



MultiDimensional Divide and Conquer

Team ID - 9

Raja Vechalapu - 2018102032

Vinay Kumar Thumpala - 2019102045

Sai Saharsh Reddy Palla - 2019102012

Vamsi Kashyap Dabbiru - 2018102042

I. INTRODUCTION

The discipline of algorithm design and analysis has made numerous theoretical and practical contributions to computer science. When compared to naive algorithms for the identical work, there are a handful of well-designed algorithms that save users thousands of dollars every month (sorting and Fourier transforms are examples of such tasks). On a more theoretical level, algorithm design has shown a number of surprising conclusions that are fascinating from a mathematics standpoint. However, at first, the field essentially consisted of a dispersed collection of outcomes with little underlying theory. This report introduces a new paradigm in algorithmic solving - Multidimensional Divide and Conquer by discussing two major problems: Domination problems and Closest pair problems.

II. PROBLEM STATEMENT

In our project report, we used the Multidimensional Divide and Conquer paradigm to give best-known solutions to problems like Empirical Cumulative Distribution Function (ECDF), Maxima, Range searching, Closest pair, and All nearest neighbor problems. Almost all of the algorithmic paradigms previously addressed, on the other hand, are at one of two extremes: They are either too general to be explained in detail, or they are too specific to be effective in fixing only one or two situations. In this study, we look at a more "middle of the road" paradigm that may be accurately stated while also being applicable to a wide range of problems. The report calls this strategy "multidimensional divide and conquer."

Multidimensional divide-and-conquer is a single algorithmic paradigm that may be used in a variety of situations. It can be described roughly as follows: to solve a problem of N points in k -space, first recursively solve two problems each of $N/2$ points in k -space, and then recursively solve one problem of N points in $(k-1)$ -dimensional space. In this report, we look at a variety of algorithms to demonstrate how they may all be understood as examples of multidimensional divide-and-conquer.

III. ALGORITHMS PROPOSED in analysis

The following problems describe Multidimensional Divide and Conquer.

A. All-points ECDF problem

1) Problem Definition:

Given a set S of N points we define the rank of point x to be the number of points in S dominated by x . In statistics the empirical cumulative

distribution function (ECDF) for a sample set S of N elements, evaluated at point x , is just $\text{rank}(x)/N$. For this problem, we need to calculate the rank of every point in the set S .

2) Algorithm for solving ECDF2:

- (Division Step) If S contains just one element, then return its rank as 0; otherwise proceed. Choose a cut line L perpendicular to the x -axis such that $N/2$ points on S have an x -value less than L 's (name this set of points A) and the rest have an x -value larger than L 's (call this collection of points B). Note that L is a median x -value of the set.
- (Recursive Step) Recursively call $\text{ECDF2}(A)$ and $\text{ECDF2}(B)$. After this step we know the true ECDF of all points in A .
- (Marriage Step) We must now find for each point in B the number of points in A it dominates (i.e., that have lesser y -value) and add this number to its partial ECDF. To do this, pool the points of A and B (while remembering their types) and sort them by y -value. Scan through this sorted list, increasing y -value as you go, keeping track in ACOUNT of the number of A 's so far observed. Each time a B is observed, add the present value of ACOUNT to its partial ECDF.

3) Algorithm for solving ECDF $_k$:

- Choose a $(k-1)$ dimensional cut plane P that divides S into two $N/2$ point subsets A and B .
- Recursively call $\text{ECDF}_k(A)$ and $\text{ECDF}_k(B)$. The actual ECDF of all points in A is then known.
- (For each B , find the number of A 's it dominates.) Project the points of S onto P , noting whether they were an A or a B for each. Now, using a modified ECDF $(k-1)$ algorithm, solve the reduced problem and add the obtained values to the partial ranks of B .

4) Complexity Analysis:

For ECDF2, by applying presorting technique,

$$T(N) = 2T(N/2) + O(N) \text{ which has solution} \quad (1)$$

$$T(N) = O(N \lg N)$$

For ECDF $_k$,

$$T(N, k) = O(N) + 2T(N/2, k) + T(N, k - 1)$$

We can use as a basis for induction on k the fact that $T(N, 2) = O(N \lg N)$

We can use as a basis for induction on k the fact that as shown previously, and this sets that

$$T(N, k) = O(N \lg^{(k-1)} N) \quad (2)$$

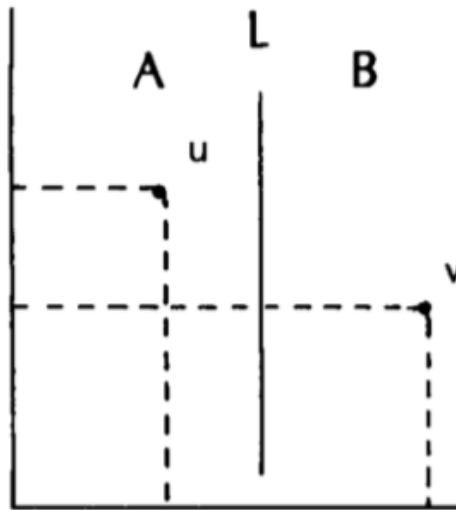


Fig. 1. Two cases of planar queries.

B. ECDF Searching Problem

1) Problem Definition:

Given a set S , organize it into a data structure such that queries of the form "what is the rank of point x " can be answered quickly (where x is not necessarily an element of S).

2) Algorithm for solving ECDF Searching :

- We choose a line L that divides S into equal-sized sets A and B , similar to the all-points approach. We now recursively process subproblems A and B into ECDF trees representing their respective subsets, rather than solving them individually.
- Having built these subtrees we are (almost) prepared to answer ECDF queries in set S .
- A query algorithm's initial step compares the query point's x -value to the line L ; the two possible outputs are shown in Figure 1 as points u and v .

- If the point is to the left of L (as u is), we must recursively search the substructure representing A to get its rank in S, because it cannot dominate any point in B.
- If a point is to the right of L (as v is), then searching B will tell us how many points in B are dominated by v, but we still need to find out how many points in A are dominated by v. To do so, we merely need to calculate v's y-rank in A, as shown in Figure 2.

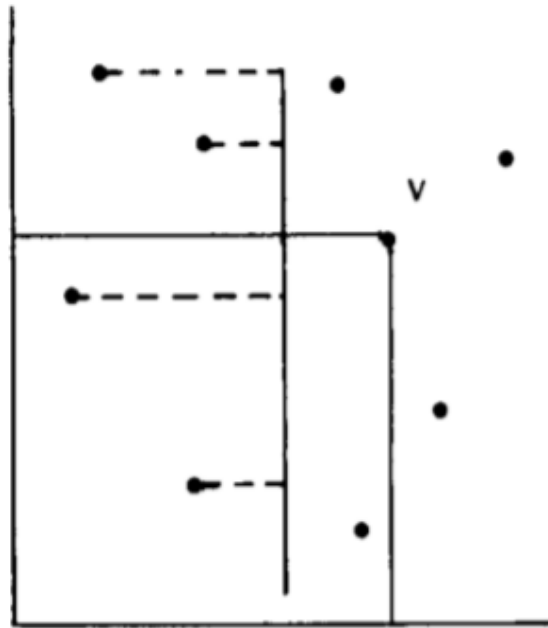


Fig. 2. Calculating v's y-rank in A.

3) Complexity Analysis:

Structures that are searched incur three costs: the preprocessing time required to build the structure, the query time required to search it, and the memory required to keep the structure in memory. When analyzing a structure containing N points, we denote these quantities by $P(N)$, $Q(N)$, and $S(N)$, respectively.

For ECDF-Searching dimension 2:-

$$P(N) = 2P(N/2) + O(N)$$

$$P(N) = O(N \lg N) \quad (3)$$

$$S(N) = 2S(N/2) + N/2$$

$$S(N) = O(N \lg N) \quad (4)$$

$$Q(N) = Q(N/2) + O(\lg N)$$

$$Q(N) = O \lg^{(2)} N \quad (5)$$

For ECDF-Searching dimension k :-

$$P(N, k) = 2P(N/2, k) + P(N/2, k - 1) + O(N)$$

$$S(N, k) = 2S(N/2, k) + S(N/2, k - 1) + O(1)$$

$$Q(N, k) = Q(N/2, k) + Q(N/2, k - 1) + O(1)$$

$$P(N, k) = O N \lg^{(k-1)} N \quad (6)$$

$$S(N, k) = O N \lg^{(k-1)} N$$

$$Q(N, k) = O \lg^{(k)} N$$

C. Maxima Finding

1) Problem Definition:

A point is said to be the maximum of a set if there is no other point that dominates it.

2) Algorithm for solving Maxima in 3 dimension:

- The first step divides into A and B, and the second step recursively finds the maxima of each of the two sets.
- In the third step, each maxima of the set B must be discarded since every maxima of B is a maxima of the set. The planar problem is solved by projecting the respective maxima sets onto the plane and then solving it. It would be possible to modify the two-dimensional maxima algorithm to solve this problem, but the scanning algorithm would be more efficient.
- Consider the case in which A and B are divided by the z-coordinate; that is, we must discard all A's in the x-y plane dominated by any B's. When we have presorted by x, we scan the sorted list right-to-left, discarding all A values with y-values less than the maximum B value observed so far. Due to the linear nature of the marriage step, this algorithm has the same recurrence relation as the two-dimensional algorithm.

3) Algorithm for solving Maxima in k dimension:

- Solving two problems of N/2 points in k-space and then solving one problem of (up to) N points in (k-1)-space.

- This reduced problem calls for finding all As in the space dominated by any Bs, and we can solve this by modifying the maxima algorithm (similar to our modifications of the ECDF algorithm).

4) Complexity Analysis:

$T(N, k) = 2T(N/2, k) + T(N, k - 1) + O(N)$ and we can use the fact that

$T(N, 3) = O(N \lg N)$ to establish that,

$$T(N, k) = O(N \lg^{(k-2)} N) \text{ for } k \geq 3 \quad (7)$$

D. Maxima Searching

1) Problem Definition:

Given a new point on each query, find whether it is a maxima or not.

2) Algorithm:

- A structure representing N points in k-space consists of two substructures representing N/2 points in k-space, as well as one substructure representing N/2 points in (k - 1) space.
- Before determining whether a new point is a maximum, we must first ascertain whether it is in A or B. If it's in B, we'll only see its right son.
- If it's in A, we'll check to see if it's dominated by any of the points in A. (visit the left son).
- And if not then we check to see if it is dominated by any point in B (by searching the (k-1)-dimensional structure).

3) Complexity Analysis:

$$P(N, k) = 2P(N/2, k) + P(N, k - 1) + O(N)$$

$$S(N, k) = 2S(N/2, k) + S(N/2, k - 1) + O(1)$$

$$Q(N, k) = Q(N/2, k) + Q(N/2, k - 1) + O(1)$$

which have solutions

$$P(N, k) = O(N \lg^{(k-2)} N) \quad (8)$$

$$S(N, k) = O(N \lg^{(k-2)} N)$$

$$Q(N, k) = O(\lg^{(k-1)} N)$$

E. Range Searching

1) Problem Definition:

The problem is to build a structure holding N points in k -space to facilitate answering queries of the form "report all points which are dominated by point U and dominate point L ." This kind of query is usually called an orthogonal range query because we are giving each dimension i a range $R_i = [l_i, u_i]$ and then asking the search to report all points x such that x_i is in range R_i for all i . A geometric interpretation of the query is that we ask for all points in a given hyper-rectangle.

2) Algorithm for 1-d case:

The sorted array can be directly used. For the ranges, we only need to perform two binary searches. Query Complexity: $O(\lg N + F)$ if a total of F points are found to be in the region.

3) Algorithm for planar case:

- Recursively, we build a range tree containing the following entities in each node: HI and LO the minimum and maximum x coordinate values in the set S , the mid value MID, which is the line that divides the set into A and B , pointers to subtrees with set A and B , and a pointer to a sorted array containing the points of S ordered by y -value.
- We visit the root of the tree with the following recursive technique to answer a range query that asks for all points with x -values in range X and y -values in range Y .
- We compare the range X to the range $[LO, HI]$ when we visit node N . A range search in the array ordered on Y can be done for points $[LO, HI]$ contained in X . (all these points satisfy both the X and Y ranges). We only search the relevant subtree (recursively) when the X range is wholly on one side of MID; otherwise, we search both subtrees.

4) Complexity Analysis for planar case:

This structure's preprocessing and storage costs are both $O(N \lg N)$. To calculate the query cost, we observe that at each of the $\lg N$ levels of the tree, at most two sorted lists are searched, and each of those searches costs at most $O(\lg N)$ plus the number of points found during that search. The query cost of this structure is therefore $O(\lg^{(2)} N + F)$, where F is the number of points found in the desired range.

5) Algorithm in k-space:

- The range tree structure can of course be generalized to k -space. Each node in such a range tree contains pointers to two subtrees representing $N/2$ points in k- space and one N point subtree in $(k - 1)$ space
- When the dimension is 2, the sorted array will be found at the bottom of the range tree.

F. Complexity Analysis

- ❖ $P(N, k) = O(N \lg^{(k-1)} N)$
- ❖ $S(N, k) = O(N \lg^{(k-1)} N) - (9)$
- ❖ $Q(N, k) = O(\lg^{(k)} N + F)$

G. Fixed-Radius Near Neighbors

1) Problem Definition:

Given a sparse set of points S we need to report all the pairs which have distance less than d between them. We define sparsity as the condition that no d -ball in the space (that is, a sphere of radius d) contains more than some constant c points. This condition ensures that there will be no more than cN pairs of close points found.

2) Algorithm for one dimension:

- Sort the points into an ascending list, then scan down that list.
- We check backward and forth on the list a distance of d when visiting point x during the scan. By the sparsity condition, this involves checking at most c points for "closeness" to x . This technique costs $O(N \lg N)$ for sorting and then $O(N)$ for scanning, for a total cost of $O(N) (N \lg N)$.

3) Algorithm for planar case:

- First we select a good cut line L which satisfy the following
 - It is possible to locate L in linear time
 - The set S is divided approximately in half by L
 - Only $O(N^{1/2})$ points of S are within d of L .
- Now we can see who our closest neighbours are in A and B . We only need to seek for spots in the width $2d$ strip surrounding L in the marriage step. We project these points onto the line(remembering if each is from A or B) and this reduces to a one dimensional problem with $O(N^{1/2})$ points which can be solved in linear time.

4) Complexity Analysis:

$$T(N) = 2T(N/2) + O(N)$$

So the time complexity is $O(N \lg N)$

5) Algorithm and complexity in k-space:

It can be shown that for sparse point sets in k -space there will always exist good cut planes, which will have not more than $O(N^{1-1/k})$ points within d of them. These planes imply that the $(k-1)$ -dimensional subproblem can be solved in less than linear time, and the full problem thus obeys the recurrence

$$T(N, k) = 2T(N/2, k) + O(N)$$

This establishes that we can solve the general problem in $O(N \lg N)$ time.

H. Nearest Neighbours

1) Problem Definition:

For each point x the nearest point to x be identified


2) Algorithm for 2 dimension and complexity:

- The first step divides S into A and B and the second step finds for each point in A its nearest neighbor in A (and likewise for each point in B).
- The third stage must be completed by determining whether any point in A has a true nearest neighbour in B , and vice versa for points in B .
- Our algorithm's final phase projects all such points of A onto L , followed by every point of B onto L . It is thus possible to ascertain if any point x in A has a point in B that is closer to x than x 's nearest neighbour in A with a linear-time scan of the resultant list.
- This results in an $O(N \lg N)$ algorithm if presorting is used.

3) Algorithm for k dimension and complexity:

- The extension of the above proposed algorithm to k -space yield $O(N \lg^{(k-1)} N)$
- It's unclear whether this algorithm has a related search structure.
- It's still unclear whether a fast k -dimensional nearest neighbour search structure exists.

IV. MOTIVATION



The multidimensional divide-and-conquer strategy can be used to solve problems involving groups of objects in a multidimensional space. These dots could represent N cities in the plane (2-space) or N aeroplanes in 3-space in a geometric setup. Multivariate data with k variables measured on N samples is frequently viewed by statisticians as N points in k -space. Researchers in database systems use yet another interpretation, viewing N records containing k keys as points in a multidimensional space. On the practical side, many of the previously best-known methods have a running time proportional to N^2 for many of the situations described. The running time of the algorithms presented in this research is proportional to $N \lg N$ (at least for low dimensional spaces).

Main motivation to introduce a new paradigm comes from the the below benefits: -

- First, a coherent presentation enables descriptions of the algorithms to be communicated more easily.
- Second, by studying the algorithms as a group, advances made in one algorithm can be transferred to others in the group.
- Third, once the paradigm is understood, it can be used as a tool with which to attack unsolved research problems.

V. CHALLENGES IN THE WORK

The work's main objective is to create a common algorithmic structure for many challenges.

Although some of the theoretically elegant algorithms provided are not suitable for execution, they do offer a number of heuristic algorithms that are currently implemented in a number of software packages.

In terms of asymptotic running time, the methods proposed for domination and closest point problems are currently the best algorithms known for their respective issues. But in reality, for problems of practical size, the efficient methods of proposed solutions should be implemented.

The paradigm proposed has consciously applied to solve research problems. So here the challenge is, although all the research problems use the common proposed paradigm, there is no centralized scheme for further optimization of the problem. The nearest neighbor searching problem in k -space, which remains an open problem.

VI. LACKING IN THE Analysis

A. ECDF

The analysis considers only the case where N is a power of 2, and analyses were given for fixed k as N grows larger.

B. Maxima Finding

The presented recurrence solution for maxima finding is,

- $T(N, k) = 2T(N/2, k) + T(N, k - 1) + O(N)$ (10)

and by using the fact $T(N, 3) = O(N \lg N)$, the recurrence solution is solved and the final equation is,

- $T(N, k) = O(N \lg^{(k-2)} N)$ for $k \geq 3$ (11)

The above equation assumes that all N points of the original set are maxima of their subsets (Worst Case scenario), but there will be relatively few A and B maxima for many sets.

C. Dynamic structure

All of the data structures discussed in this analysis were static in the sense that they could not be expanded after they were constructed. However, many applications necessitate a dynamic structure into which new elements can be inserted.

VII. SCOPE FOR IMPROVEMENT

Developing methods to reduce the time complexity of proposed algorithms from $O(N \lg^{(k)} N)$ to $O(N \lg N)$.

A. ECDF Section IV.A can be improved by [1], where this difficulty can be overcome by the use of implementation-dependent constant c . The analysis also showed that the leading terms of the ECDF searching structures performances have similar coefficients i.e inverse factorials. The analysis showed that the time taken for Algorithm ECDF $_k$ is given by, $T(N, k) = c(N \lg^{k-1} N) / (k - 1)! + O(N \lg^{k-2} N)$ (12)

B. Maxima Finding: Section IV.B can be solved by [2], it is proved that only a very small number of points usually remain as maxima for many probability distributions. If only m points remain, then the term $T(N, k - 1)$ in the above recurrence is replaced by $T(m, k - 1)$, which for small enough m (i.e., $m = O$

(N^p) for some $p < 1$ has running time $O(N)$. If this is true, then the recurrence describing the maxima algorithm is

$$T(N, k) = 2T(N/2, k) + O(N) \quad (13)$$

which has solution $T(N) = O(N \lg N)$. The average running time of the algorithm is $O(N \lg N)$ for a wide class of distributions. [2] also presented a linear expected-time maxima algorithm (with poorer worst-case performance than this algorithm).

- C. Dynamic Structure Section VI.C problem can be solved by using the techniques described by [16] can be applied to all of the data structures that we have seen in this analysis to transform them from static to dynamic.

The cost of this transformation is that both query and preprocessing times will increase by an additional factor of $O(\lg N)$ ($P(N)$ now denotes the time required to insert N elements into an initially empty structure), while storage requirements will remain unchanged.

[10] and [14] can be applied to all of the data structures in this analysis to convert them to dynamic at the cost of an $O(\lg N)$ increase in $P(N)$, leaving both $Q(N)$ and $S(N)$ unchanged. Additionally, their method facilitates deletion of the elements.

VIII. RELATED WORK

[3] and [4] described a few basic paradigms similar to multidimensional divide and conquer that discuss a number of important analysis techniques in algorithm design.

[5] described a method for multidimensional searching that is radically different from one that we study (the proposed paradigm).

[6] and [7] have thoroughly investigated a large number of computational problems in plane geometry and have achieved many fascinating results.

Because it provides a good estimate of an underlying distribution given only a selection of randomly chosen points from that distribution, the ECDF is frequently used in statistical applications. This solves the problem of hypothesis testing and important multivariate tests require computing the allpoints ECDF problem (Hoeffding, multivariate KolmogorovSmirnov, and multivariate Cramer-Von Mises tests). Certain approaches to density estimation, which ask for an estimate of the underlying probability density function given a sample, require the solution to the ECDF searching issue. These and other applications of ECDF problems in statistics are described by [8].

[10] provided the complexity analysis and showed that $\theta(k N \lg N)$ time is necessary and sufficient for all the points in the k -space in the decision tree model of computation.

Range Searching Problem is used in querying a geographic database . In addition to database problems, range queries are also used in certain statistical applications. These applications and a survey of the different approaches to the problem are discussed in [11] survey of range searching.

The multidimensional divide-and-conquer technique has been applied to range searching problems by [12],[13],[14] who independently achieved structures very similar to the ones we proposed but with a different style of approach.

The details of Closest-Point problems and the proof of sparsity are explained clearly in [15].

IX. CRITICAL REVIEW

Section III Algorithms dealt with two basic classes of problems : all points problems and searching problems. For all-points problems of N points in k -space, algorithms running time of $O(N \lg^{(k-1)} N)$; for certain of these problems there were some extra techniques to reduce the running time even more . For searching problems data structures could be built on $O(N \lg^{(k-1)} N)$ time, used $O(N \lg^{(k-1)} N)$ space, and could be searched in $O(\lg^{(k)} N)$ time.

Both the all-points algorithms and the searching data structures were constructed by using one paradigm: multidimensional divide-and-conquer. All of the all-points problems have $\Omega(N \lg N)$ lower bounds and all of the searching problems have $\Omega(\lg N)$ lower bounds; the algorithms and data structures that are proposed are therefore within a constant factor of optimal (some for only small k , others for any k).

- At the first level of the analysis, a number of particular results of both theoretical and practical interest. The algorithms in Section-III are currently the best algorithms known for their respective problems in asymptotic running time.
- At the second level of the analysis, a particular algorithmic paradigm is introduced, multidimensional divide and conquer which has three distinct advantages.
- First, a large number of results in a nutshell were presented in the analysis. Second, the advances made in developing a research problem are used for other problems. Third, the paradigm proposed has been used to discover new algorithms and data structures.

REFERENCES

- [1] Monier, L. Combinatorial solutions of multidimensional divide- and conquer recurrences.
- [2] Bentley, J.L., Kung, H.T., Schkolnick, M., and Thompson, C.D. On the average number of maxima in a set of vectors and applications. *J. ACM* 25, 4 (Oct. 1978), 536-543.
- [3] Aho, AV., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
- [4] Weide, B. A survey of analysis techniques for discrete algorithms. *Computng. Surv.* 9, 4 (Dec. 1977), 291-313.
- [5] Dobkin, D., and Lipton, R.J. Multidimensional search problems. *SIAM J. Computng.* 5, 2 (June 1976), 181-186.
- [6] Shamos, M.I. *Computational geometry*. Unpublished Ph.D. dissertation, Yale Univ., New Haven, Conn., 1978.
- [7] Shamos, M.I. Geometric complexity. In *Proc. 7th ACM Symp. Theory of Computng.*, May 1975, pp. 224-233.
- [8] Bentley, J.L., and Shamos, M.I. A problem in multivariate statistics: Algorithm, data structure, and applications. In *Proc. 15th Allerton Conf. Communication, Control, and Computng.*, Sept. 1977, pp.193-201.
- [9] Jon Louis Bentley. 1980. Multidimensional divide-and-conquer. *Commun. ACM* 23, 4 (April 1980), 214-229.
- [10] 20. Lueker, G. A data structure for orthogonal range queries. In *Proc. 19th Symp. Foundations of Computr. Sci.*, Oct. 1978, pp. 28-34.
- [11] Bentley, J.L., and Friedman, J.H. Algorithms and data structures for range searching. *Computng. Surv.* 11, 4 (Dec. 1979), 397-409 .
- [12] Lee, D.T., and Wong, C.K. Qintary trees: A file structure for multidimensional database systems. To appear in *ACM Trans. Database Syst.*
- [13] Lueker, G. A data structure for orthogonal range queries. In *Proc. 19th Symp. Foundations of Computr. Sci.*, Oct. 1978, pp. 28-34.
- [14] Willard, D.E. *New data structures for orthogonal queries*. Harvard Aiken Computr. Lab. Rep., Cambridge, Mass., 1978.
- [15] Bentley, J.L. *Divide and conquer algorithms for closest point problems in multidimensional space*. Unpublished Ph.D. dissertation, Univ. of North Carolina, Chapel Hill, N.C., 1976.
- [16] Bentley, J.L. Decomposable searching problems. *Inform. Proc. Letters* 8, 5 (June 1979), 244-251.