SPACE PARTITIONING AND ITS APPLICATION
TO GENERALIZED RETRIEVAL PROBLEMS

David Avis

School of Computer Science
McGill University
805 Sherbrooke Street West
Montreal, Quebec, Canada H3A 2K6

1. Introduction

A fundamental problem in the area of database systems is the retrieval of data satisfying certain criteria. The simplest such problem is the retrieval of data based on a single key, and a comprehensive treatment of this problem is contained in Knuth[10]. A more difficult problem is the retrieval of data based on criteria for several keys, the so-called multidimensional search problem. Suppose that each of n records of a *file* contains a fixed number, d, of *keys.* A *query* asks for all records for which the d keys satisfy certain criteria. For example, upper and lower bounds maybe specified for some or all keys, yielding the so-called *orthogonal range query problem.*

In this paper we study the multidimensional search problem from a geometric standpoint. We consider the $d-tuple$ of keys for a given record as a point in d-dimensional space, R^d. The previously mentioned orthogonal range query problem can now be interpreted as finding all points in a given $d-dimensional$ hyperrectangle. Suppose we are given upper bounds, u_i, and lower bounds l_i for each of the $i = 1, 2, \cdots, d$ keys. Then we are asked to find every record whose key $(x_1, \cdots, x_d)$ satisfies:

$$l_i \le x_i \le u_i \qquad i = 1, \ldots, d.$$

A more general query could consist of a linear function: $a_1x_1 + a_2x_2 + \cdots + a_dx_d + a_{d+1}$. The *half-space query problem* is to find all records whose keys satisfy:

$$a_1x_1 + a_2x_2 + \cdots + a_dx_d + a_{d+1} \le 0.$$

Even more generally, we may be given a set of k such linear functions, which we represent in matrix form:

$$A = \begin{bmatrix} a_{11} & . & . & . & a_{1d} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ a_{k1} & . & . & . & a_{kd} \end{bmatrix} \quad , b = \begin{bmatrix} a_{1,d+1} \\ . \\ . \\ . \\ a_{k,d+1} \end{bmatrix}$$

Letting $x = (x_1, x_2, \ldots, x_d)$ be the *d-vector* of keys for a record, we search for all such vectors satisfying

$$Ax + b \le 0.$$

Such a query is called a *polyhedral range query.* Note that the orthogonal range query problem is just a special case of this problem obtained by letting $k = 2d$, and by setting

$$\left.\begin{array}{lcr} a_{2i-1,i} & = & 1 \\ a_{2i,i} & = & -1 \\ a_{2i-1,d+1} & = & -u_i \\ a_{2i,d+1} & = & l_i \end{array}\right\} \quad i = 1, 2, \ldots, d$$

$$a_{ij} = 0 \quad otherwise.$$

An excellent survey of orthogonal range searching is contained in the paper by Bentley and Friedman[4].

As an example of when a polyhedral range query may be necessary, consider the following student database. Each record contains 6 keys representing the student ID number, and the number of A, B, C, D and F grades obtained. The ID number is a 7 digit integer of the form, *yyzzzzz*, where *yy* indicates the student first registered in 19*yy*. Assume a grade point scheme where A=4, B=3, C=2, D=1 and F=0. The grade point average (GPA) is defined to be the average of the grades obtained, using the above numerical conversion. An orthogonal range query may be used to list, for example, all students with more than 4 F's who started in or before 1979:

$$x_1 \leq 7999999$$

$$x_6 \geq 4.$$

But suppose we want to find all students with GPA at least 3.2 and who started in 1980 or 1981. Then we need a general half plane query:

$$.8x_2 - .2x_3 - 1.2x_4 - 2.2x_5 - 3.2x_6 \geq 0$$

$$x_1 \geq 8000000$$

$$x_2 \leq 8199999$$

Similar examples can be found in any application involving several numeric fields: payroll, inventory control, accounting etc.

A completely different application for polyhedral query problems arises when the data is itself geometrical. In a geometrical database, the data corresponds to physical objects in space. For example, we may have a digital representation of a satellete photograph, where we have x, y coordinates for every building or road in the picture (see [1]). We may be required to retrieve all points lying inside a given region, for example, a municipality. If the region can be described by straight-line segments, then this query is a 2-dimensional polyhedral query (or perhaps several queries if the region is non-convex). This type of 2-dimensional query is called a *polygonal query* [17]. Sometimes even regions with curved boundaries can be transformed into polyhedral queries. Yao [19] gives the example of circular queries in 2-dimensions. We wish to find all records satisfying

$$(x_1 - a_1)^2 + (x_2 - a_2)^2 - a_3{}^2 \leq 0. \tag{1}$$

For example, we may be interested in all dwellings within 50km of a TV transmitter. This query problem can be mapped to a polyhedral query in 3 dimensions. For each key (x_1, x_2) create the 3-dimensional key $(x_1, x_2, x_3) = (x_1, x_2, x_1{}^2 + x_2{}^2)$. The inequality (1) can be rewritten:

$$-2a_1x_1 - 2a_2x_2 + x_3 + (a_1^2 + a_2^2 - a_3^2) \leq 0$$

which is a half-space query. Similarly, any d-dimensional spherical query can be reformulated as a $d+1$-dimensional half-space query.

In this paper we consider data structures for storing the keys for each record in a file. Using the geometrical approach, we consider d-dimensional space to be partitioned into regions. All keys falling into the same region of space are thus associated together in the data structure. In the next section, we review several successful space partitioning techniques, including quad trees, $k-d$ trees and multipaging. Whilst these methods work well for orthogonal range queries, they do not seem well suited for the more general half-space queries. For these types of queries, a new space partitioning approach introduced by Willard [17] for 2-dimensions, and generalized by Yao [19] for 3-dimensions is discussed. Section 3 contains theoretical results relating to the new type of data structure. The results of Willard and Yao are reviewed and new results for higher dimensions are given. The best known algorithms for the various basic data structure operations are also reviewed. Finally, in the fourth section, we mention some of the many open problems related to this data structure. Some conclusions as to its applicability are also given.

2. Space Partition

In this section, we discuss various techniques for partitioning a set of records given as points in d-dimensional space. Let $S = \{ p_1, \ldots, p_n \}$ be a set of points in R^d, each corresponding to a record of a file with d-keys. The first three partitioning techniques that we will consider all consist of partitions of S by hyperplanes that are parallel to the coordinate axes. These techniques are respectively: quad trees, $k-d$ trees and multipaging. In analysing these structures we will consider the following cost functions:

$P(n, d)$: the cost of preprocessing S into a data structure,

$S(n, k)$: the storage required by the data structure,

$I(n, k)$: the cost of inserting a new point into the data structure,

$D(n, k)$: The cost of deleting a point from the data structure.

One of the earliest data structures of a space partitioning nature is the quad tree [7]. A quad tree for S is built as follows. First take some point p_i as the root. Taking p_i as the origin, d-space is split into 2^d orthants by considering the d hyperplanes parallel to the coordinate axes. Each orthant is then split again using the same technique. This is continued until each region has exactly one point, corresponding to a leaf of the tree. An example is given in Figure 1. In a quad tree, the internal nodes each have degree at most four. In degenerate cases, as for example when all the points lie on a diagonal, the internal nodes have degree two and the depth of the tree is $\log_2 n$ rather than the best possible bound of $\log_{2^d} n$. In [7] an expected time insertion algorithm of $O(\log n)$ is given, however no efficient deletion algorithm is given. Using the subscript Q to denote quad tree we have

$$P_Q\,(n, d) = O\,(\,n \log n\,)$$
$$S_Q\,(n, d) = O\,(\,n\ d\,)$$
$$I_Q\,(n, d) = O\,(\,\log n\,).$$

Overmars and van Leeuwen [15] describe a variant of the quad tree called the pseudo-quad tree that allows both efficient insertions and deletions. Denoting this data structure by PQ, they prove

$$P_{PQ}\,(n, d) \;=\; O\,(\,n \log n)$$
$$S_{PQ}\,(n, d) \;=\; O\,(\,n\ d\,)$$
$$I_{PQ}\,(n, d) \;=\; D_{PQ}\,(n, d) \;=\; O(\,\frac{1}{\delta} \log^2 n\,)$$
$$\text{for } \textit{every constant } 0 < \delta < 1 \;\; (\textit{average}).$$

Another space-partitioning data structure is the multidimensional binary search tree or $k-d$ tree due to Bentley [3]. A $k-d$ tree for S is built as follows. Again we choose some point p_i from S. Using the first coordinate only, we split S into two subsets. The point p_i is made the root of the $k-d$ tree and the two subsets are made its two sons. In each subset we again choose a point and split according to the second coordinate. This procedure is repeated for the first d levels in the tree, and is then repeated for each set of d levels, as required until each region contains one point. An example for $d = 2$ is shown in Figure 2. Bentley gives algorithms for $k-d$ trees which yield

$$P_{kd}\,(n, d) \;=\; O(\,dn \log n\,)$$
$$S_{kd}\,(n, d) \;=\; O(\,n\ d\,)$$
$$I_{kd}\,(n, d) \;=\; O\,(\,\log n\,) \quad \textit{average}$$
$$D_{kd}\,(n, d) \;=\; O\,(\,n^{1-\frac{1}{d}}\,) \quad \textit{worst case}$$
$$= O\,(\,\log n\,) \quad \textit{average}$$

Again, Overmars and van Leeuwen [15] are able to improve on the insertion deletion times by introducing a modification to the data structure. They obtain results similar to those already quoted for quad trees.

In contrast to the two tree based methods described here are the so-called cell based methods for direct access to multidimensional data. Schemes are given by Rothnie and Lozano [16], Liou and Yao [11] and Merrett [13]. A convenient description of these techniques and detailed analysis of the latter technique, called multipaging, is contained in the thesis of Otoo[14]. The basic idea of the cell method is to partition the space by families of hyperplanes parallel to the given coordinate axes. For the i^{th} axis, $i = 1, \ldots, d$, some number k_i of values are chosen. Suppose these values are $a_{i1}, a_{i2}, \ldots, a_{ik_i}$. Then d-space is partitioned by the $k_1 + \cdots + k_d$ hyperplanes of the form:

$$x_i \;=\; a_{ij} \qquad j = 1, \ldots, k_i \;, \quad i = 1, 2, \ldots, d \;.$$

This produces a partition of space into

$$(\, k_1 + 1 \,) \;\; (\, k_2 + 1 \,) \;\; \cdots \;\; (\, k_d + 1 \,)$$

hyperrectangular regions. An example for $d = 2$ and $k_1 = 2, k_2 = 3$ is shown in Figure 3. The idea is to make the cells small enough so that all of the points in S lying in a single cell can fit in one page of memory. In cases where the points are highly clustered, some overflow mechanism is required to make use of underoccupied cells. If n pages or cells are generated the directory requires $0(\, dn^{\frac{1}{d}} \,)$ space. For this type of data structure, worst case results are usually very bad, however they give good average time performance.

The common feature of all of the above schemes is that the partitioning hyperplanes are all parallel to the coordinate axes. Thus the regions created are always hyperrectangles. In the fourth data structure to be described this restriction is dropped. This data structure was first described by Willard[17] who called it a *polygon tree* in two dimensions. In general, we will call it a *polyhedron tree.* In d –space, d mutually intersecting hyperplanes are chosen. These decompose space into 2^d regions. The d hyperplanes are associated with the root of the tree. The subsets of S that lie in each of the 2^d regions form the 2^d sons of the root. Each of these regions is again subdivided by d mutually intersecting hyperplanes. The process is continued until each region contains at most one point of S. An example for $d = 2$ is shown in Figure 4. The similarity with quad trees is readily apparent. The power of polyhedron trees is the ability to partition points of S into subsets of essentially the same cardinality. Consider figure 5(a) and (b). In the quad tree, 5(a), two of the sons of the root contain almost no points. In a polygon tree, 5(b), however, each son has one quarter of the points. It is perhaps somewhat surprising that in two dimensions, S can slways be partitioned by two lines so that each open region contains at most a quarter of the points[17]. Yao[19] gives a similar result for 3 dimensions. These theoretical results are described in the next section. We now outline how polyhedral trees can be used to solve the half-space query problem in sublinear time.

For simplicity, we consider the 2-dimensional problem. Assume that we have built a polygon tree as illustrated in figure 4. The algorithm for half-plane queries consists of a standard top down walk through the polygon tree. All nodes whose ranges intersect with the requested half-plane are visited. Each leaf visited will have the coordinates of the point it contains tested against the given query. The success of the algorithm depends on the following observation:

> A half-plane can intersect at most three of the four regions formed by a partition of the plane by two lines.

This means that at most three of the four subtrees of an internal node need to be visited. Let h be the height of the polygon tree. Then a simple analysis shows that the total number of nodes visited is

$$\sum_{d=0}^{h} 3^d \;\;=\;\; O\,(\, n^{\log_4 3} \,)\,, \qquad \log_4 3 = .792\ldots\,,$$

where n is the number of data items in the tree. If k data items are contained in the output of the query, then the algorithm has sublinear complexity $O\,(\, n^{.792} + k \,)$. By partitioning the data into 6 equal cardinality regions by three lines, this complexity can be reduced to $O\,(\, n^{.774} + k \,)$,[6]. Furthur optimizations were obtained by Edelsbrunner and Welzl [6]. By reusing some subdividing lines they reduce the exponent to 0.694.

Analagous results hold in three dimensions[19]. A half-space in three dimensions can intersect at most 7 of the 8 regions formed by a partition by three intersecting planes. This gives rise to a half-space query algorithm of complexity $O(n^{.936} + k)$, since $\log_8 7 = .936 \ldots$.

It is clear that the success of these methods depends on efficient space partition. Generalizations to higher dimensions will depend on generalizing the results on space partitioning. These topics are discussed in the next section.

3. Geometrical Results

In this section we discuss the geometrical results which are the basis for fast search algorithms that use polyhedron trees. Some of these results are rather technical, and the reader who is not mathematically inclined may skip most of this material. From an applications point of view, the important results are corollary 2.1, theorem 3, and theorem 6. The reader is referred to the corresponding references to see how these results are applied. For the interested reader, the development in this section will be done informally to give some intuition into the results. For formal proofs, the reader is referred to the original sources. We begin with some terminology.

Let S be a set of n points in R^d and let $0 < \alpha < 2^{-d}$. We say that S is $\alpha -$ *partitionable* if there exist a set of d intersecting hyperplanes that partition S so that no open region contains more than αn points of S. We say that S is *non-partitionable* if for every partition of S by d intersecting hyperplanes, some set of $2^d - 1$ open regions together contain all the points of S. A point x in space is called a *centre* of S if every hyperplane H through x contains at least $\frac{n}{(d+1)}$ points in each half-space bounded by H. Two classical results are the Centre Theorem and the Ham Sandwich Theorem.

Theorem 1. Centre Theorem[18]

Every set S in R^d has a centre.

Theorem 2. Ham-Sandwich Theorem[12]

Given d disjoint sets of points in R^d, there is a hyperplane which simultaneously divides each point set evenly.

From theorem 2, we get the immediate corollary:

Corollary 2.1[17]

Every set S in the plane can be $\frac{1}{4}$ - partitioned. Furthermore one of the partitioning lines may be chosen to be an arbitrary bisector of S.

Proof: Choose a line l that bisects S. This can easily be found by choosing the median point in one of the coordinate directions and taking a line parallel to the appropriate coordinate axis. This divides the set S into two disjoint sets, to which theorem 2 may be applied

Corollary 2.1 thus shows that a perfectly balanced polygon tree can always be constructed. This justifies the sublinear half-plane algorithm described in the previous section. Before we move to three dimensions, another 2-dimensional result is required.

Lemma 1[19]

Let R and B be two n point sets in the plane with a common centre. Then R and B can be simultaneously $\frac{1}{12}$ - partitioned by two lines.

It can now be shown that all sets in R^3 can be partitioned.

Theorem 3[19]

Every set S in 3-dimensions can be $\frac{1}{24}$ - partitioned. Furthermore, one of the partitioning planes may be chosen to be any bisector of S.

Proof: Let H be any bisecting plane through S, splitting S into sets S_1 and S_2. Let x_1 be a centre for S_1 and x_2 be a centre for S_2. Map every point of S onto H by projecting parallel to the line through x_1 and x_2. This gives rise to sets R and B of size $\frac{n}{2}$ that satisfy the conditions of lemma 1. Therefore R and B can be $\frac{1}{12}$ - partitioned by two lines l_1 and l_2. Let H_i be the plane containing l_i, x_1 and x_2, $i = 1, 2$. Then H, H_1, H_2 form a $\frac{1}{24}$ - partition of S.

This result is sufficient to get an $O(n^{.98} + k)$ algorithm for half-space queries in three dimensions. To get the faster algorithm described in section 2, we require the following stronger and more difficult result.

Theorem 4[20] [5]

Every set S in 3-dimensions can be $\frac{1}{8}$ - partitioned.

In [5] , a proof is given based on Theorem 2. The proof is rather technical and is not included here. Yao has apparently proved that theorem 4 can be strengthened to allow one hyperplane to be chosen as an arbitrary bisector of S[20].

In four dimensions, it is not known whether an analogue of corollary 2.1 and theorem 4 holds. In the strengthened form they definitely do not hold.

Theorem 5[2]

For any $\alpha > 0$ and any n, there exists a set S in R^4, of cardinality n, that cannot be α - partitioned by a prescribed bisecting hyperplane and any three other hyperplanes.

In dimensions of 5 and higher, the situation becomes worse. Let $\delta_d = d \bmod 2$.

Theorem 6[2]

For any integer n, and $d \geq 5$, there exist non-partitionable sets of cardinality n. Furthermore, sets exist for which every partition by d hyperplanes leaves at least $2^d - d^2 - \delta_d$ open regions with no points of S.

The "nasty" point sets of theorems 5 and 6 are easily described. For any integer t, let $s_t = (t, t^2, \ldots, t^d)$. For any integer n we set

$$S = \{ s_t : 1 \leq t \leq n \}. \tag{2}$$

These sets play a fundamental role in the theory of convex polyhedra, as they form the vertices of the *cyclic polytopes*. An accessible introduction to these fascinating polytopes is the paper by Gale[8]. Let

$$f(x_1, \ldots, x_d) = \sum_{i=1}^{d} a_i x_i + a_{d+1}.$$

We denote by f the hyperplane $f(x_1, x_2, \ldots, x_d) = 0$ in d-space. The set S is partitioned by f according to the sign of the function

$$g(t) = \sum_{i=1}^{d} a_i t^i + a_{d+1} .$$

Integers t for which $g(t)$ is positive correspond to points s_t in S that lie in one of the open half-spaces bounded by f, and integers t for which $g(t)$ is negative correspond to points in the other open half-space. Let H and L, respectively denote these point sets. Integers for which $g(t) = 0$ correspond to points of S on f. Such points are denoted by M. Since g is a polynomial of degree d, it has some number k, which is at most d, of real roots. Therefore f can contain at most d points of S. If k is zero, then all points of S lie on the same side of f. If k is positive, let $t_1 \leq t_2 \leq \cdots \leq t_k$ denote the real roots of g. Then

$$M = \{ t_1, \ldots, t_k \},$$

and H and L contain $k + 1 \leq d + 1$ strings of consecutive points of S. If we consider the d partitioning hyperplanes together, it can be shown using this fact[2] that at most $d^2 + 1$ of the 2^d open regions contain points of S. In fact, if d is even, only d^2 regions can contain points. This outlines the proof of theorem 6.

For theorem 5, we observe that $d = 4$ implies $d^2 = 2^d$, and so the previous argument does not yield anything of interest. However the situation is very "tight" in the following sense. The set S defined by (2) with $d = 4$ can be partitioned if and only if there exists a binary matrix of dimension 4×16 with the following properties:

P1: Every column of A is distinct.

P2: Each row of A consists of at most 5 strings of zeroes and ones.

P3: Two consecutive columns of A differ in exactly one row.

Properties P_1 and P_3 define a Gray code. Gilbert[9] has published a list of all Gray codes of length 16. Of the 9 such codes, one satisfies P_2. It is:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

This matrix determines how the four intersecting hyperplanes must partition S. Suppose S contains 16 points. Row 1 of A can be interpreted as saying that points $s_1, s_4, s_5, s_6, s_{13}, s_{14}, s_{15}$, and s_{16}

$$s_1, s_4, s_5, s_6, s_{13}, s_{14}, s_{15}, \text{ and } s_{16}$$

must lie above hyperplane H_1. Row 2 can be similarly interpreted for H_2, etc. Following this prescription, a separating set of four hyperplanes can be contructed. But suppose we begin with a bisecting hyperplane H such that $s_1, s_3, s_5, \ldots, s_{15}$ lie above H and $s_2, s_4, \ldots, s_{16}$ lie below H. Such a pattern does not appear in A (or any cyclic permutation of the columns of A). Since A is essentially unique, this shows that H cannot be used in any partition of S by 4 hyperplanes. A similar argument can be shown for arbitrary n, demonstrating theorem 5.

As a final result of this type, we show that lemma 1 cannot be generalized to 3-dimensions.

Theorem 7

There exist two n point sets R and B in 3 dimensions, with a common centre, such that R and B are not simultaneously partitionable.

Proof: (Outline)

Consider the set S and hyperplane H described in the last paragraph. Let S_1 and S_2 be the sets separated by H. Let x_1 and x_2 be points in the centres of S_1 and S_2. Project S_1 and S_2 onto H parallel to the line through x_1 and x_2 giving R and B respectively. Now suppose that R and B could be simultaneously partitioned by three planes. Then as in lemma 1, this partition could be extended to a partition of S in 4-dimensions by 4 hyperplanes, one of which was H. But by the remarks preceding the theorem, this is impossible. Hence R and B may not be simultaneously partitioned.

The preceding results are negative and tell us that the algorithms for 2 and 3 dimensions may not be extended easily to high dimensions. Theorem 6 tells us that sets in 5-dimensional space may not be separable by 5 hyperplanes. This leads us to ask: Is there some suitably high dimensional space, such that every set of points can be partitioned by 5 hyperplanes? This leads to the following definition. Let k be any positive integer. Then $f(k)$ is the smallest dimension such that every point set in $f(k)$ - dimensional space can be 2^{-k} - partitioned by k mutually intersecting hypeplanes. Then we can prove:

Theorem 8

$$\frac{2^k - 1}{k} \leq f(k) \leq 2^{k-1} .$$

4. Conclusions

In this paper we have surveyed various techniques for partitioning multidimensional data for efficient searching. The standard methods involve partitioning data by hyperplanes parallel to the coordinate axes. These techniques prove very efficient for orthogonal range queries but are not useful for more general half-space queries. This type of query can be more efficiently answered by a new type of data structure called a polyhedron tree, when the dimension is 2 or 3. For higher dimensions, it has been demonstrated that these trees may not always be constructible.

Many open questions remain. In particular, the non-partitionable data set constructed in dimensions 5 and above is highly pathological. One wonders if in practice, most data sets in dimensions 4, or 5 may be partitionable. There also remains the problem of finding fast partitioning algorithms, and the problem of updating the data base. Perhaps there is a role for heuristics here. Finally, there is the theoretical question of obtaining better bounds on the function $f(k)$ described at the end of the last section.

5. Acknowledgements

Many of the results reported in this paper were obtained whilst the author was a visiting researcher at the University of Tokyo, supported by a grant from the Japanese Society for the Promotion of Science. The author expresses his thanks to Professor Masao Iri for providing such a stimulating research environment. The author would also like to thank an anonymous referee for a carefull reading of the paper.

References

1. M. Iri et al., In JapaneseFundamental Algorithms for Geographic Data Processing,'' *Operations Research Society of Japan*, vol. TR 83-1, 1983.
2. D. Avis, In JapaneseNon-Partitionable Point Sets,'' *Information Proc. Letters*, vol. 19, pp. 125-129, 1984.
3. J. L. Bentley, In JapaneseMultidimensional Binary Search Trees used for Associative Searching,'' *Comm. ACM*, vol. 18, pp. 509-517, 1975.
4. J.L. Bentley and J. H. Friedman, In JapaneseData Structures for Range Searching,'' *Comp. Surveys*, vol. 11, pp. 397-409, 4, 1979.
5. D.P. Dobkin and H. Edelsbrunner, In JapaneseSpace Searching for Intersecting Objects,'' in *Proceedings of the 25th FOCS*, pp. 387-391, IEEE, Ocober, 1984.
6. H. Edelsbrunner and E. Welzl, In JapaneseHalfplane Range Search in Linear Space and $O(n^{0.695)}$ Query Time,'' F111, Technical University of Graz, 1982.
7. R.A. Finkel and J. L. Bentley, In JapaneseQuad Trees: A Data structure for retreival on composite keys,'' *Acta Informatica*, vol. 4, pp. 1-9, 1974.
8. D. Gale, In JapaneseNeighborly and Cyclic Polytopes,'' *Proceedings of the Symp. in Pure Mathematics*, vol. 7, pp. 225-232, 1963.
9. E. N. Gilbert, In JapaneseGray Codes and Paths on the n-cube,'' *Bell System Technical Journal*, pp. 815-826, 1958.
10. D. E. Knuth, *The Art of Computer Programming, Vol 3: Sorting and Searching,* Addison-Wesley, Reading, MA, 1973.

11. J. H. Liou and S. B. Yao, In JapaneseMultidimensional Clustering for Data Base Organization,'' *Information Systems*, vol. 2, pp. 187-198, 1977.

12. B. Mendelson, *Introduction to Topology,* Allyn & Bacon, Boston, 1962.

13. T. Merrett, In JapaneseMultidimensional Paging for Efficient Data Base Querying,'' in *Proc. Int. Conf. on Management of Data, ICMOD*, pp. 277-290, milano, 1978.

14. E. J. Otoo, In JapaneseLow Level Structures in the Implementation of the Relational Algebra,'' SOCS 83.17, McGill University, School of Computer Science, August 1983.

15. M. H. Overmars and J. van Leeuwen, In JapaneseDynamic Multi-Dimensinal Data Structures Based on Quad- and K-D Trees,'' *Acta Informatica*, vol. 17, pp. 267-285, 1982.

16. J. B. Rothnie and T. Lozano, In JapaneseAttribute Based File Organization in Paged Memory Environment,'' *Comm. of ACM*, vol. 17, pp. 63-69, 2, 1974.

17. D. E. Willard, In JapanesePolygon Retrieval,'' *SIAM Journal on Computing*, vol. 11, pp. 149-165, 1982.

18. I.M. Yaglom and V.G. Boltyanskii, *Convex Figures,* Holt, Rinehart and Winston, 1961.

19. F. Yao, In Japanese3-Space Partition and its Applications,'' in *Proceedings of the 15th STOC*, pp. 258-263, A.C.M., Boston, Massachusets, 1983.

20. F. Yao, *Private Communication*, 1983.