Multi Dimensional Divide and Conquer

Sai Krishna Charan Dara (20171140) Venkata Susheel Voora (20171164)

Algorithms and Operating Systems - Project
International Institute of Information & Technology Hyderabad, India
sai.krishna@students.iiit.ac.in, venkata.susheel@students.iiit.ac.in

I. Introduction

The field of algorithm design and analysis has made many contributions to computer science of both theoretical and practical significance. One can cite a number of properly designed algorithms that save users thousands of dollars per month when compared to naive algorithms for the same task (sorting and Fourier transforms are examples of such tasks). On the more theoretical side, algorithm design has shown us a number of counter intuitive results that are fascinating from a purely mathematical viewpoint.

But the field,at the start, consisted primarily of a scattered collection of results, without much underlying theory. The author aims to design a common algorithmic structures which solve a pool of problems. This paper introduces a new paradigm in algorithmic solving - Multi Dimensional Divide and Conquer by discussing two major problems: Domination problems and Closest pair problems.

II. Problem Statement

The paper uses ***Multi Dimensional 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 discussed before are at one of two extremes, however: Either they are so general that they cannot be discussed precisely, or they are so specific that they are useful in solving only one or two problems. In this paper we examine a more "middle of the road" paradigm that can be precisely specified and yet can also be used to solve many problems in its domain of applicability. The paper introduces this paradigm as multidimensional divide-and-conquer.

Multidimensional divide-and-conquer is a single algorithmic paradigm that can be used to solve many particular problems. 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 paper we study a number of different algorithms and see how each can be viewed as an instance of multidimensional divide-and-conquer.

III. Algorithms Proposed in Paper

The author discusses the following problems to describe Multi-Dimensional 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 $rank(x)/N$. For this problem, we need to calculate 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 of S have x -value less than L 's (call this set of points A) and the remainder have greater x -value (call this set B). Note that L is a median x -value of the set.
- (Recursive Step) Recursively call ECDF2 (A) and $\mathrm{ECDF2}(B)$. 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 (remembering their type) and sort them by y -value. Scan through this sorted list in increasing y -value, keeping track in ACOUNT of the number of A s so far observed. Each time a B is observed, add the current value of ACOUNT to its partial ECDF.

3) Algorithm for solving ECDFk:

- Choose a $(k-1)$ dimensional cut plane P dividing S into two subsets A and B, each of $N/2$ points.
- Recursively call ECDF $k(A)$ and $\mathrm{ECDF}k(B)$. After this we know the true ECDF of all points in A.
- (For each B find the number of A s it dominates.) Project the points of S onto P, noting for each whether it was an A or a B. Now solve the reduced problem using a modified ECDF $(k-1)$ algorithm and add the calculated values to the partial ranks of B.

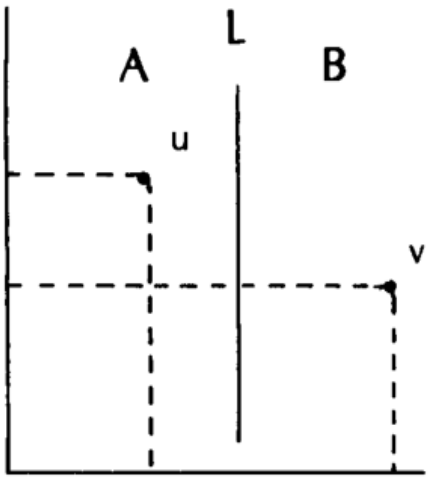

Fig. 1. Two cases of planar queries.

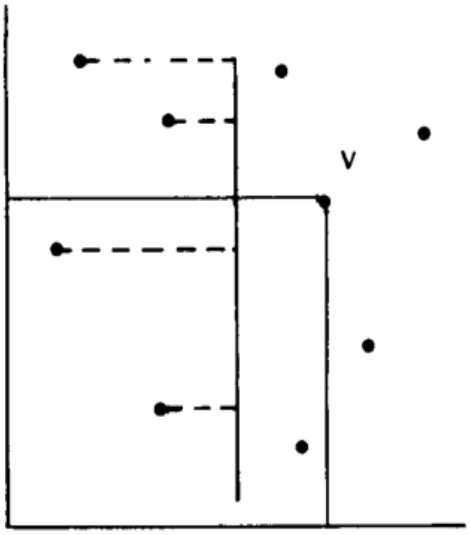

Fig. 2. Calculating v's y-rank in A.

4) Complexity Analysis: For ECDF2, by applying presorting technique,

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

For ECDFk,

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

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 establishes that

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

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 :

- By analogy to the all-points algorithm, we choose a line L dividing S into equal sized sets A and B. Instead of solving subproblems A and B, however, we now *recursively* process them into ECDF trees representing their respective subsets.
- Having built these subtrees we are (almost) prepared to answer ECDF queries in set S.
- The first step of a query algorithm compares the x-value of the query point with the line L; the two possible outcomes are illustrated in Figure 1 as points u and v.
- If the point lies to the left of L (as u does), then we find its rank in S by recursively searching the substructure rep- resenting A, for it cannot dominate any point in B.
- If the point lies to the right of L (as v does), then searching B tells how many points in B are dominated by v, but we still must find how many points in A are dominated by v. To do this we need only calculate v's y-rank in A; this is illustrated in Figure 2.

3) Complexity Analysis: There are three costs associated with a search structure: the *preprocessing* time required to build the structure, the *query* time required to search a structure, and the *storage* required to represent 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:

$$P(N) = 2P(N/2) + O(N)$$
$$P(N) = O(N \lg N) \tag{3}$$

$$S(N) = 2S(N/2) + N/2)$$
$$S(N) = O(N \lg N) \tag{4}$$

$$Q(N) = Q(N/2) + O(\lg N)$$
$$Q(N) = O\left(\lg^2 N\right) \tag{5}$$

For ECDF-Searching dimension k:

$$P(N,k) = 2P(N/2,k) + P(N/2,k-1) + O(N)$$
$$S(N,k) = 2S(N/2,k) + S(N/2,k-1) + O(1)$$
$$Q(N,k) = Q(N/2,k) + Q(N/2,k-1) + O(1)$$
$$P(N,k) = O\left(N \lg^{k-1} N\right) \tag{6}$$
$$S(N,k) = O\left(N \lg^{k-1} N\right)$$
$$Q(N,k) = O\left(\lg^k N\right)$$

C. Maxima Finding

1) Problem Definition:: A point is said to be 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.
- Since every maxima of B is a maxima of the whole set, the third step must discard every maxima of B. This is accomplished by projecting the respective maxima sets onto the plane and then solving the planar problem. We could modify the two-dimensional maxima algorithm to solve this task, but it will be slightly more efficient to use the scanning algorithm.
- Suppose we cut into A and B by the z-coordinate; we must discard all A's dominated by any B's in the x-y plane. If we have presorted by x, then we just scan right to left down the sorted list, discarding A's with y-values less thatn the maximum B y-values observed to date. This marriage step will have linear time, so 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:

$$\begin{aligned} &T(N,k) = 2T(N/2,k) + T(N,k-1) + O(N) \\ &\text{and we can use the fact that } T(N,3) = O(N \lg N) \\ &\text{to establish that,} \\ &T(N,k) = O\left(N \lg^{k-2} N\right) for k \geq 3 \end{aligned} \tag{7}$$

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 contains two substructures representing N/2 points in k-space and one substructure representing N/2 points in $(k-1)$- space.
- To test if a new point is a maximum we first determine if it lies in A or B. If it is in B, then we visit only the right son.
- If it lies in A, we first see if it is dominated by any point 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:

$$\begin{aligned} &P(N,k) = 2P(N/2,k) + P(N,k-1) + O(N) \\ &S(N,k) = 2S(N/2,k) + S(N/2,k-1) + O(1) \\ &Q(N,k) = Q(N/2,k) + Q(N/2,k-1) + O(1) \\ &\text{which have solutions} \\ &P(N,k) = O\left(N \lg^{k-2} N\right) \\ &S(N,k) = O\left(N \lg^{k-2} N\right) \\ &Q(N,k) = O\left(\lg^{k-1} N\right) \end{aligned} \tag{8}$$

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 and dominate point L." This kind of query is usually called an orthogonal range query because we are in fact giving for 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 are asking for all points that lie in a given hyper-rectangle.

2) Algorithm for 1-d case: The sorted array can be directly used. We just need to do two binary searches for the ranges. Query Complexity: $O(\lg N + F)$ if a total of F points are found to be in the region.

3) Algorithm for planar case:

- We construct a range tree recursively with the following entities in the node: LO and HI the min and max x coordinate value in the set S, the mid value MID which is the line that divides the set in A and B,pointers to subtrees having set as A and B respectively, a pointer to a sorted array, containing the points of S sorted by y-value.
- We answer a range query asking for all points with x-value in range X and y-value in range Y by visiting the root of the tree with the following recursive proce- dure.
- When visiting node N we compare the range X to the range [LO, HI]. If [LO, HI] is contained in X, then we can do a range search in the sorted array for all points in the range Y (all these points satisfy both the X and Y ranges). If the X range lies wholly to one side of MID, then we search only the appropriate subtree (recursively); otherwise we search both subtrees.

4) Complexity Analysis for planar case: The preprocessing costs of this structure and the storage costs are both $O(N \lg N)$. To analyze the query cost we note that at most two sorted lists are searched at each of the lg N levels of the tree, and each of those searches cost at most $O(\lg N)$, plus the number of points found during that search. The query cost of this structure is therefore $O\left(\lg^2 N + F\right)$, 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.

- The range tree at the bottom when the dimension is 2 will contain the sorted array described before.

F. Complexity Analysis

$$\begin{aligned} P(N,k) &= O\left(N \lg^{k-1} N\right) \\ S(N,k) &= O\left(N \lg^{k-1} N\right) \\ Q(N,k) &= O\left(\lg^{k} N + F\right) \end{aligned} \tag{9}$$

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 a list in ascending order and then scan down that list.
- When visiting point x during the scan we check backward and forward on the list a distance of d. By the sparsity condition, this involves checking at most c points for "closeness" to x. The cost of this procedure is $O(N \lg N)$ for the sorting and then $O(N)$ for the scan, for a total cost of $O(N \lg N)$.

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 $O\left(N^{1/2}\right)$ points of S are within d of L .
- Now we get the near neighbours in A and B. In the marriage step we just have to look for the points in the width 2d strip around L. We project these points on to the line(remembering if each is from A or B) and this reduces to a one dimensional problem with $O\left(N^{1/2}\right)$ points which can be solved in linear time.

4) Complexity Analysis: T(N)=2 T(N / 2)+O(N)

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

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 $O\left(N^{1-1/k}\right)$ 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 step must *patch up* by finding if any point in A actually has its true nearest neighbor in B, and similarly for points in B.
- The final step of our algorithm projects all such points of A onto L and then projects every point of B onto L. It is then possible to determine during a linear-time scan of the resulting list if any point x in A has a point in B nearer to x than x's nearest neighbor in A.
- 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 $O\left(N \lg^{k-1} N\right)$.
- It is not clear that there is a search structure corresponding to this algorithm.
- Whether there exists a fast k-dimensional nearest neighbor search structure is still an open question.

IV. Motivation

Multidimensional divide-and-conquer is applicable to problems dealing with collections of objects in a multidimensional space. In a geometric setting these points might represent N cities in the plane (2-space) or N airplanes in 3-space. Statisticians often view multivariate data with k variables measured on N samples as N points in k-space. Yet another interpretation is used by researchers in database systems who view N records each containing k keys as points in a multidimensional space. On the practical side, the previous best-known algorithms for many of the problems discussed have running time proportional to N^2. The algorithms discussed in this paper have running time 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 key challenge in the work is to design a common algorithmic structure for different problems.

Some of the theoretically elegant algorithms proposed are not suitable for implementation, but the advantage is, they suggest certain heuristic algorithms which are currently implemented in several software packages.

Algorithms proposed for domination and closest point problems are currently the best algorithms known for their

respective problems in terms of asymptotic running time. But in reality, for problem of practical size, the efficient methods of proposed solutions should be implemented.

The paradigm the author proposed has consciously applied to solve research problems. In addition to the paradigm he proposed, he also added extra optimizations to get the fastest result. So here the challenge is, although all the research problems uses the common proposed paradigm, there is no centralized scheme for further optimization of the problem.

The author didn't solve the the nearest neighbor searching problem in k-space, which remains as open problem.

VI. Lacking in the Paper

A. ECDF

The paper 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,

$$T(N,k) = 2T(N/2,k) + T(N,k-1) + O(N) \tag{10}$$

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

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

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

C. Dynamic structure

All of the data structures described in this paper have been static in the sense that once they are built, additional elements cannot be inserted into them. Many applications, however, require a dynamic structure into which additional elements can be inserted.

VII. Scope for improvement

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

A. ECDF

Section IV.A can be improved by [1], where he overcomes this difficulty by the use of implementation-dependent constant c. His analysis also showed that the leading terms of the ECDF searching structures performances have similar coefficients i.e inverse factorials.

His analysis showed that the time taken for Algorithm ECDFk is given by,

$$T(N,k) = c\left(N \lg^{k-1} N\right)/(k-1)! + O\left(N \lg^{k-2} N\right) \tag{12}$$

B. Maxima Finding

Section IV.B can be solved by [2], he 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., $m = O\left(N^p\right)$ for some $p < 1$) has running time $O(N)$. If this is true, then the recurrence describing the maxima algorithm is

$$T(N,k) = 2T(N/2,k) + O(N) \tag{13}$$

which has solution $T(N) = O(NlogN)$. He showed that the average running time of the algorithm is $O(NlogN)$ for a wide class of distributions.

[2] also presented 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 paper to transform them from static to dynamic.

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

Recent work by [10] and [14] can be applied to all of the data structures in this paper 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 discusses 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] had thoroughly investigated a large number of computational problems in plane geometry and has achieved many fascinating results.

The ECDF is often required in statistical applications because it provides a good estimate of an underlying distribution, given only a set of points chosen randomly from that distribution. This solves the problem of hypothesis testing and important multivariate tests require computing the all-points ECDF problem (Hoeffding, multivariate Kolmogorov-Smirnov, and multivariate Cramer-Von Mises tests).The solution to the ECDF searching problem is required for certain approaches to density estimation, which asks for an estimate of the underlying probability density function given a sample. These and other applications of ECDF problems in statistics are described by [8].

[10] provided the complexity analysis and showed that $\theta(kN \lg N)$ 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 problem 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 in 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 $O\left(N \lg^{k-1} N\right)$; for certain of these problems there were some extra techniques to reduce the running time even more . For searching problems the author developed data structures that could be built on $O\left(N \lg^{k-1} N\right)$ time, used $O\left(N \lg^{k-1} N\right)$ space, and could be searched in $O\left(\lg^{k} N\right)$ 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 paper, the author presented 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 paper, the author introduces particular algorithmic paradigm, multidimensional divide and conquer which has three distinct advantages.

First, the author presented a large number of results in a nutshell. Second, the advances made in developing a research problem is used for other problem. 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. Comptng. Surv. 9, 4 (Dec. 1977), 291-313.

[5] Dobkin, D., and Lipton, R.J. Multidimensional search problems. SIAM J. Comptng. 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 Comptng., 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 Comptng., 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 Comptr. Sci., Oct. 1978, pp. 28-34.

[11] Bentley, J.L., and Friedman, J.H. Algorithms and data structures for range searching. Comptng. 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 Comptr. Sci., Oct. 1978, pp. 28-34.

[14] Willard, D.E. New data structures for orthogonal queries. Harvard Aiken Comptr. 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.

[17]