INTERDISTANCE SELECTION BY PARAMETRIC SEARCH

Jeffrey S. Salowe

Computer Science Report No. TR-87-20
November 13, 1987

INTERDISTANCE SELECTION BY PARAMETRIC SEARCH

Jeffrey S. Salowe

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903
November 12, 1987

ABSTRACT

The selection problem asks for the k^{th} largest or smallest element in a set S. In general, selection takes linear time, but if the set is constrained so that some relations between elements are known, sublinear time selection is sometimes possible. Chazelle [5] described one technique for selection when the set is constrained by geometry. This paper demonstrates a new technique for geometric selection problems based on the idea of parametric search [6, 14, 16] and applies it to show that given n points in $\Re^d$, the k^{th} largest L_∞ interdistance can be selected in $O(d\,n \log^d n)$ time. This is the first selection algorithm for multidimensional interdistances to run in $O(n \log^{O(1)} n)$ time.

KEYWORDS

Selection, interdistance, Computational Geometry.

INTRODUCTION

Given a totally ordered set S of n items, the rank of element $s \in S$ is $\rho_s = |\, v : v \leq s \,|$. For inputs S and s, the ranking problem is to calculate ρ_s, and for inputs S and k, the selection problem is to find an element $s \in S$ with $\rho_s = k$ (the k^{th} order statistic). This paper makes two contributions to the understanding of selection problems; first, it describes a new, general method for selecting elements from a geometric set in $o(|S|)$ time, and second, it applies this technique to produce a $O(d\,n \log^d n)$ time algorithm to select the k^{th} largest L_∞ interdistance determined by n points in $\Re^d$. This is the first selection algorithm for multidimensional interdistances with respect to any metric to run in $O(n \log^{O(1)} n)$ time.

It is well known that the k^{th} largest or smallest element in any set S can be found in linear time [3]. However, if there are constraints placed on set S, selection in sublinear time may be possible. The first result of this kind seems to be that of Shamos, Jefferson and Tarjan in [19] on selecting the median of a Cartesian sum; this result was followed by a plethora of results, including [9-12, 15]. In [5], Chazelle specifically examined the selection problem for geometrically constrained sets S. He proposed an approach for designing sublinear selection algorithms and applied this method to the problems of finding the k^{th} longest bridge length between two polygons, finding the k^{th} longest interdistance determined by n points along an analytic curve, finding the k^{th} largest area triangle determined by n points in the plane and finding the k^{th} longest Euclidean interdistance in $\Re^d$.

Nevertheless, there are several interesting geometric problems that resist Chazelle's method; that is, his technique is not a substantial improvement over the trivial method of first enumerating the set and then using the linear time selection algorithm. In particular, there is no truly efficient method to select Euclidean interdistances in dimension $d \geq 2$, with the notable exception of finding the smallest interdistance. We show that there is a good general selection algorithm for L_∞ interdistances which uses a new technique for geometric selection problems based on the idea of parametric search [6, 14, 16]. To put this result in perspective, the known results for selecting median interdistances are given in Figure 1; the time bounds proved in this paper are marked with an asterisk. We note that the parametric search technique for selection was developed by [7] to produce an optimal time algorithm for the problem of selecting the k^{th} largest sloped line passing through 2 of n points in the plane.

PARAMETRIC SEARCH

We begin by briefly sketching Chazelle's result. The papers of [9, 10] describe sublinear time selection algorithms when the set S is given in the form of a structured matrix. In the former paper, this matrix has sorted columns, and in the latter paper, the matrix has both sorted rows and sorted columns. Chazelle noted that many geometric relationships have an embedded ordering so that the relations can be mapped onto a small collection of structured matrices in $o(|S|)$ time. His technique consists of a sublinear time transformation followed by an established sublinear time selection algorithm. Although it is effective for many problems, it may not work for other problems because of the difficulty in reorganizing the set of objects into a small collection of structured matrices.

Interdistance Selection Results				
norm (p)	dim (d)	Bound (median)	Proven Optimal?	Reference
-	1	$O(n \log n)$	Yes	[9, 19], *
1	2	$O(n \log^2 n)$	No	*
1	$d>2$	$o(n^2)$	No	[18]
2	$d \geq 2$	$O(n^{2-2^{-(d+1)}} \log^{1-2^{-(d+1)}} n)$	No	[5]
∞	$d \geq 2$	$O(d\, n \log^d n)$	No	*

Figure 1 — General Interdistance Selection Results

Our approach to selection algorithms is based on an ingenious technique of using parallel algorithms in the design of efficient sequential algorithms [6, 16] which in turn is a refinement of the method of parametric search presented in [14]. Given a search problem **A**, a parametric search algorithm is one that finds a particular item s^* contained in the search space S_A using another algorithm **B** to guide the search.

In fact, if problem **A** meets certain conditions and there is a good parallel algorithm for problem **B**, parametric search may be accelerated. The first step to do this is to simulate the parallel algorithm in a sequential fashion; that is, perform all the operations on the first level in any order, then the operations on the second level, and so on. The impetus for this approach is the following. Suppose there are algorithms which use some basic operation to guide the search for $s^* \in S_A$. Although each operation may be costly, it is often the case that operations are related so that after performing one operation expensively, many others can be resolved cheaply. It is precisely in this circumstance that the independence of operations in a parallel algorithm is useful. On each level of the parallel algorithm, choose the operation which affects the greatest number of the remaining ones, partition the set of operations into those that are consequently resolved and those that are not, and then recurse on the the set of operations that are not resolved. This has the effect of drastically reducing the number of expensive operations performed.

How does parametric search relate to geometric selection problems? For several geometric problems, it is easier to rank than to select. Let the set of geometric objects in problem **A** be the search space S_A, assume that any element in S_A can be ranked, and let $s^* \in S_A$ be an element with $\rho_{s^*} = k$. Our selection algorithm searches for element s^* by ranking elements in $S' \subseteq S_A$. Problem **B** is chosen so that its operations are solved by ranking and it implicitly ranks only a small subset of S_A which must contain the k^{th} element. Since parametric search does not subdivide the geometric set, it is well suited to solving certain geometric selection problems. These general remarks will be made clearer when they are tied down to a specific example in the next section.

A NEW OPTIMAL ONE-DIMENSIONAL INTERDISTANCE SELECTION ALGORITHM

Given a set $X = \{x_1, ..., x_n\}$, the set of one-dimensional interdistances is

$$S = \{\ |x_i - x_j| : i < j\ \},$$

and the one-dimensional interdistance selection problem is to find the k^{th} largest or smallest element in S. To facilitate the explanation, assume that the elements in S are distinct, though this restriction can be removed without affecting the asymptotic time complexities of the algorithms. An optimal algorithm for this selection problem results from a theorem of [9] on selecting Cartesian sums. The Cartesian sum of n vector Y and m vector Z is the set

$$Y + Z = \{\ y_i + z_j : 1 \le i \le n,\ 1 \le j \le m\ \},$$

and a Cartesian sum is clearly a constrained set since $m+n$ inputs determine the mn elements in $Y+Z$. The theorem of Frederickson and Johnson states that the k^{th} largest or smallest element in $Y+Z$ may be selected in time $O(m + p \log k/p)$, $p = \min\{k, m\}$, which is optimal in the comparison tree model of computation. Since the set S of one-dimensional interdistances is the largest $\left[\begin{matrix} n \\ 2 \end{matrix}\right]$ elements in $X + -X$, the Cartesian sum algorithm can be used to give a selection algorithm for one-dimensional interdistances with the same time complexity. Note that finding the smallest interdistance requires $\Omega(n \log n)$ time in the linear decision tree model [8] and that finding the median interdistance requires $\Omega(n \log n)$ time in the comparison tree model [19]. (See [17] for more detail on the first lower bound.)

In this section, the technique of parametric search is applied to produce a new optimal algorithm for one-dimensional interdistances. It is optimal in the sense that for the worst possible k, it gives the best possible running time, but it does not give the fastest algorithm for k in the neighborhood of $k = \binom{n}{2}$. This result will be used to construct the algorithm for L_∞ selection.

The main theorem of this section is:

Theorem 1: Given a set X of n points in $\Re$, let S be the set of one-dimensional interdistances generated by X. Then the k^{th} one-dimensional interdistance in S can be selected in $O(n \log n)$ time.

This theorem is proved in two parts. In the first part, we show how to rank an interdistance in linear time after an initial $O(n \log n)$ preprocessing cost. In the second part of the proof, this result is incorporated into a parallel merging algorithm to find the k^{th} interdistance.

Theorem 2: Given a sorted set X of n numbers $x_i \in \Re$, distance r can be ranked among the $\binom{n}{2}$ interdistances in S in $O(n)$ time.

Proof: By assumption, the input points $\{x_1,...,x_n\}$ are sorted. Ranking interdistance r is equivalent to counting the number of points within distance r of each x_i, summing these counts and dividing by two. To do this, we add the total number of points between $(x_i,x_i+r]$ to the total number of points between $[x_i-r,x_i)$ for each i. Algorithm *Right–Sum* counts the number of points between the intervals $(x_i,x_i+r]$, and a similar algorithm which is not presented, *Left–Sum*, counts the number of points between the intervals $[x_i-r,x_i)$ (see Figure 2).

Algorithm *Right–Sum* merges X with the set $\{x_i + r : x_i \in X\}$. It sweeps from the leftmost point to the rightmost point, keeping track of two parameters, *rank* and *count*. *Rank* is the subtotal of the input points within the intervals determined so far, and *count* is the number of input points (i.e. centers of intervals) between x_i and $x_i + r$. At each input point, *count* is increased by 1, since there is an additional center between x_i and $x_i + r$. At each right endpoint, *rank* is incremented by the current value of *count* and then *count* is decreased by 1 because this right endpoint matches some input point. It is easy to prove by induction that these parameters are correct and that they accurately count the number of points between the intervals. □

```
ALGORITHM RIGHT-SUM ($\{x_1,\ldots,x_n\}, r$)
    FOR $i=1$ TO $n$ DO
        $L_1[i] \leftarrow x_i$
        $tag(L_1[i]) \leftarrow center$
        $L_2[i] \leftarrow x_i + r$
        $tag(L_2[i]) \leftarrow right$
    $L \leftarrow Merge(L_1, L_2)$
    $rank \leftarrow 0$
    $count \leftarrow -1$
    $i \leftarrow 1$

    REPEAT
        WHILE $tag(L[i]) = center$ DO
            $count \leftarrow count + 1$
            $i \leftarrow i + 1$
        /* $L[i]$ is $right$
        $rank \leftarrow rank + count$
        $count \leftarrow count - 1$
        $i \leftarrow i + 1$
        IF $i > 2n$ RETURN ($rank$)
    FOREVER
```

Figure 2 — ALGORITHM *Right–Sum*

In the second part of the proof, a merging algorithm is used to guide the search for the k^{th} interdistance. To see the tie with merging, define a one-dimensional ball, $B(x_i, r)$, with center x_i and radius r to be a closed interval of length $2r$ centered at x_i. For any ball $B(x_i, r)$, there is a left side $B_l(x_i, r)$ at $x_i - r$ and a right side $B_r(x_i, r)$ at $x_i + r$. Given r, consider the sorted set $\cup_{1 \le i \le n}(B_l(x_i, r) \cup x_i)$. When r is equal to the k^{th} largest interdistance, r^*, $\sum_{1 \le i \le n} |B(x_i, r^*) \cap X| = 2k$ and there is an associated ordering of $\cup_{1 \le i \le n}(B_l(x_i, r^*) \cup x_i)$. When r is any other interdistance, $\sum_{1 \le i \le n} |B(x_i, r) \cap X| \ne 2k$ and the ordering of $\cup_{1 \le i \le n}(B_l(x_i, r) \cup x_i)$ differs from the ordering for r^*. No two interdistances are associated with the same ordering.

We begin the process of determining this ordering of left endpoints and input points for r^* by sorting the input points. As a consequence, the set $\{B_l(x_i, r) : x_i \in X\}$ is also sorted for any $r > 0$. The guiding algorithm merges the sorted list of $\{B_l(x_i, r^*) : x_i \in X\}$ with the sorted inputs X for an unknown interdistance r^* with rank k. Though r^* is not known, the outcome of comparison "Is $B_l(x_i, r^*)$ less than x_j?" for some $x_i > x_j$ can be resolved by ranking interdistance $r_{ij} = |x_i - x_j|$. If $\rho_{r_{ij}} > k$, then $r_{ij} > r^*$ and $x_j < B_l(x_i, r^*)$, and if $\rho_{r_{ij}} < k$, then $r_{ij} < r^*$ and

$x_j > B_l(x_i, r^*)$. Otherwise, $r_{ij} = r^*$ so r_{ij} is the k^{th} interdistance.

Lemma 1: Any algorithm that merges by comparisons the set $\{B_l(x_k, r^*) : x_k \in X\}$, $r^* = x_i - x_j$, with X must compare x_i with x_j.

Proof: In the ordering of the left endpoints and the input points with respect to r^*, $B_l(x_i, r^*)$ and x_j must be adjacent since they coincide. Therefore, the result of every other comparison is the same regardless of whether $B_l(x_i, r^*) < x_j$, $B_l(x_i, r^*) = x_j$ or $B_l(x_i, r^*) > x_j$. If the merging algorithm does not compare $B_l(x_i, r^*)$ with x_j, then the sorted order of these two points is never determined, a contradiction. □

Proof of Theorem 1: As a result of Lemma 1, any algorithm that merges by comparison will eventually compare the points determining the k^{th} interdistance. If each of these comparisons is done by a full ranking, this parametrically guided selection algorithm takes $O(n) \cdot O(n) + O(n \log n) = O(n^2)$ time. However, many comparisons need not be resolved by ranking. Each comparison C_{ij} asks "Is $B_l(x_i, r^*)$ less than x_j?" for some $x_i > x_j$ and may be identified with an interdistance r_{ij}; if $r_{ij} < r^*$ is discovered by ranking in linear time, then all comparisons C_{st} for which $r_{st} < r_{ij}$ can be resolved in <u>constant time</u> each. If $r_{ij} > r^*$, then all C_{st} for which $r_{st} > r_{ij}$ can be resolved. It is said that the comparisons have an ordering property. As a result, comparisons are either resolved by ranking (expensive) or by an appeal to transitivity (cheap).

To minimize the number of expensive rankings, a sequentialized version of the parallel merging algorithm of [4, 20] is used to find the desired r^*. [20] showed that two lists of size n can be merged using $O(\mathit{loglog}\ n)$ levels of $O(n)$ comparisons each, and these bounds also hold when overheads are counted [4]. In effect, the merging algorithm can be run sequentially in $O(n\ \mathit{loglog}\ n)$ time, and it ends once the $O(n\ \mathit{loglog}\ n)$ comparisons are resolved.

With respect to a level of the sequentialized parallel algorithm, the comparisons may be performed in any order. A reasonable strategy to reduce the number of expensive comparisons is to find the median comparison on a given level and rank it in linear time. As a result, half of the comparisons on that level are resolved in constant time each. The set of comparisons may be partitioned into those that are resolved and those that are not, and the latter set is resolved recursively. On each level, only $O(\log n)$ comparisons have been ranked rather than $O(n)$ comparisons, and transitivity is used to resolve the remainder. The cost of this algorithm is

$$O(n \, loglog \, n) + O(n) \cdot O(log \, n) \cdot O(loglog \, n) = O(n \log n \, loglog \, n),$$

a significant improvement over $O(n^2)$. The first term in the sum accounts for the overheads and comparisons resolved by transitivity, and the second term is the cost of ranked comparisons.

In fact, using the strategy of Cole [6] of weighting comparisons based on their level and then determining the weighted median comparison over all levels, only $O(log \, n)$ interdistances need to be ranked. To do this, let an "active" comparison be an unresolved comparison whose inputs are known. Assign an active comparison at level j weight 4^{-j}, starting with level 0. With respect to the set of active comparisons, resolve the weighted median. Cole proves that there are no active comparisons on level j after $5(j + \frac{1}{2} \log n)$ resolutions, so our algorithm must finish after $O(loglog \, n + \log n)$ interdistances are ranked. By Theorem 2, each ranking is done at a cost of $O(n)$ time after a single sorting step, so the total cost of selection is $O(n \log n)$. An argument similar to the proof of Lemma 1 and the ordering property of comparisons prove that the actual comparison corresponding to r^* must be ranked. □

A sequentialized version of the AKS sorting network [1] could have been used to guide the search in the same time bound, but the multiplicative factor is much larger than the one for the parallel merging algorithm.

L_∞ INTERDISTANCE SELECTION

The L_∞ metric is a member of the family of L_p distance functions. For $p \geq 1$ and $d > 1$, the $\underline{L_p \text{ distance}}$ between points $x_i = (x_{i_1}, \ldots, x_{i_d})$ and $x_j = (x_{j_1}, \ldots, x_{j_d})$ is defined as

$$\|x_i - x_j\|_p = (\sum_{k=1}^{d} |x_{i_k} - x_{j_k}|^p)^{1/p}.$$

The limit of $\|x_i - x_j\|_p$ as $p \rightarrow \infty$ is the L_∞ metric and may be shown to be

$$\|x_i - x_j\|_\infty = \max_{1 \leq k \leq d} |x_{i_k} - x_{j_k}|.$$

When $p = 2$, $\|x_i - x_j\|_2$ is the usual Euclidean distance between two points.

For the purpose of selection, we define "unit balls" corresponding to each L_p metric. The unit ball for L_p centered at the origin, $U(p,d)$, is defined as

$$U(p,d) = \{x : \|x\|_p = 1, x \in \Re^d\}.$$

Geometrically, $U(1,d)$ is a d-octahedron, $U(2,d)$ is a hypersphere, and $U(\infty,d)$ is a d-cube, also denoted by H^d.

The remaining L_p unit balls may be visualized by continuously deforming the d-octahedron to the hypersphere to the hypercube H^d. As these objects deform, $U(p,d)$ is completely enclosed by $U(p+\varepsilon,d)$, but the unit basis vectors along with their mirror images remain fixed.

The known bounds on L_∞ selection are: (1) The lower bounds for the one-dimensional problem also hold for L_∞ since any one-dimensional problem can be embedded into an L_∞ problem of arbitrary dimension. (2) The smallest L_∞ interdistance can be computed in asymptotically optimal $O(n \log n)$ time using the divide and conquer algorithm of [2]. (3) The largest interdistance can be found in $O(dn)$ time by determining the maximum difference between the points coordinatewise, and (4) no $o(n^2)$ time upper bounds for the general selection problem are explicitly stated in the literature, but it is possible to extend Chazelle's L_2 interdistance algorithm to select L_∞ interdistances in $O(n^{2-2^{-(d+1)}} \log^{1-2^{-(d+1)}} n)$ time.

We now improve the upper bound on general L_∞ selection. Let $X = \{x_1, \ldots, x_n\}$ be a set of n points in $\Re^d$, let S^d be the set of L_∞ interdistances generated by X, and assume that the elements in S^d are distinct. Removing this assumption does not adversely affect the asymptotic complexity of the algorithm. Define $H^d(x_i, r)$ to be a d-cube of side $2r$ centered at input point x_i. If r is an interdistance, there is a d-cube, say $H^d(x_i, r)$, which touches point x_j, and another d-cube, $H^d(x_j, r)$, which touches x_i. This corresponds to the notion described in the previous section where at an interdistance, one endpoint of a ball touched an input point. In the present case, there are $2d$ different sides in $\Re^d$ rather than just 2, leading to d projected subproblems to be solved by the merging strategy.

Theorem 3: Given a set X of n points in $\Re^d$, the k^{th} largest L_∞ interdistance in S^d can be selected in $O(d\, n \log^d n)$ time.

Again, there are two parts to the proof. The first part is to determine the complexity of ranking and the second part is to organize the parametric search. The principal building block in L_∞ interdistance ranking is the observation that its unit ball is a hypercube, hence the points within a specified L_∞ distance of $x_i \in \Re^d$ are contained in a hypercube centered at x_i. The problem of counting the number of points within an arbitrary hyperrectangle is known as the orthogonal range counting problem for which [21] and [13] independently devised the layered range tree with optimal space and query time (see [17]):

Theorem 4: [13,21] Orthogonal range counting of n points in $\Re^d$, $d \geq 2$, can be effected by an algorithm which uses $O(n \log^{d-1} n)$ preprocessing time, $O(n \log^{d-1} n)$ space and $O(\log^{d-1} n)$ time queries.

To apply this result to ranking, we have

Lemma 2: Let X be a set of n points in $\Re^d$. Distance r may be ranked in S^d by n orthogonal range counts.

Proof of Lemma 2: Suppose distance r (not necessarily an interdistance) is to be ranked; that is, given r, find ρ_r. For every $x_i \in X$, the points for which $\|x_i - x_j\|_\infty \le r, j \ne i$, are inside the d-cube of side $2r$ centered at x_i. The rank of r is determined by summing the orthogonal range counts and then dividing the result by 2 to compensate for double-counting. □

Along with Theorem 4, this proves

Theorem 5: Let X be a set of n points in $\Re^d$, $d \le 2$. Distance r can be ranked in S^d in time $O(n \log^{d-1} n)$.

Proof of Theorem 3: We are now in a position to prove Theorem 3, the main result of the paper. The selection algorithm is parametrically driven by a collection of d one-dimensional merging problems. In the s^{th} such problem, input points $x_i \in X$ are projected onto the s^{th} coordinate axis by setting all but the s^{th} coordinate, x_{i_s}, to zero; these projected points are then sorted. At interdistance r_{ij}, the projection of the boundary of $H^d(x_i, r_{ij})$ is precisely $B(x_{i_s}, r_{ij})$. The s^{th} merging problem is to determine for some unknown interdistance r^* the ordering of left endpoints and input points in each of the projected one-dimensional problems, where $\rho_{r^*} = k$.

To be more specific, Cole's technique is used in each of the d subproblems to choose a set of $O(d \log n)$ L_∞ distances (not necessarily interdistances) to rank. The one-dimensional merging scheme is applied to this problem, except that a comparison of an endpoint with an input point is resolved through the L_∞ ranking of r, not the one-dimensional ranking. In the s^{th} subproblem, a comparison of $B_l(x_{i_s}, r^*)$ with x_{j_s}, $x_{j_s} < x_{i_s}$, asks "Is $B_l(x_{i_s}, r^*)$ less than x_{j_s}?" and is resolved by ranking $r_{ij_s} = |x_{i_s} - x_{j_s}|$ in S^d. If $\rho_{r_{ij_s}} > k$, then $r_{ij_s} > r^*$ and $B_l(x_{i_s}, r^*) < x_{j_s}$, if $\rho_{r_{ij_s}} < k$, then $r_{ij_s} < r^*$ and $B_l(x_{i_s}, r^*) > x_{j_s}$, and if $\rho_{r_{ij_s}} = k$, then $r_{ij_s} = r^*$. It is clear that any pair of comparisons can be ordered by the distances they correspond to. Furthermore, in at least one of the projected subproblems, the k^{th} largest interdistance r^* is realized in that $B_l(x_{i_s}, r^*) = x_{j_s}$ corresponds to $H^d(x_i, r^*)$ touching x_j. This interdistance must be ranked due to the same reasons given in the proof of Lemma 1.

Since the ranking procedure takes $O(n \log^{d-1} n)$ time and each of d subproblems requires $O(log\ n)$ rankings, $O(d\,n \log^d n)$ time is needed in total. □

CONCLUSIONS AND OPEN PROBLEMS

The selection technique presented in this paper is effective when there is some transformation from the selection problem to merging or sorting. Good problems include distance functions whose unit balls are polytopes as well as problems dealing with vertices in arrangements with respect to some direction[18]. A speed-up is possible in the latter problem via approximate rankings.

The first interesting open problem is to speed-up L_∞ selection. The difficulty here is that L_∞ ranking is far more complicated than approximating inversions. A second open problem is to select Euclidean interdistances in $O(n \log^{O(1)} n)$ time. As mentioned in the introduction, the best results for this problem are almost quadratic [5] and are related to an ingenious method of [22] for constructing Euclidean minimum spanning trees in $\Re^d$.

REFERENCES

1. M. Ajtai, J. Komlos and E. Szemeredi, An $O(n \log n)$ Sorting Network , *Proc. of the 15th Annual ACM Symp. on Theory of Computing*, 1983, pp. 1-9 .
2. J. L. Bentley and M. I. Shamos, Divide-and-Conquer in Multidimensional Space , *Proc. of the 8th Annual ACM Symp. on Theory of Computing*, 1976, pp. 220-230 .
3. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, Time Bounds for Selection , *J. Computer and System Sciences*, 7 (4), 1973, pp. 448-461 .
4. A. Borodin and J. E. Hopcroft, Routing, Merging and Sorting on Parallel Models of Computation , *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, 1982, pp. 338-344 .
5. B. Chazelle, New Techniques for Computing Order Statistics in Euclidean Space , *Proc. of the 1st Annual Symposium on Computational Geometry*, 1985, pp. 125-134 .
6. R. Cole, Slowing Down Sorting Networks to Obtain Faster Sorting Algorithms , *Proc. of the 25th Annual Symp. on Foundations of Computer Science*, 1984, pp. 255-259 .
7. R. Cole, J. S. Salowe, W. L. Steiger and E. Szemeredi, Selecting Slopes , Submitted to Siam J. on Computing, 1987.
8. D. P. Dobkin and R. J. Lipton, On the Complexity of Computations Under Varying Sets of Primitives , *J. Computer and System Sciences*, **18** , 1979, pp. 86-91 .
9. G. N. Frederickson and D. B. Johnson, The Complexity of Selection and Ranking in $X+Y$ and Matrices with Sorted Columns , *J. Computer and System Sciences*, **24** , 1982, pp. 197-208 .
10. G. N. Frederickson and D. B. Johnson, Generalized Selection and Ranking: Sorted Matrices , *Siam J. on Computing*, **13** (1), 1984, pp. 14-30 .
11. Z. Galil and N. Megiddo, A Fast Selection Algorithm and the Problem of Optimum Distribution of Effort , *J. ACM* , **26** , 1979, pp. 58-64 .

12. D. B. Johnson and T. Mizoguchi, Selecting the k^{th} Element in $X+Y$ and $X_1+X_2+\cdots+X_m$, *Siam J. on Computing*, **7**, 1978, pp. 147-153.
13. G. S. Lueker, A Data Structure for Orthogonal Range Queries, *Proc. of the 19th Annual Symp. on Foundations of Computer Science*, 1978, pp. 28-34.
14. N. Megiddo, Combinatorial Optimization with Rational Objective Functions, *Math. Oper. Res.*, **4** (4), 1979, pp. 414-424.
15. N. Megiddo, A. Tamir, E. Zemel and R. Chandrasekaran, An $O(n \log^2 n)$ Algorithm for the k^{th} Longest Path in a Tree with Applications to Location Problems, *Siam J. on Computing*, **10** (2), 1981, pp. 328-337.
16. N. Megiddo, Applying Parallel Computation Algorithms in the Design of Serial Algorithms, *J. ACM*, **30** (4), 1983, pp. 852-865.
17. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer Verlag, New York, NY, 1985.
18. J. S. Salowe, *Selection Problems in Computational Geometry*, PhD Thesis, Rutgers University, 1987.
19. M. I. Shamos, Geometry and Statistics: Problems at the Interface, in *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub (ed.), Academic Press, New York, NY, 1976, 251-280.
20. L. G. Valiant, Parallelism in Comparison Problems, *Siam J. on Computing*, **4** (3), 1975, pp. 348-355.
21. D. E. Willard, *Predicate-oriented Database Search Algorithms*, PhD Thesis, Harvard University, 1978.
22. A. C. Yao, On Constructing Minimum Spanning Trees in k-Dimensional Space and Related Problems, *Siam J. on Computing*, **11** (4), 1982, pp. 721-736.