of the partial sums. This is to allow the construction of the actual partition if the algorithm terminates successfully. Assume at iteration $i-1$ a partial sum of value j can be constructed. Then at iteration i we can construct sums of value $j_1 = j + s_i$ and $j_2 = j + t_i$. We record how these new sums are constructed by setting respectively $flag[i,\ j_1] = "S"$ and $flag[i,\ j_2] = "T"$. Later, we describe how *flag* is used to construct the partition.

Algorithm $TEST - GRAPH - 2\ (G_t,\ k)$

1. Find connected components (S_i, T_i), $i = 1, \ldots, m$.

   set $s_i = |S_i|, \quad t_i = |T_i|, i = 1, \ldots, m$.

   set $w[i] = 0 \quad i = 0, \ldots, k$;

   set $w[s_1] = 1,\ w[t_1] = 1,\ flag[1,\ s_1] = "S",\ flag[1,\ t_1] = "T"$;

2. **for** $i = 2$ **to** m **do**

   for $j = k$ **to** 0 **do**

   if $w[j] = 1$ **then**

   begin

   $w[j] = 0$;

   $j_1 = j + s_i$;

   $j_2 = j + t_i$;

   if $j_1 \le k$ **then** $w[j_1] = 1,\ flag[i,\ j_1] = "S"$;

   if $j_2 \le k$ **then** $w[j_2] = 1,\ flag[i,\ j_2] = "T"$;

   end;

3. **if** $w[k] = 1$ **then return (true) else return (false).**

The correctness of algorithm $TEST - GRAPH - 2\ (G_t,\ k)$ can be verified by induction on m, the number of connected components. Our hypothesis is that for $0 \le j \le k$, $w[j] = 1$ if and only if there exists a bipartition (S, T) of G_t with $|S| = j$. This is clearly true for $m = 1$. Assume that the hypothesis is true for arbitrary m. Suppose G_t has $m + 1$ components. By hypothesis after the algorithm has executed the main for loop with $i = m$ the array w tells us exactly which bipartitions are possible using the first m components of G_t. Then there is a bipartition of G_t with $|S| = j$ if and only if at this stage in the computation $w[j - s_{m+1}] = 1$ or $w[j - t_{m+1}] = 1$. Since the array w is processed in decreasing order by subscript, this update can be done in place with no additional storage. This completes the induction.

We now show how to construct a partition which obtains the value k whenever such a partition exists. If $flag[m, k] =$ "S" then the last subset of vertices used in constructing the partition was S_m, otherwise it was T_m. In the former case, the penultimate subset of vertices used can be found by examining $flag[m - 1, k - s_m]$. In the latter case, we examine $flag[m - 1, k - t_m]$. Repeating in this way, we can trace the entire partition in m steps.

The modified algorithm $TEST - GRAPH - 2$ still runs in $O(n^2)$ time. Thus we are able to test the predicate $Q(t, k)$ in time $O(n^2)$ time. Using it in procedure $MIN - DIAM$ in place of $TEST - GRAPH - 1$ therefore gives an $O(n^2 \log n)$ algorithm for finding the minimum diameter balanced partition of P. This proves theorem 2.3.

Theorem 2.3. *A minimum diameter balanced partition of a planar point set can be found in time* $O(n^2 \log n)$.

3. Minimum Diameter Separable Partitions

In this section we consider the problem of partitioning a point set P into two linearly separable point sets S and T with minimum diameter. We allow points of S and T to lie on the separating line, which we will denote l. For any real number t consider the following predicate $Q_1(t)$:

> P can be partitioned into two separable subsets S and T such that $\max\,(diam(S), diam(T)) \le t$.

We proceed as in Section 2 by defining the auxiliary graph $G_t = (P, E_t)$. We also require the following definition. Let E be a set of line segments in the plane. A line l is a *stabbing line* for E if l intersects every line segment in E. Figure 3 illustrates the construction of G_t and a stabbing line for the point set of Figure 1. Observe that if E_t has a stabbing line then G_t is bipartite. Therefore, corresponding to Lemma 2.1 we have the following.

Lemma 3.1. *$Q_1(t)$ is true if and only if there exists a stabbing line for E_t.*

An algorithm for finding a stabbing line for a set of m line segments in $O(m \log m)$ time has been given by Edelsbrunner et al.[6]. For our problem, this gives an $O(n^2 \log n)$ bound for determining if E_t has a stabbing line. We may therefore test $Q_1(t)$ in $O(n^2 \log n)$ time. By using the algorithm $TEST-GRAPH-3\ (G_t)$, algorithm $MIN-DIAM$ may be used to find the smallest t such that $Q_1(t)$ is true in time $O(n^2 \log^2 n)$. This proves theorem 3.2.

Theorem 3.2. *A minimum diameter separable partition can be found in time $O(n^2 \log^2 n)$.*

> **Algorithm** $TEST-GRAPH-3\ (G_t)$
>
> **begin**
>
>   **if** G_t has a stabbing line **then return (true)**
>
>   **else return (false)**
>
> **end;**

4. Specified Diameter Partitions

In this section we describe an algorithm that takes as input a point set P and four distinct points p_a, p_b, p_c, p_d of P. The algorithm finds a partition of P into subsets S containing p_a, p_b and T containing p_c, p_d, such that $diam(S) = d(p_a, p_b)$ and

$diam(T) = d(p_c, p_d)$, or decides that such a partition does not exist. For convenience throughout we assume that

$$d(p_a, p_b) \le d(p_c, p_d).$$

As in previous sections, our approach is to construct an auxiliary graph $G = (P, E)$, where (p_i, p_j) is an edge in E if and only if

$$d(p_i, p_j) > d(p_a, p_b).$$

We will also colour the edges E green and black. Edge (p_i, p_j) is green if $d(p_i, p_j) \le d(p_c, p_d)$ and black otherwise. Then S and T is a *valid partition* of P if and only if

(i) S is an independent set in G;

(ii) T only contains green edges.

Figure 4 illustrates the construction of G and a valid partition.

Algorithm $SPEC-DIAM(P, a, b, c, d)$ finds a valid partition of P or concludes that such a partition does not exist. The idea of the algorithm is as follows. S must contain p_a, p_b and T must contain p_c, p_d. Given any point x in S, any vertex y adjacent to x must lie in T. Similarly, if x is a vertex in T that is adjacent to a vertex y by a black edge, then y must be in S. The algorithm begins by repeatedly assigning vertices to subsets using the above two rules until either a conflict occurs, or no further vertices can be assigned. In the first case, the algorithm terminates because P is not partitionable. In the second case, as will be shown later, the remaining vertices U must induce a bipartite graph or again P is not partitionable. The algorithm terminates by finding a bipartition of U and by arbitrarily assigning one part to S and one part to T.

In the algorithm Q is a queue that holds points, for which the correct subset has been determined, until they are scanned. For any subset U of P, $G(U)$ denotes the subgraph of G induced by U. That is, points in U are joined if and only if they are joined in G.

Algorithm $SPEC - DIAM(P, a, b, c, d)$

1. **set** $S' = \{p_a,\ p_b\} \quad T' = \{p_c,\ p_d\}$;

   set $Q = \{p_a,\ p_b,\ p_c,\ p_d\}$;

2. **while** $Q \neq \phi$ **begin**

   remove x from Q;

   for every edge $(x, y) \varepsilon E$ **do**

   begin

   case $\{x \varepsilon S'\}$

   if $y \varepsilon S'$ **then return (false)**

   else if $y \not\varepsilon T'$ **then** $T' = T' \cup \{\ y\ \}$, $Q = Q \cup \{\ y\ \}$;

   case $\{x \varepsilon T'$ and $(x,\ y)$ *is black* $\}$

   if $y \varepsilon T'$ **then return (false)**

   else if $y \not\varepsilon S'$ **then** $S' = S' \cup \{\ y\ \}$, $Q = Q \cup \{\ y\ \}$;

   end {for}

   end {while}

   set $U = P - S' - T'$;

3. **if** $G(U)$ is not bipartite **then return (false)**

   else let S^* and T^* be a bipartition of $G(U)$;

   set $S = S' \cup S^* \quad T = T' \cup T^*$;

   return (true).

Figure 5 illustrates the situation after step 2 terminates. For clarity, the edges from p_e and p_f are not shown.

The correctness of the algorithm depends on the following lemmas. We say that a step terminates successfully if the **return (false)** exit is not taken.

Lemma 4.1. *If step 2 of SPEC-DIAM terminates successfully:*

(a) *S' is an independent set in G;*

(b) *T' contains only green edges;*

(c) *Every valid partition (S, T) of P satisfies $S' \subseteq S$, $T' \subseteq T$.*

Proof: Suppose that $x, y \in S'$ and $(x, y) \in E$. Clearly $\{x, y\} \neq \{a, b\}$. Further, both x and y must have been placed into Q at some time. Let x be the point first considered in the **while** loop. At this stage, y is not in S otherwise SPEC-DIAM would return false. But then y is put into T' by the first **case** statement in the **for** loop. Once a point is placed in either S' or T' it can not be placed into the other subset. Therefore (a) holds.

Suppose now that $x, y \in T'$, and (x, y) is a black edge. Therefore $\{x, y\} \neq \{c, d\}$. Again let x be processed before y in the **while** loop. Then y is not in T' when x is processed and so y is placed in S' and (b) holds.

It remains to verify (c), Obviously (c) is true when we begin to execute step 2. We need only verify inductively that as each new point is added to S' or T' (c) remains true. But if a point about to be added to T' were put into S' condition (i) would be violated, so that S could not be extended into a valid partition. Similarly, if a point about to be added to S' were added to T', condition (ii) would be violated.

Lemma 4.2. *If step 2 of SPEC-DIAM terminates successfully:*

(a) *The only edges with exactly one endpoint in U are green;*

(b) *U induces a bipartite subgraph in G if and only if P has a valid partition.*

Proof: Part (a) follows from the fact that every point in $S' \cup T'$ is at some time placed in Q. The first **case** statement ensures that no edges go from S' to U; the second **case** statement ensures that no black edges go from T' to U.

For part (b) we require the following definition.

$$Lune(p_a, p_b) = \{x \mid \max (d(x, p_a), d(x, p_b)) < d(p_a, p_b)\}$$

If $y \in P$ and y is not in $Lune(p_a, p_b)$ then y is adjacent to either p_a or p_b in G. Since both p_a and p_b are considered in step 2, y must be put into T'. Therefore,

$$U \subseteq Lune(p_a,\ p_b)$$

This is illustrated in Figure 5. Join p_a to p_b by a line segment splitting $Lune(p_a,\ p_b)$ into two regions L and R. For definiteness, we include the line segment in R. Now it is easily verified that if $(x,\ y) \varepsilon L$ or $(x,\ y) \varepsilon R$, then $d(x,\ y) < d(p_a, p_b)$. Therefore (x, y) is not an edge in E. Therefore, $U \cap L$ and $U \cap R$ is a bipartition of U, hence U induces a bipartite subgraph.

We may now state the main result of this section.

Theorem 4.3. *SPEC-DIAM finds a valid partition of P or concludes none exists in time* $O(n^2)$.

Proof: By Lemma 4.1 (a)-(c), if step 2 terminates successfully then S' and T' satisfy conditions (i) and (ii) and are extendable to all valid partitions of P. If the step is unsuccessful, then P cannot be partitioned. Step 3 terminates successfully whenever U is bipartite. Let S^* and T^* be the bipartition produced by the algorithm. We claim that $S = S' \cup S^*$ and $T = T' \cup T^*$ is a valid partition of P. By Lemma 4.2(a) there are no edges from S' to S^*, thus (i) holds. Similarly, there are only green edges between T' and T^*. Thus (ii) holds, proving the claim. The time bound is easily verified.

5. Generalizations and Open Problems.

We first consider generalizing the results in the preceding sections to d dimensions. As mentioned in Section 2, no restrictions are placed on the distance function for the results in that section. Therefore a minimum diameter balanced partition can be found in time $O(dn^2 \log n)$.

In Section 3, the dimension of the problem plays a role. In higher dimensions, we require the subsets to be separated by a hyperplane. Using the method of Section 3, we need an efficient algorithm to find a hyperplane that intersects a given set of line segments. Such an algorithm has been announced in[4] for $d = 3$ that runs in $O(n^3)$ time.

For the problem of partitioning a set into subsets with given diameters, considered in Section 4, the results again do not generalize immediately to higher dimensions. The argument in the proof of Lemma 4.2(b) does not work even in three dimensions. Perhaps step 2 of $SPEC - DIAM$ can be applied recursively to the unpartitioned subset U.

A different type of generalization is to consider partitions into three or more subsets. Here the results for the general problem are negative. It has been shown independently in [9] [13] [15] that the m-centers problem mentioned in the introduction is NP-complete. As pointed out in[11] a similar argument shows that the problem of partitioning a planar point set into k subsets of diameter t is also NP-complete.

Finally there is the question of lower bounds. The author knows of no non-trivial lower bounds for any of the problems considered here. Therefore it is an open problem whether any of the algorithms given here are optimal.

6. Acknowledgements.

The author wishes to thank two anonymous referees for suggesting improvements to the original paper.

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass., 1974.

2. D. Avis, ‘‘Non-Partitionable Point Sets,’’ *Information Proc. Letters*, vol. 19, pp. 125-129, 1984.

3. D. Avis, ‘‘On the Partitionability of Point Sets in Space,’’ *Proceedings of the Symposium on Computational Geometry*, pp. 116-121, ACM, Baltimore, MD, 1985.

4. D. Avis and M. Doskas, ‘‘Algorithms for High Dimensional Stabbing Problems,’’ *Discrete Applied Math.*, vol. 27, pp. 39-48, 1990.

5. D.P. Dobkin and H. Edelsbrunner, ‘‘Space Searching for Intersecting Objects,’’ in *Proceedings of the 25th FOCS*, pp. 387-391, IEEE, Ocober, 1984.

6. H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A.L. Rosenberg, E. Welzl, and D. Wood, ‘‘Stabbing Line Segments,’’ *BIT*, vol. 22, pp. 274-281, 1982.

7. H. Edelsbrunner and E. Welzl, ‘‘On the Number of Line Separations of a Finite Set in the Plane,’’ *Journal of Combinatorics A*, vol. 38, pp. 15-29, 1985.

8. P. Erdős, L. Lovász, A. Simmons, and E. G. Straus, ‘‘Dissection Graphs of Planar Point Sets,’’ in *A Survey of Combinatorial Theory*, ed. J.N. Srivastava et al., pp. 139-149, North-Holland, 1973.

9. R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, ‘‘Optimal Packing and Covering in the Plane are NP-Complete,’’ *Information Processing Letters*, vol. 12, pp. 133-137, 1981.

10. M.R. Garey and D.S. Johnson, *Computers and Intractability,* Freeman, New York, 1979.

11. D. S. Johnson, ‘‘NP-Completeness Column,’’ *J. of Algorithms*, vol. 3, pp. 182-195, 1982.

12. L. Lovász, ‘‘On the Number of Halving Lines,’’ *Ann. Univ. Sci. Budapest*, vol. Sect. Math. 14, pp. 107-108, 1971.

13. S. Masuyama, T. Ibaraki, and T. Hasegawa, ‘‘The Computational Complexity of the m-centers Problem in the Plane,’’ *Trans. IECE of Japan*, pp. 57-64, 1981.

14. M. I. Shamos, ‘‘Computational Geometry,’’ *Ph.D. Thesis*, Yale University, 1978.

15. K. Supowit, ‘‘Topics in Computational Geometry,’’ Report No. UIUCDCS-R-81-1062, Dept. of Computer Science, University of Illinois, Urbana, Ill., 1981.

16. D. E. Willard, ‘‘Polygon Retrieval,’’ *SIAM Journal on Computing*, vol. 11, pp. 149-165, 1982.

17. F. Yao, ‘‘3-Space Partition and its Applications,’’ in *Proceedings of the 15th STOC*, pp. 258-263, A.C.M., Boston, Massachusets, 1983.