SELECTING VERTICES IN ARRANGEMENTS OF HYPERPLANES

Jeffrey S. Salowe

Computer Science Report No. TR-87-19
November 13, 1987

SELECTING VERTICES IN ARRANGEMENTS OF HYPERPLANES

ABSTRACT

Given any arrangement of n hyperplanes in $\mathbf{R}^d$, there are $O(n^d)$ vertices. This paper shows that the time complexity of selecting the vertex with k^{th} smallest x_1-coordinate in such an arrangement is $O(n^{d-1} log^{3/2} n)$, which is sublinear in the number of vertices. This extends the result of [6] from 2-dimensions to d-dimensions.

1 Introduction

1.1 Background

Let $S = \{x_1,...,x_n\}$ be a totally ordered set of n items. The rank of $x \in S$ is given by $rank(x) = |\{v : v \leq x, v \in S\}|$. Given S and an item $x \in S$, the ranking problem is to determine $rank(x)$; for inputs S and k, $1 \leq k \leq n$, the selection problem is to return an element x for which $rank(x) = k$, and the verification problem is to decide for input item x and input rank k whether $rank(x) = k$. This paper deals with the selection problem when S is constrained to be the set of vertices of a d-dimensional arrangement.

Selection problems are interesting in several respects. First of all, order statistics succinctly describe a set of objects in a robust way. For instance, both the median and the mean approximate the behavior of a "typical" member of a set, but the median is less sensitive to erroneous data than the mean. The problem is also interesting since selection is sometimes used to partition in divide-and-conquer or elimination algorithms.

The algorithm of [3] shows that for any set S, the k^{th} largest (or smallest) element may be selected in $O(n)$ time. However, if order relations are known between elements of S, we can sometimes select in $o(n)$ time. This phenomenon was observed for problems which are combinatorial in nature by Shamos, Jefferson and Tarjan in [22], and subsequently by [12, 13, 14, 18] and many others. These algorithms have several known applications in statistics (calculating the Hodges-Lehman estimator, [12, 22]) and operations research (optimum distribution of effort, finding p-centers, [14, 15, 18]).

In [4], Chazelle considered the selection problem when the set S is constrained by geometry. He described one technique to build efficient selection algorithms for geometric problems which consists of mapping the geometric set onto a combinatorial structure with a known efficient selection algorithm. The combinatorial recipient is either a group of matrices with sorted columns or a group of matrices with sorted rows and sorted columns. If the time needed to map the set is sublinear in its size, the resulting selection algorithm will run in sublinear time.

A second technique for geometric selection was described in [6] and [21]. This method is based on the parametric search idea developed in [19, 17, 5] along with a notion of approximate ranking. In [6], an optimal time algorithm was presented for the selection of slopes, a problem that was initially mentioned by [22] as computing the Theil estimator and one which seems to resist Chazelle's method. We use this second technique to extend the slope selection result of [6] to the problem of selecting vertices in high dimensional arrangements of hyperplanes.

1.2 Terminology

We define the following terminology with respect to sets of points in $\mathbf{R}^d$. A p-flat is an affine subspace of dimension p. A 0-flat is called a point, a 1-flat is called a line and a $(d-1)$-flat is called a hyperplane. Let $H = \{h_1,....,h_n\}$ be a set of n hyperplanes. H induces a cell complex of $O(n^d)$ cells in $\mathbf{R}^d$ called a d-arrangement, and this arrangement will be denoted by $A_d(H)$. Each cell is convex, and the dimension of a cell is the dimension of its affine hull. The cells in $A_d(H)$ of dimension 0 are called vertices and the cells of dimension 1 are called edges.

The complete skeleton of arrangement $A_d(H)$ is the set of all vertices and edges of $A_d(H)$ [8, 9], but we will shorten this name to the skeleton. By a theorem of [10], an incidence graph representing the skeleton of an arrangement can be constructed in optimal $O(n^d)$ time. From this graph, it is possible to list the vertex-edge sequence along any line in the skeleton by simply following pointers. References [8, 10] provide a detailed description of this representation.

Initially, we assume that $A_d(H)$ is simple, which means that the common intersection

of p hyperplanes in H, $p \leq d$, is a $(d-p)$-flat. The simplicity assumption makes the following arguments easier to describe but can be removed without affecting the asymptotic time complexity. We warn the reader that this requirement is dropped in the last section to allow vertices at infinity.

1.3 An Overview of the Paper

We wish to prove the following theorem.

Theorem 1: Given a set H of n hyperplanes, the vertex in $A_d(H)$ with k^{th} smallest x_1-coordinate can be selected in time $O(d\, n^{d-1} log^{3/2} n)$.

The proof of Theorem 1 will be in three stages. In Section 2, we show how to rank a vertex in $A_d(H)$ according to its x_1-coordinate. In Section 3, this ranking algorithm is incorporated into Megiddo's parametric search paradigm [17], which results in an $O(d\, n^{d-1} log^2 n)$ time algorithm. We then complete the proof by showing how to shave off a factor of $\sqrt{log\, n}$ in Section 4 by approximating the rank of vertices in $A_d(H)$. This proof does not depend on the use of the x_1-coordinate so the theorem generalizes to any direction in $\mathbf{R}^d$. The last section of the paper shows a difficulty in proving non-trivial lower bounds for this problem.

2 The Ranking Procedure

The first step in proving the main theorem is to construct an efficient ranking algorithm. It turns out that the rank of the x_1-coordinate of a vertex in a d-arrangement can be calculated using information in the skeleton of the intersection of arrangement $A_d(H)$ and hyperplane $x_1 = t$. This structure will be called the intersection skeleton at t, and will be denoted by $A_{d-1}(H)|t$.

There are two distinct subproblems in the ranking computation. First, the vertex with the smallest x_1-coordinate (the leftmost vertex) must be found to establish a starting skeleton. Second, there must be a procedure which compares the intersection skeletons at t_1 and t_2 and returns the number of vertices $v_i = (x_{i_1},...,x_{i_d})$ for which $t_1 \leq x_{i_1} \leq t_2$.

Lemma 2: The leftmost vertex in $A_d(H)$ can be found in time $O(n^{d-1} log\, n)$.

Proof: Determining the leftmost vertex in $A_d(H)$ is solved by induction on d. A recursive algorithm can be built to simulate the induction.

As a basis, let P be a plane in $\mathbf{R}^d$ with unit vector e, and let $A_2(H \cap P)$ be a 2-arrangement of n lines in P. Also, let e_i be the unit vector corresponding to the i^{th} coordinate axis. If $e = \pm e_1$, any vertex in $A_2(H \cap P)$ is a leftmost vertex. Otherwise, orthogonally project $A_2(H \cap P)$ onto the x_1x_i plane, where $i > 1$ is the smallest index with $e \neq \alpha e_1 + \beta e_i$. Call this projected arrangement $A_2'(H \cap P)$. The leftmost vertex in $A_2'(H \cap P)$ may be found in $O(n \log n)$ time by sorting the projections of the lines in $H \cap P$ by slope, determining the intersection points of adjacent lines with respect to this sorted order and choosing the leftmost intersection point. Since the direction of projection is orthogonal to the x_1-axis, the leftmost vertex in $A_2'(H \cap P)$ corresponds to the leftmost vertex in $A_2(H \cap P)$.

Assume inductively that the leftmost vertex in any $(d-1)$-arrangement lying on an arbitrary hyperplane of n $(d-2)$ flats in $\mathbf{R}^d$ can be found in $O(n^{d-2} \log n)$ time. Arrangement $A_d(H)$ may be decomposed into a collection of n $(d-1)$ arrangements of $(n-1)$ $(d-2)$-flats which we denote by $A^i_{d-1}(H - \{h_i\}) = A_d(H) \cap h_i$. Every p-face in $A_d(H)$, $p \leq d-1$, is in $\cup_i A^i_{d-1}$, and a leftmost vertex must be in exactly d of the $A^i_{d-1}(H - \{h_i\})$ (this is true because of the simplicity assumption). To find the leftmost vertex in $A_d(H)$, we choose the vertex with minimum x_1-coordinate in each of the $A^i_{d-1}(H - \{h_i\})$ and pick the minimum of this set. This procedure can be effected in $O(n \cdot (n-1)^{d-2} \log n + n^2) = O(n^{d-1} \log n)$ time. •

Ranking is accomplished by relating the intersection skeleton passing through a point to the left of every vertex (the starting skeleton) to the intersection skeleton passing through the vertex to be ranked (the ranked skeleton). Specifically, the idea is to compare the number of inversions determined by the permutation of vertices along the 1-flats in the ranked skeleton with the number of inversions in the starting skeleton. This procedure is based on the following observation. Suppose hyperplane $x_1 = t$ sweeps past a vertex v_j of $A_d(H)$ with x_1-coordinate t_j. Let $v_j = h_{j_1} \cap h_{j_2} \cap \ldots \cap h_{j_d}$, and pick an arbitrary $(d-2)$-sized subset of these hyperplanes, $\{h_{j_1}, \ldots, h_{j_{d-2}}\}$. Examine the order in which the rest of the hyperplanes in H intersect line $h_{j_1} \cap \ldots \cap h_{j_{d-2}} \cap \{x_1 = t_j - \epsilon\}$ of $A_{d-1}(H)|t_j - \epsilon|$, $\epsilon > 0$, and suppose $h_{j_{d-1}}$ precedes h_{j_d}. There are two such sequences, one the reverse of the other: we require

that the orientation be consistent throughout the computation. As the hyperplane $x_1 = t$ sweeps further to the right, $h_{j_{d-1}}$ and h_{j_d} coincide in $A_{d-1}(H)|t_j|$, and $h_{j_{d-1}}$ follows h_{j_d} in $A_{d-1}(H)|t_j + \epsilon|$, $\epsilon > 0$. In the ordered list for $\{h_{j_1},...,h_{j_{d-2}}\}$, there is one additional inversion with respect to the starting sequence. Likewise, there is one additional inversion in the lists for each of the $\binom{d}{d-2} = \binom{d}{2}$ subsets of $\{h_{j_1},...,h_{j_d}\}$. At every vertex v_j, the total number of inversions along all the lines of $A_{d-1}(H)|t|$ is increased by exactly $\binom{d}{2}$, one for each line incident with v_j.

To implement this procedure, we construct the starting intersection skeleton $A_{d-1}(H)|t|$, for an appropriate t. For each line in the intersection skeleton, the initial list of vertices is computed by following pointers in the representation of the skeleton. Then, given an input vertex $v_i = (x_{i_1},...,x_{i_d})$, we construct $A_{d-1}(H)|x_{i_1} + \epsilon|$ and compute the ordered list of vertices along the lines determined by each $(d-2)$-set of hyperplanes. (The term ϵ disambiguates the order of coincident vertices at t.) By [10], this takes $O(n^{d-1})$ time and space. For each line, we calculate the number of inversions from the starting list for that line. The rank is this total number of inversions divided by $\binom{d}{2}$.

Correctness of the procedure follows from the argument in the previous paragraphs, and the time complexity of the procedure is dominated by the time to do inversion counts. Counting the number of inversions in one list of length n takes $O(n \log n)$ time [16], so for $O(n^{d-2})$ lists this amounts to $O(n^{d-1} \log n)$ time.

Theorem 3: A vertex in $A_d(H)$ can be ranked with respect to a given coordinate (or direction) in time $O(n^{d-1} \log n)$.

3 The Selection Procedure

The selection algorithm is a search procedure. There are a total of $O(n^d)$ vertices, each of which can be ranked in $O(n^{d-1} \log n)$ time. We use the technique of Megiddo [19, 17, 5] to "parametrically" find a subset of vertices of size $O(d \log n)$ which must contain the vertex with k^{th} smallest x_1-coordinate. Roughly, the procedure consists of (1) viewing the selection problem as a group of sorting problems which guide the algorithm to the k^{th} vertex, (2) using the ranking procedure to resolve comparisons in the sorting algorithm and (3) implementing the sorting algorithm with a serial version of the AKS sorting network. In the next few paragraphs we briefly review the parametric search technique.

In the special case when the guiding algorithm is sorting, Megiddo's technique can be sketched as follows. Suppose a "search" problem can be efficiently reduced to a sorting problem; although solving a sorting problem amounts to resolving $O(n \log n)$ comparisons, the comparisons in this sorting algorithm may be expensive (i.e. $\omega(1)$ time). Nevertheless, it may be the case that the comparisons have an ordering property such that the resolution of a single comparison can in turn resolve many other comparisons. That is, depending on the outcome of a given comparison C, either all the comparisons less than C can be resolved in constant time each or all those greater than C can be resolved.

Although any sorting algorithm based on comparisons will solve the search problem, a sequentialized parallel sorting algorithm affords the greatest flexibility and can be completed with the fewest expensive comparison resolutions. By a sequentialized version of a parallel algorithm, we mean an algorithm which performs each level in series so that the comparisons on the first level are executed in any order, then the comparisons on the second level, and so on. We use the AKS network as our parallel sorting algorithm [1], and it is made up of $O(\log n)$ levels with $\frac{n}{2}$ comparisons on each level.

Since the output of the sequentialized algorithm is not affected by the order in which comparisons on a given level are performed, the comparisons can be analyzed to determine the one that, regardless of its outcome, resolves the greatest number of other comparisons on that level. A natural idea is to resolve the median comparison, partition the comparisons into those that are resolved and those that are not and recurse on the latter set. Instead of $\frac{n}{2}$ expensive comparison computations per level, only $O(\log n)$ expensive comparisons are needed to resolve all comparisons on that level, $O(\log^2 n)$ over the entire network. In fact, the network can be resolved in $O(\log n)$ expensive comparisons (see 5 for details).

For the current problem of selecting the vertex in a d-arrangement with k^{th} smallest x_1-coordinate, the sorting problems determine the ordering of vertices in the $\binom{n}{d-2}$ lists corresponding to the intersection skeleton at the unknown k^{th} vertex, $v^* = (x_1^*, \ldots, x_d^*)$. Suppose $h_{i_{d-1}} < h_{i_d}$ in the initial ordering for $\{h_{i_1}, \ldots, h_{i_{d-2}}\}$. Each comparison is in the form

"Is $h_{i_{d-1}} < h_{i_d}$ in the list for $\{h_{i_1}, \ldots, h_{i_{d-2}}\}$ in $A_{d-1}(H)\,x_1^*$?"

This comparison can be resolved by ranking the vertex $v_{test} = h_{i_1} \cap \ldots \cap h_{i_d}$. The number of vertices to the left of $x_1 = t$ (i.e. the rank) is monotonically nondecreasing with respect to x_1-coordinate. Therefore, if $rank(v_{test}) < k$, $x_1(v^*) > x_1(v_{test})$, and $h_{i_{d-1}} > h_{i_d}$ in the list for $\{h_{i_1},\ldots,h_{i_{d-2}}\}$ in $A_{d-1}(H)[x_1^*]$, where $x_1(v)$ is the x_1-coordinate of v. If $rank(v_{test}) > k$, $x_1(v^*) < x_1(v_{test})$, and $h_{i_{d-1}} < h_{i_d}$ in the list for $\{h_{i_1},\ldots,h_{i_{d-2}}\}$ in $A_{d-1}(H)[x_1^*]$. Otherwise, $rank(v_{test}) = k$ and $v^* = v_{test}$.

The following lemma shows that the vertex with k^{th} smallest x_1-coordinate must be ranked if all the sorting problems are solved.

Lemma 4: Any algorithm that sorts the vertices along each line in the intersection skeleton at $v^* = h_{i_1} \cap \ldots \cap h_{i_d}$ by comparisons must compare $h_{i_{d-1}}$ and h_{i_d} in the list for $\{h_{i_1},\ldots,h_{i_{d-2}}\}$.

Proof: In the list corresponding to $\{h_{i_1},\ldots,h_{i_{d-2}}\}$ in the intersection skeleton passing through v^*, $h_{i_{d-1}}$ and h_{i_d} must be adjacent since they are coincident at v^*. Suppose the sorting algorithm does not compare $h_{i_{d-1}}$ with h_{i_d}. Then the results of all other comparisons are the same regardless of whether $h_{i_{d-1}} < h_{i_d}$, $h_{i_{d-1}} > h_{i_d}$ or $h_{i_{d-1}} = h_{i_d}$. This contradicts the assumption that the algorithm obtains the sorted list. •

Theorem 5: The vertex in $A_d(H)$ with k^{th} smallest x_1-coordinate can be selected in $O(d\, n^{d-1} \log^2 n)$ time.

Proof: We use Cole's improvement to implement the parametric search [5]. The vertex selection problem satisfies the two criteria mentioned in that paper.

1. The problem can be solved by sorting where each comparison costs $O(n^{d-1} \log n)$.
2. The comparisons can be ordered by x_1-coordinate. That is, let C_1 and C_2 be two comparisons. Each comparison is in the form "Is $h_{i_{d-1}} < h_{i_d}$ in the list for $\{h_{i_1},\ldots,h_{i_{d-2}}\}$ in $A_{d-1}(H)[x_1^*]$?" so each comparison corresponds to a vertex $v_i = h_{i_1} \cap \ldots \cap h_{i_d}$. Then $C_1 < C_2$ if and only if $x_1(v_1) < x_1(v_2)$.

There are $\binom{n}{d-2}$ sorting networks, one for each line in the intersection skeletons, and if they are run simultaneously, they can be considered to be a single algorithm with $O(\log n)$ levels and $O(n^{d-2}) \cdot O(n) = O(n^{d-1})$ processors per level. The results of [5] show that $O(d \log n)$ expensive comparisons (i.e. rankings) suffice to resolve all $O(n^{d-1} \log n)$ com-

parisons. Since the cost of each expensive comparison is $O(n^{d-1} \log n)$ and the cost of all other comparisons is $O(1)$, $O(d\, n^{d-1} \log^2 n)$ time is needed. Correctness follows by the discussion above, Lemma 4 and a similar argument proving that v^* must be one of the ranked comparisons in the network. •

4 Faster Selection by Approximate Ranking

It is not necessary to rank a vertex in order to resolve a comparison. All we need is the relative position of v_{test} with respect to v^*. In the previous section, this step took $O(n^{d-1} \log n)$ time, but in this section, it is shown that relative position can be determined by an approximation at an average cost of $O(n^{d-1} \log^{1/2} n)$ per vertex. This leads to an $O(d\, n^{d-1} \log^{3/2} n)$ time selection algorithm. When $d = 2$, this approximation was developed by Cole, Salowe, Steiger and Szemeredi, but it was replaced by a more economical approximation in [6]. We first present the approximation for $d = 2$ and then extend this approximation to higher dimensions.

4.1 The Approximation

In the case $d = 2$, the starting intersection skeleton consists of n points along the vertical line $x_1 = t$, where t is less than the smallest x_1-coordinate in $A_2(H)$. If we assume the lines in the arrangement are indexed according to decreasing slope, point $y_i(t)$ is the x_2-coordinate of the intersection of vertical line $x_1 = t$ with the i^{th} line in the arrangement, so the $y_i(t)$ are initially sorted. At any other intersection skeleton, the top-to-bottom order of indices is permuted, and the number of inversions in this permutation is precisely the rank of the corresponding vertex. It is evident that the key to approximating the rank is to approximate the number of intersecting lines between two x_1-coordinates.

Theorem 6: (Cole, Salowe, Steiger, Szemeredi) The k^{th} vertex in a 2-arrangement can be found in $O(n \log^{3/2} n)$ time.

We prove this result by a series of lemmas. First, we describe the top level of how the approximation works, and then we present the actual approximation algorithm. The approximation is analyzed by first bounding its error and then determining how much additional work must be done to ensure the approximation is accurate.

The top level of the approximation algorithm estimates the number of inversions between the intercepts along reference line r and the intercepts along the vertical query line q incident to v_{test}. A reference line has the property that its rank is known exactly; the approximate rank of vertex v_{test} is thus the sum or difference of the number of inversions at the reference line and the approximate number of inversions between the reference line and q. The absolute error e of the latter approximation depends on the true number of inversions between r and q.

Denote the approximate number of inversions between r and q by K', let K be the true number of inversions, let K^* be the number of inversions between r and the vertical line incident to v^*, and suppose that both q and v^* are to the right of r. Note that $K \in (K'-e, K'+e)$ by definition. Whenever $K^* \notin (K'-e, K'+e)$, the relative ordering of v_{test} and v^* can be determined by the approximation. That is, if $K^* > K'+e$, v^* is to the right of v_{test}, and if $K^* < K'-e$, v^* is to the left of v_{test}. If $K^* \in (K'-e, K'+e)$, the ordering of v_{test} and v^* cannot be determined by the approximation. In this case, $rank(v_{test})$ is computed exactly and q is made the new reference line. An algorithm for the top level of the approximation is given in Figure 1. In this description, we assume that the relative position of v_{test} and v^* cannot be determined by transitivity.

Algorithm APPROX was called by the top level of the algorithm and consists of the following steps (see Figure 2). Assume that r and q are as before and that the intercepts along r are numbered in top-to-bottom sequence from 1 to n. If $G \subseteq [n]$, let the image of the i^{th} point in G at q be $\Pi_G(i)$. The algorithm will sort a small subset of $\Pi_{[n]}$ and use the number of inversions to find the estimate for K, K'.

The first step is to divide the points at r into $\lfloor n/\tau \rfloor$ contiguous intervals of size $\lceil \tau \rceil$ and possibly one group of size less than $\lceil \tau \rceil$. For each interval G, merge-sort $\Pi_G(i)$, $1 \le i \le |G|$. This has the effect of ordering the image of G as well as counting the number of inversions between elements of G. This number of inversions is added to K'.

The elements $\Pi_G(i)$ are now sorted. Assign the first $\lceil \tau^{3/4-\epsilon/2} \rceil$ elements and the last $\lceil \tau^{3/4-\epsilon/2} \rceil$ elements weight 1. A small sample of the remaining elements is taken. Divide them into intervals of length $\gamma = \lceil \tau^{1/2} \rceil$, assign the median of each interval weight γ, and assign the rest weight zero. For the topmost interval G_1 to the bottommost interval, place

ALGORITHM TOP-LEVEL $(v_{test}, r, k, position)$

```
q ← x_1(v_test)
IF rank(r) < k THEN
        K* ← k − rank(r)
        K' ← Approx(r,q)
        IF K* ∈ (K' − e, K' − e) THEN
            ρ_{v_test} ← rank(v_test)
            IF ρ_{v_test} > k THEN
                position ← "right"
            ELSE IF ρ_{v_test} < k THEN
                position ← "left"
            ELSE
                position ← "on"
            Record sequence of intercepts at q
            r ← q
        ELSE
            IF K* < K' − e THEN
                position ← "left"
            ELSE
                position ← "right"

ELSE
        K* ← rank(r) − k
        K' ← Approx(r,q)
        IF K* ∈ (K' − e, K' − e) THEN
            ρ_{v_test} ← rank(v_test)
            IF ρ_{v_test} > k THEN
                position ← "right"
            ELSE IF ρ_{v_test} < k THEN
                position ← "left"
            ELSE
                position ← "on"
            Record sequence of intercepts at q
            r ← q

        ELSE
            IF K* < K' − e THEN
                position ← "right"
            ELSE
                position ← "left"
```

Figure 1: ALGORITHM Top-level

ALGORITHM APPROX (r, q)

$K' \leftarrow 0$

DIVIDE the intercepts at r into consecutive intervals of size $\lceil \tau \rceil$

$G_i = \{y_{\sigma_j}(r) : (i-1)\lceil \tau \rceil + 1 \leq j \leq i\lceil \tau \rceil\}$, σ sorts $\{y_1(r),...,y_n(r)\}$.

FOR each group G_i

SORT $\Pi_{G_i}(j)$, $1 \leq j \leq |G_i|$, and

CALCULATE the number of inversions C within G_i.

$K' \leftarrow K' - C$

ASSIGN

$\{j : \Pi_{G_i}(j) \in (1, \lceil \tau^{3/4+\epsilon/2} \rceil)\}$

$\{j : \Pi_{G_i}(j) \in (\lceil \tau \rceil - \lceil \tau^{3/4-\epsilon/2} \rceil + 1, \lceil \tau \rceil)\}$

weight 1.

ASSIGN

$\{j : \Pi_{G_i}(j) = \lceil \tau^{3/4+\epsilon/2} \rceil + (k+1/2)\lceil \tau^{1/2} \rceil\}$,

$0 \leq k \leq \lfloor \lceil \tau \rceil - 2\lceil \tau^{3/4+\epsilon/2} \rceil - \lceil \tau^{1/2} \rceil / 2 \rfloor \cdot \lceil \tau^{1/2} \rceil - 1$

weight $\lceil \tau^{1/2} \rceil$.

FOR $\Pi_{G_i}(1)$ **TO** $\Pi_{G_i}(\lceil \tau \rceil)$

IF $weight\,(\Pi_{G_i}(j)) > 0$ **THEN**

APPEND to list l

Weighted-Inversions$(l.A, l.\ell, k)$

$K' \leftarrow K' - k$

Figure 2: ALGORITHM Approx

the elements of positive weight, in their order at q, into the list l. Therefore, the elements within a group are ordered with respect to q, but elements in different groups are ordered with respect to r.

The estimate K' for the number of inversions between r and q is the number of inversions within the G_m's plus the estimated number of inversions between the G_m's. The number of inversions within the G_m's was counted when the groups were sorted, and the estimated number of inversions between the G_m's is calculated by algorithm Weighted-Inversions(l,A,$|l|$,k). (See Figure 3.) Algorithm Weighted-Inversions has the property that whenever i and j invert (i is above j at r but j is above i at q), $weight(i) \cdot weight(j)$ is contributed to the weighted sum.

Lemma 7: (Correctness) For list l:

1. Weighted-Inversions does not detect an inversion within a group G.
2. If $x_1 \in G_i$, $x_2 \in G_j$, $i \neq j$, invert between r and q, then $weight(x_1) \cdot weight(x_2)$ is contributed to K'.
3. If $x_1 \in G_i$, $x_2 \in G_j$, $i \neq j$, do not invert, nothing is contributed to K'.

Proof: For an arbitrary weighted permutation, if $weight(x_1) \cdot weight(x_2)$ is contributed to the output sum whenever x_1 and x_2 invert and if nothing is contributed to the sum whenever x_1 and x_2 do not invert, then all three claims are true. The sort is stable and the elements in G are placed into l according to their ordering at q.

Assume x_1 is before x_2 in l. During the execution of Weighted-Inversions, x_1 and x_2 are eventually in adjacent lists, with the list containing x_1 immediately to the left of the list containing x_2. Element x_1 precedes element x_2 initially, and the only time they can invert is when these lists are merged. During this merge, the weight of x_2 is multiplied by the sum of the weights of the elements in the left list which are below x_2 at q. If x_1 and x_2 cross, $weight(x_1) \cdot weight(x_2)$ is contributed: otherwise, nothing is contributed. •

ALGORITHM Weighted-Inversions (L, A, n, k)

/* L is the input list, n is its size, A is the sorted output and k is the

/* total of weighted inversions. Elements are compared based on their order at q.

IF n=1 **THEN**

$A \leftarrow L$

$k \leftarrow 0$

RETURN

$i, i_1, i_2 \leftarrow 1$

Weighted-Inversions $(L[1:\lceil n/2 \rceil], A1, \lceil n/2 \rceil, k1)$

Weighted-Inversions $(L[\lceil n/2 \rceil + 1:n], A2, n - \lceil n/2 \rceil, k2)$

$k \leftarrow k1 + k2$

$T \leftarrow 0$

FOR $i = \lceil n/2 \rceil$ **DOWNTO** 1

$T \leftarrow T + weight(A1[i])$

$W_i \leftarrow T$

WHILE $i_1 \leq \lceil n/2 \rceil$ **AND** $i_2 \leq n - \lceil n/2 \rceil$

IF $A1[i_1] < A2[i_2]$ **THEN**

$A[i] \leftarrow A1[i_1]$

$i_1 \leftarrow i_1 + 1$

$i \leftarrow i + 1$

ELSE

$A[i] \leftarrow A2[i_2]$

$i_2 \leftarrow i_2 + 1$

$i \leftarrow i + 1$

$k \leftarrow k + weight(A2[i_2]) \cdot W_{i_1}$

IF $i_1 > \lceil n/2 \rceil$ **THEN**

$A[i:n] \leftarrow A2[i_2:n - \lceil n/2 \rceil]$

ELSE

$A[i:n] \leftarrow A1[i_1:\lceil n/2 \rceil]$

Figure 3: ALGORITHM Weighted-Inversions

4.2 The Analysis of the Approximation

Throughout this section, we assume that r is sufficiently large. If K' is the number of inversions counted by algorithm APPROX, the following series of lemmas show that $e \leq \frac{K'}{\lfloor r^{\epsilon} \rfloor}$, $0 < \epsilon < 1/2$; we show why this error behavior is useful in Lemma 11. The bound on the error is done by comparing the estimated number of inversions between two groups G_i and G_j, $i < j$, with the true number of inversions. For notation, let

$$K_{ST} = |\{(x,y) : x \in S, y \in T, x < y, \Pi(x) > \Pi(y)\}|.$$

and let K'_{ST} and e_{ST} be the restrictions of K' and e to sets S and T. If $K'_{G_iG_j}$ and $K_{G_iG_j}$ are respectively the estimated and the true number of inversions between the elements of G_i and G_j, $i \neq j$, we show that for every G_i–G_j pair,

$$K_{G_iG_j} \in (K'_{G_iG_j} - \frac{2K'_{G_iG_j}}{\lfloor r^{\epsilon} \rfloor}, K'_{G_iG_j} - \frac{2K'_{G_iG_j}}{\lfloor r^{\epsilon} \rfloor}).$$

This implies $K \in (K' - \frac{2K'}{\lfloor r^{\epsilon} \rfloor}, K' - \frac{2K'}{\lfloor r^{\epsilon} \rfloor})$. The proof of this statement is divided into cases based on the weights of the elements. First, we bound the error contributed by the weight 1 elements.

Lemma 8: Let $x \in G_j$, $weight(x) = 1$. Then either

$$K'_{G_ix} = K_{G_ix}$$

or

$$K'_{G_ix} \geq \lceil r^{3/4 + \epsilon/2} \rceil, \; e_{G_ix} \leq \gamma/2.$$

Proof: If x crosses all of G_i, $K'_{G_ix} = K_{G_ix} = \lceil r \rceil$, while if x does not cross any element in G_i of weight 1, $K'_{G_ix} = K_{G_ix} = 0$. Otherwise, the image of x lies between the image of z_1 and the image of z_2, $z_1, z_2 \in G_i$. If z_1 and z_2 have weight 1, then $K'_{G_ix} = K_{G_ix}$. If z_1 has weight 1 and z_2 has weight γ, then K'_{G_ix} might overestimate K_{G_ix} by $\gamma/2$. If z_1 has weight γ and z_2 has weight 1, K'_{G_ix} might underestimate K_{G_ix} by $\gamma/2$. Otherwise

$weight(z_1) = weight(z_2) = \gamma$. In this case, the error can vary from $-\gamma/2$ to $\gamma/2$. In each of the latter three cases, element x has crossed the lower $\lceil r^{3/4+\epsilon/2} \rceil$ elements of G_i, each of which has weight 1 and is therefore counted in the approximation. •

Denote the set of elements in G_i of weight γ as g_i. A bound on the error contributed by these elements is given in the following lemma.

Lemma 9:

$$e_{g_i g_j} \leq \lceil r^{3/2} \rceil$$

Proof: Suppose that G_i is above G_j at r, and consider the ordered sequence of $g_i \cup g_j$ at q. Denote this as

$$\ldots x_{i_1} \ldots x_{i_a} y_{j_1} \ldots y_{j_b} x_{i_{a+1}} \ldots x_{i_{at}} y_{j_{b+1}} \ldots$$

where $x_{i_m} \in g_i$ and $y_{j_m} \in g_j$. Each x or y is the median of a small group of size γ. Let p be the number of alternations from y to x in this sequence and note that $0 \leq p \leq \gamma$.

Each x_{i_s} corresponds to an interval in G_j of size γ, but errors may occur due to the coarseness of the approximation. If x_{i_s} inverts with adjacent y_{j_m}, then the approximation adds $\lceil r \rceil$ to the sum, though only $\frac{\lceil r \rceil}{4}$ inversions must occur in the exact count. Similarly, if x_{i_s} does not cross y_{j_m}, the approximation counts 0 inversions, though as many as $(\gamma/2-1)^2$ may occur in the exact count.

The worst overestimation happens in the following situation. At a $y_1 \ldots y_m \rightarrow x_1 \ldots x_s$ block, $(\gamma/2-1)$ of the elements in the group represented by y_m do not cross the block $x_1 \ldots x_s$. Similarly, $(\gamma/2-1)$ of the elements in the group represented by x_1 do not cross the block $y_1 \ldots y_m$. These blocks are isolated from the remainder of the list by the (possibly nonexistent) x that precedes y_1 and the y that follows x_s. This means that $(\gamma/2+1)+(m-1)\gamma$ of the y's must cross $(\gamma/2+1)+(s-1)\gamma$ of the x's. The approximate count is thus $sm\lceil r \rceil$ while the actual count may be as small as $(s-1/2)(m-1/2)\lceil r \rceil$. The error per alternation is therefore at most $(s+m)\frac{\lceil r \rceil}{2} - \frac{\lceil r \rceil}{4}$. Summing this error over all the alternations gives

$$e_{g_i g_j} \leq (2\gamma)\frac{\lceil \tau \rceil}{2} - p\frac{\lceil \tau \rceil}{4}$$

$$\leq \lceil \tau^{3/2} \rceil$$

as asserted.

The worst underestimation is similar. •

Now that we have bounded the errors contributed by weight 1 elements and weight γ elements, we present two main lemmas describing the approximation. The first lemma gives the accuracy of the approximation and the second counts how many expensive updates are needed.

Lemma 10:

$$K_{G_i G_j} \in (K'_{G_i G_j} - \frac{2K'_{G_i G_j}}{\lfloor \tau^{\epsilon} \rfloor}, K'_{G_i G_j} - \frac{2K'_{G_i G_j}}{\lfloor \tau^{\epsilon} \rfloor}),$$

$0 < \epsilon < 1/2.$

Proof: Assume that G_i is above G_j at r. For the proof, abbreviate $K_{G_i G_j}$ to K, and do the same for K' and e. The goal is to show that $e \leq \frac{2K'}{\lfloor \tau^{\epsilon} \rfloor}$: there are two cases.

Case 1: No weight γ items cross. Then the only intersections which contribute to the estimation are those between weight 1 elements or those between weight 1 elements with weight γ elements. Denote the lowest $\lceil \tau^{3/4+\epsilon/2} \rceil$ elements from G_i as L_{G_i}, and denote the highest $\lceil \tau^{3/4+\epsilon/2} \rceil$ elements from G_j as U_{G_j}. If $min(U_{G_j})$ is below some element of L_{G_i}, then the only intersections and errors possible are those determined by Lemma 8. Therefore for each element, either $e = 0$ or

$$e \leq \frac{\gamma}{2} < \lceil \tau^{3/4-\epsilon/2} \rceil \leq \frac{K'}{\lfloor \tau^{\epsilon} \rfloor}.$$

If $min(U_{G_j})$ crosses $max(L_{G_i})$, then there may be an additional error of $(\gamma/2-1)^2$ due to the weight 0 items between the weight γ items. Nevertheless,

$$e < \frac{\lceil \tau \rceil}{4} + \lceil \tau^{5/4 + \epsilon/2} \rceil \leq \lceil \tau^{3/2} \rceil \leq \frac{\lceil \tau^{3/2 + \epsilon} \rceil}{\tau^{\epsilon}} \leq \frac{K'}{\tau^{\epsilon}}.$$

Case 2: Some weight γ items cross. Then Lemmas 8 and 9 show that the total error is bounded by

$$\lceil \tau^{3/2} \rceil + 2\lceil (\tau^{3/4 + \epsilon/2}) \rceil \cdot \frac{\gamma}{2} < 2\tau^{3/2}.$$

The first term is due to weight γ elements and the second term is due to weight 1 elements. However, since all of U_{G_j} crosses all of L_{G_i},

$$e < 2\tau^{3/2} \leq \frac{2\tau^{3/2 + \epsilon}}{\tau^{\epsilon}} \leq \frac{2K'}{\tau^{\epsilon}}. \quad \bullet$$

As a consequence of the error bound, we have the second main lemma which gives a bound on the number of reference line updates.

Lemma 11: Let $K \in (K' - \frac{2K'}{\tau^{\epsilon}}, K' - \frac{2K'}{\tau^{\epsilon}})$. Then at most $O(\frac{\log n}{\epsilon \log \tau})$ reference line updates are needed.

Proof: Let K^*_j be the actual number of inversions between the j^{th} reference line and the vertical line passing through v^*. Recall that an update is needed whenever $K^* \in (K' - \frac{2K'}{\tau^{\epsilon}}, K' + \frac{2K'}{\tau^{\epsilon}})$. Then, the maximum number of inversions between the vertical line incident with v^* and q is $\frac{4K'}{\tau^{\epsilon}}$. Therefore,

$$K^*_i \leq \frac{4}{\tau^{\epsilon}} K'_{i-1} \leq \frac{4}{\tau^{\epsilon}} K^*_{i-1} (1 - \frac{2}{\tau^{\epsilon}})^{-1}$$

so the errors are cut by a factor of $\frac{4}{\tau^{\epsilon}}$ at each update. Since

$$K^*_0 \leq \binom{n}{2},$$

the recurrence leads to

$$K^*_i \leq \frac{4^i}{\lfloor \tau^{\epsilon} \rfloor^i} n^2 (1 - \frac{2}{\lfloor \tau^{\epsilon} \rfloor})^{-i}.$$

The smallest number of iterations to ensure $K^*_i \leq 1$ is

$$i = O(\frac{\log n}{\epsilon \log \tau}). \quad \bullet$$

Proof of Theorem 6: The cost of algorithm APPROX is made up of the following terms. First, $\tau \log \tau$ is needed to sort each G_i and determine its internal inversions. For all $\frac{n}{\tau}$ groups, this is $\frac{n}{\tau} \cdot \tau \log \tau = n \log \tau$ time. Second, given this sorting information the list l is formed in $\frac{cn}{\tau} \cdot \lceil \tau^{3/4+\epsilon/2} \rceil$ time, and Weighted-Inversions makes its approximation in $O(|l| \log |l|)$ time. Since $|l|$ is $\frac{cn}{\lceil \tau^{1/4-\epsilon/2} \rceil}$, the cost of the approximation is

$$O(n \log \tau - \frac{cn}{\tau^{1/4-\epsilon/2}} \log n),$$

and at most $O(\log n)$ approximations are made.

By Lemma 11, at most

$$O(\frac{\log n}{\epsilon \log \tau})$$

full inversion counts are needed when the approximation cannot distinguish the relative order of v^* and v_{test}. Including the operation of the AKS network, this leads to

$$\log n \{n \log \tau - \frac{n}{\tau^{1/4-\epsilon/2}} \log n\} + n \log n \{\frac{\log n}{\epsilon \log \tau}\} - n \log n.$$

A balance is reached when $\log \tau = \sqrt{\log n}$, or $\tau = 2^{\sqrt{\log n}}$, in which case the total cost is

$$n \log^{3/2} n + \frac{n \log n}{2^{\{1/4-\epsilon/2\}\sqrt{\log n}}}.$$

The former term dominates the latter as n gets large. •

Proof of Theorem 1: For $d > 2$, the 2-dimensional approximation is a subcomponent of the d-dimensional approximation. Let l be an arbitrary line in the intersection skeleton of $A_{d-1}(H)[t]$. Let K_l be the true number of inversions between $A_{d-1}(H)[r]$ and $A_{d-1}(H)[q]$ along l, where $x_1 = r$ is a reference hyperplane and $x_1 = q$ is the hyperplane through the vertex to be ranked. Using the same notation as before, the approximation K'_l returned by APPROX satisfies

$$K_l \in (K'_l - \frac{K'_l}{\lfloor \tau^\epsilon \rfloor}, K'_l - \frac{K'_l}{\lfloor \tau^\epsilon \rfloor}).$$

If K is the total number of inversions between r and q summed over all the lines in the intersection skeleton, then

$$K \in (K' - \frac{K'}{\lfloor \tau^\epsilon \rfloor}, K' - \frac{K'}{\lfloor \tau^\epsilon \rfloor}).$$

Choosing $\tau = 2^{\sqrt{\log n}}$ leads to $O(n^{d-1} \log^{1/2} n)$ time approximate rankings with only $O(d \log^{1/2} n)$ full rankings. The analysis is identical to Theorem 6. and the main theorem is proved. •

5 Remarks on Lower Bounds

Obtaining a good lower bound on the vertex selection problem seems to be difficult because of its association with bounds on the complexity of determining whether a set of n points is in general position. Though the definition of general position varies from problem to problem. we consider the case of deciding whether there is a $(d+1)$-sized subset of n points in $\mathbf{R}^d$ which lie on a single hyperplane. This problem will be called (d)-general position. It is known that (d)-general position can be decided in time $O(n^d)$ for $d > 1$ and in time $O(n \log n)$ time for $d = 1$ [10, 11]. However, the only tight lower bound is $d = 1$ in

that [7] proved that "element distinctness" requires $\Omega(n \log n)$ steps in the linear decision tree model. This result was subsequently strengthened to the algebraic decision tree model by [2, 23] (see [20]).

There is a natural correspondence between finding the leftmost vertex in $A_d(H)$ and $(d-1)$-general position, provided we allow a vertex at infinity to be the leftmost (as well as the rightmost) vertex. For notation, $A \leq_{C(n)} B$ means that an instance of problem A can be transformed to an instance of problem B in $C(n)$ time.

Theorem 12: For $d \geq 2$,

$(d-1)$-general position $\leq_{O(n)}$ Leftmost vertex in $A_d(H)$.

Proof: Suppose there is an algorithm which on input $H = \{h_1,...,h_n\}$, returns the leftmost vertex in $A_d(H)$. Note that if d hyperplanes do not intersect in a point, they describe a vertex at infinity. The vertex at infinity is both a leftmost and rightmost vertex. We now present a method to decide $(d-1)$-general position using a subroutine which returns the leftmost vertex in $A_d(H)$.

To decide if n points are in $(d-1)$-general position, we define two transformations, T and D. Let $p_i = (t_1,...,t_{d-1}) \in \mathbf{R}^{d-1}$ be an input point. $T: \mathbf{R}^{d-1} \rightarrow \mathbf{R}^d$ is defined by $T p_i = (t_1,...,t_{d-1},i) \in \mathbf{R}^d$, and $D: \mathbf{R}^d \rightarrow \mathbf{R}^d$ is the standard point-hyperplane duality

$$p: (t_1,...,t_d) \rightarrow D p: x_d = t_1 x_1 - ... - t_{d-1} x_{d-1} - t_d$$

$$h: x_d = t_1 x_1 - ... - t_{d-1} x_{d-1} + t_d \rightarrow D h: (-t_1,...,-t_{d-1},t_d).$$

The first step is to map each point $p_i = (t_1,...,t_{d-1})$ to $p_i' = T p_i$. Then we use duality D to form $H = \{D p_i'\}$ and call the leftmost vertex subprogram on H. If the input set is <u>not</u> in $(d-1)$-general position, there are d points lying on a $(d-2)$-flat in $\mathbf{R}^{d-1}$. The image of this set lies on a hyperplane in $\mathbf{R}^d$ whose normal vector has no x_d-component. Under duality D, the duals of these points "meet" in a vertex at infinity. On the other hand, if the input set is in $(d-1)$-general position, then there is no vertex at infinity in $A_d(H)$ and so the x_1-coordinate of the leftmost vertex is finite. •

Again, this theorem does not yield a non-trivial lower bound on finding the leftmost vertex in $A_d(H)$, but it does state that may be difficult to build an algorithm that can find the leftmost vertex in $A_d(H)$ in $o(n^{d-1})$ time since, despite years of effort, there are no $o(n^{d-1})$ time algorithms for $(d-1)$-general position

There is a second correspondence between selecting the leftmost vertex in $A_d(H)$ and $(d-1)$-general position. For convenience, we define the decision problem (d)-vertices at infinity: does arrangement $A_d(H)$ have a vertex at infinity?

Theorem 13: For $d \geq 2$.

(d)-vertices at infinity $\leq_{O(n)}$ $(d-1)$-general position.

The proof of this statement is basically the reverse of the last proof. Perturb $A_d(H)$ so that there are no vertical hyperplanes, then apply transformation D. If there is a vertex at infinity, then after D is applied, d points lie on a hyperplane in $\mathbf{R}^d$ whose normal vector has no x_d-component. Truncating each point to $\mathbf{R}_{d-1}$ by removing the last coordinate, this set of points is <u>not</u> in $(d-1)$-general position.

This suggests that a non-trivial lower bound for (d)-vertices at infinity is a difficult problem since it would imply the same lower bound for both $(d-1)$-general position ar leftmost vertex in $A_d(H)$. A more promising direction for future research is to lower the $O(d\, n^{d-1} \log^{3/2} n)$ time upper bound on selection. It is likely that, using the approximation idea of [6], the complexity of selection may be decreased to $O(d\, n^{d-1} \log n)$.

References

1. Ajtai, M., Komlos, J., Szemeredi, E. An $O(n \log n)$ Sorting Network. Proc. 15th Ann. Symp. Theory of Comput., ACM, 1983, pp. 1-9.

2. Ben-Or, M. Lower Bounds for Algebraic Computation Trees. Proc. 15th Ann. Symp. Theory of Comput., ACM, 1983, pp. 80-86.

3. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L, Tarjan, R.E. "Time Bounds for Selection". *Jour. Comp. Sci. Sys. 7*, 4 (1973), 448-461.

4. Chazelle, B. New Techniques for Computing Order Statistics in Euclidean Space. Proc. 1st Ann. Symp. on Comp. Geom., ACM, 1985, pp. 125-134.

5. Cole, R. Slowing Down Sorting Networks to Obtain Faster Sorting Algorithms. Proc. 25th Ann. Symp. Found. Comput. Sci., IEEE, 1984, pp. 255-259.

6. Cole, R., Salowe, J.S., Steiger, W.L., Szemeredi, E. Optimal Slope Selection. Submitted to SIAM J. Comput.

7. Dobkin, D.P., Lipton, R.J. "On the Complexity of Computations Under Varying Sets of Primitives". *Jour. Comp. Sci. Sys. 18* (1979), 86-91.

8. Edelsbrunner, H., *Arrangements and Geometric Computations.* Springer-Verlag, 1986. Forthcoming Book.

9. Edelsbrunner, H. "Edge-Skeletons in Arrangements With Applications". ***Algorithmica 1*** (1986), 93-109.

10. Edelsbrunner, H., O'Rourke, J., Seidel, R. "Constructing Arrangements of Lines and Hyperplanes with Applications". *SIAM J. Comput. 15* (1986), 341-363.

11. Edelsbrunner, H., Guibas L.J. Topologically Sweeping an Arrangement. Proc. 18th Ann. Symp. Theory of Comput., ACM, 1986.

12. Frederickson, G.N., Johnson, D.B. "The Complexity of Selection and Ranking in $X+Y$ and Matrices with Sorted Columns". *Jour. Comp. Sci. Sys. 24* (1982), 197-208.

13. Frederickson, G.N., Johnson, D.B. "Generalized Selection and Ranking: Sorted Matrices". *SIAM J. Comput. 13*, 1 (1984), 14-30.

14. Galil, Z., Megiddo, N. "A Fast Selection Algorithm and the Problem of Optimum Distribution of Effort". *JACM 26* (1979), 58-64.

15. Johnson, D.B., Mizoguchi, T. "Selecting the k^{th} Element in $X+Y$ and $X_1+X_2+\ldots+X_m$". *SIAM J. Comput. 7* (1978), 147-153.

16. Knuth, D.E., *The Art of Computer Programming. Volume III: Sorting and Searching.* Addison-Wesley, 1973.

17. Megiddo, N. "Combinatorial Optimization with Rational Objective Functions". *Math. Oper. Res. 4*, 4 (1979), 414-424.

18. Megiddo, N., Tamir, A., Zemel, E., Chandrasekaran, R. "An $O(n \log^2 n)$ Algorithm for the k^{th} Longest Path in a Tree with Applications to Location Problems". *SIAM J. Comput. 10*, 2 (1981), 328-337.

19. Megiddo, N. "Applying Parallel Computation Algorithms in the Design of Serial Algorithms". *JACM 30*, 4 (1983), 852-865.

20. Preparata, F.P., Shamos, M.I., *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

21. Salowe, J.S. *Selection Problems in Computational Geometry.* Ph.D. Th., Rutgers University, 1987.

22. Shamos, M.I. Geometry and Statistics: Problems at the Interface. In Traub, J.F., Ed., *Algorithms and Complexity: New Directions and Recent Results*, Academic Press, New York, NY, 1976, pp. 251-280.

23. Steele, J.M., Yao, A.C. "Lower Bounds for Algebraic Decision Trees". *J. Algorithms 3* (1982), 1-8.