International Journal of Computational Geometry & Applications Vol. 2, No. 1 (1992) 1–27
© World Scientific Publishing Company

A RANDOMIZED ALGORITHM FOR SLOPE SELECTION [*]

MICHAEL B. DILLENCOURT [†]
Information and Computer Science Department
University of California, Irvine, California 92717-3425, USA

DAVID M. MOUNT[‡]
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland, College Park, Maryland 20742, USA

and

NATHAN S. NETANYAHU[§]
Computer Vision Laboratory
Center for Automation Research
University of Maryland, College Park, Maryland 20742, USA

Received 12 November 1990
Revised 4 February 1992
Communicated by L. J. Guibas

ABSTRACT

A set of n distinct points in the plane defines $\binom{n}{2}$ lines by joining each pair of distinct points. The median slope of these $O(n^2)$ lines was proposed by Theil as a robust estimator for the slope of the line of best fit for the points. We present a randomized algorithm for selecting the k-th smallest slope of such a set of lines which runs in expected $O(n \log n)$ time. An efficient implementation of the algorithm and practical experience with the algorithm are discussed.

Keywords: Slope selection, randomized algorithm, Theil-Sen estimator, line fitting, robust estimation.

1. Introduction

Consider a set of n distinct points in the plane. These points define $\binom{n}{2}$ lines by joining each pair of distinct points. The *slope selection problem* is that of de-

[*]A condensed version of this paper appeared in the Third Canadian Conference on Computational Geometry, Vancouver, Canada, 1991, 135–140.

[†]This research was supported by a UCI Faculty Research Grant and by the National Science Foundation under grant IRI-88-02457.

[‡]This research was supported by the National Science Foundation under grant CCR-8908901.

[§]Present address: Mail Code 936, Space Data and Computing Division, NASA Goddard Space and Flight Center, Greenbelt, Maryland 20771, USA. This research was supported by the Defense Research Projects Agency (ARPA order No. 6989) and the U.S. Army Engineer Topographic Laboratories under Contract DACA76-89-C-0019.

termining the k-th smallest slope among these lines. This problem is of interest in statistical estimation. Given a set of n points which are hypothesized to lie on a straight line, the median slope of the lines determined by these points was proposed by Theil[30] and Sen[25] as a robust estimator of the slope of the line fitting the points. The advantage of this method over mean-based methods such as least squares is its lower sensitivity to outliers. More formally, the ***breakdown point*** of an estimator is roughly defined to be the fraction of outlying data points that may cause the estimator to take on an arbitrarily large aberrant value. (See Donoho and Huber[6] or Rousseeuw and Leroy[24] for an exact definition.) The Theil-Sen estimator in the plane has a breakdown point of $1 - \sqrt{1/2} \approx 29.3\%$ (Ref. 24, p. 67). The least squares estimator has a breakdown point of zero because a single outlier can bias the estimator arbitrarily.

There are a number of methods for determining a linear approximation to a set of points in the plane. In the following list, all but the last have a breakdown point of zero.

- The L_2 (least squares) estimator can be computed in $O(n)$ time (Ref. 5, pp. 88–95).
- The L_∞ (Chebyshev) estimator can be computed in $O(n \log n)$ time by first finding the convex hull of the point set.[26] Megiddo showed it could be computed in $O(n)$ time by linear programming in R^3 (Ref. 20).
- The L_1 approximation was shown to be computable in $O(n)$ time by Imai, Kato and Yamamoto,[15] also using Megiddo's technique.
- The line minimizing the squared median of the residuals, *least median of squares*, was proposed by Rousseeuw.[23] This estimator was shown to be computable in $O(n^2)$ time and $O(n^2)$ space by Souvaine and Steele.[28] The space bound was improved to $O(n)$ by Edelsbrunner and Souvaine.[9] The breakdown point for least median of squares is 50%. Although this is better than the Theil-Sen estimator, the computation time is relatively large.

Our motivation for studying the slope-selection problem arises from the problem in computer vision of fitting a line to a set of points in an image. The use of a Theil-like estimator by finding the median of intercepts for fitting a line to noisy images was proposed by Kamgar-Parsi, Kamgar-Parsi, and Netanyahu.[16] In addition to its relative insensitivity to outliers, this approach makes no assumptions regarding noise distribution, unlike the method of maximum likelihood estimation (Ref. 13, pp. 334–339). Furthermore, the Theil-like estimator requires no artificial quantization of line parameters, unlike the Hough transform [7,14] and the RANSAC paradigm of Fischler and Bolles.[12]

The problem of finding the k-th smallest slope was posed by Shamos.[27] It was subsequently considered by Cole, Salowe, Steiger, and Szemerédi,[4] who discovered an optimal (deterministic) $O(n \log n)$ algorithm for this problem. The algorithm of Cole *et al.* relies on two ingenious but sophisticated techniques: Cole's improvement of Megiddo's technique for parametric search based on parallel sorting algorithms, and an algorithm for computing approximate ranks of elements in an array.

We present a simple $O(n \log n)$ expected time randomized algorithm for the slope selection problem, describe an implementation of the algorithm including the handling of degeneracies, and demonstrate the algorithm's practical efficiency. While our time bound is theoretically weaker than the deterministic bound of Cole *et al.*, we feel that our algorithm will be of greater interest in practice, since (1) it is quite easy to implement, relying only on simple modifications of mergesort; (2) the constants of proportionality hidden by the asymptotic notation are small; and (3) the $O(n \log n)$ expected running time occurs with extremely high probability. The random variation in the running time of our algorithm is solely due to randomization and is not dependent on the structure of the input. This means that on *any* input set of size n, irrespective of its structure or its closeness of fit to a line, the running time depends only on the choice of random numbers. The algorithm always terminates in worst case $O(n^2)$ time (although the probability of achieving this worst case is extremely small for large n). When the algorithm terminates, its output is correct (not just probabilistically correct as is the case with some randomized algorithms).

Our algorithm is based on the basic subtasks of counting and sampling from a set of inversions in a list. An *inversion* in a list $a_1, a_2, \ldots, a_k$ of elements over a totally ordered domain is a pair of indices $i < j$ where $a_i > a_j$. In Section 2 we discuss the relevance of inversion counting and random sampling to the slope selection problem and present algorithms for these problems. In Section 3 we discuss how to apply random sampling to refine the search for the desired slope. We present the complete algorithm in Section 4, and we discuss its implementation and our experience with the algorithm's performance in Section 5.

2. Inversion Counting and Sampling

Let (a_i, b_i), for $1 \le i \le n$, denote a set of n distinct points in the real plane. Each of the $\binom{n}{2}$ distinct pairs of points determines a line. We wish to determine the k-th smallest slope (the median slope being a special case). If multiple lines have the same slope, then these slopes are counted multiply. We adopt the convention that a vertical line has the largest possible slope, $+\infty$. As observed by Cole *et al.*,[4] it seems to simplify the problem to describe it in its planar dual form, by transforming points into lines and vice versa. Consider the transformation that maps the point (a, b) to the line $y = ax - b$. For two points (a_i, b_i) and (a_j, b_j) the corresponding dual lines $y = a_i x - b_i$ and $y = a_j x - b_j$, respectively, intersect at the x-coordinate

$$x = \frac{b_i - b_j}{a_i - a_j}.$$

Thus the slope of the line passing through points (a_i, b_i) and (a_j, b_j) is equal to the x-coordinate of the intersection point of the two corresponding dual lines. We make the convention that if $a_i = a_j$ then these lines intersect at $x = +\infty$. This is consistent with our earlier assumption that vertical lines have slope $+\infty$. By a similar derivation it follows that the y-coordinate of the intersection point of the dual lines is just the negation of the y-intercept of the line determined by the original points. Thus this same algorithm can be applied for computing the median y-intercept of the set of lines determined by all pairs of points. In general, by the use of other dual transformations, median parameters for other line representations can be derived.

We have now reduced the slope selection problem to the following dual form. Given a set of n lines $y = a_i x - b_i$, $1 \leq i \leq n$, determine the k-th smallest x-coordinate among the $\binom{n}{2}$ intersection points of these lines. As before, multiple intersection points (where three or more lines intersect) are counted multiply. We use the term ***intersection ordinate*** to denote the x-coordinate of the intersection point between two lines. Note that this transformation is purely conceptual since there is no effort spent in converting the initial input into this form—we still assume that the input consists of a sequence of pairs (a_i, b_i) and the output will be a single real number, possibly $+\infty$.

Our basic approach is to maintain two x-values, x_{lo} and x_{hi}, $-\infty \leq x_{lo} \leq x_{hi} \leq +\infty$. Let $(x_{lo}, x_{hi}]$ denote the half-open, half-closed interval of points x, $x_{lo} < x \leq x_{hi}$. Let $I(x_{lo}, x_{hi}]$ denote the set of intersection ordinates in this interval. (Actually this is a ***multi-set*** because multiple equal ordinates are possible.) We will maintain the invariant that the k-th smallest intersection ordinate is in $I(x_{lo}, x_{hi}]$. Initially $x_{lo} = -\infty$ and $x_{hi} = +\infty$. Note that initially $I(x_{lo}, x_{hi}]$ includes all $\binom{n}{2}$ intersection ordinates because, by convention, no two lines intersect at $-\infty$.

The algorithm operates in a series of ***stages***. At the start of each stage the k-th smallest intersection ordinate lies within an interval $(x_{lo}, x_{hi}]$. We contract the interval into a smaller interval by randomly sampling from the set of intersection ordinates. Using this sample, we can find a subinterval $(x'_{lo}, x'_{hi}]$ which contains the k-th smallest ordinate. We choose this subinterval in such a way that, with high probability, the number of intersection ordinates in the subinterval is only $O(1/\sqrt{n})$ times the number of ordinates in the original interval (note that n here is the number of initial data points, not the number of ordinates in the interval). Since the number of intersection ordinates in the initial interval is $O(n^2)$ it follows that (with high probability) after two stages the number of remaining intersection ordinates in the interval will drop to $O(n)$. At this point we will be able to simply enumerate all of the remaining ordinates in $O(n \log n)$ time and use any standard selection algorithm to find the desired element. Each stage will take $O(n \log n)$ deterministic time, but the number of stages may vary due to randomization.

In order to describe the algorithm, we have to describe (1) how to count the number of intersection ordinates in an interval efficiently, (2) the sampling procedure, and (3) the strategy for selecting successive subintervals. The first two items are discussed below and the third item is discussed in the next section. To simplify the presentation, we will make the following ***general position*** assumptions.

- No two intersection ordinates are equal to one another.
- No two lines intersect the vertical lines $x = x_{lo}$ or $x = x_{hi}$ at the same

y-coordinate.

Given the second assumption, it does not matter whether we consider interval $(x_{lo}, x_{hi}]$ to be open or closed, but we retain this notation because in Section 5 we will discuss how to augment the algorithm to handle these degeneracies properly.

2.1. Counting Intersection Ordinates

First we consider how to count the number of intersection ordinates in the interval $(x_{lo}, x_{hi}]$. For the purpose of this discussion, assume x_{lo} and x_{hi} are finite; the cases $x_{lo} = -\infty$ and $x_{hi} = +\infty$ are considered in Section 5.1. We begin by associating each line with the y-coordinate of its intersection with the vertical line passing through the interval's *left* endpoint. Thus, the line $y = a_i x - b_i$ is associated with the value $a_i x_{lo} - b_i$. We sort the lines in increasing order by these y-coordinates. Next we replace each entry in this sorted list with the y-coordinate of the intersection of the line with the vertical line passing through the *right* endpoint of the interval. Thus the line $y = a_i x - b_i$ is associated with the value $y_i = a_i x_{hi} - b_i$. Let P denote the resulting list.

Define an ***inversion*** in P to be a pair of values, y_i and y_j where y_i is greater than y_j but $i < j$. If y_i and y_j are inverted in P, then the relative orders in which the corresponding lines have intersected the two vertical lines $x = x_{lo}$ and $x = x_{hi}$ are reversed. This means that the lines corresponding to y_i and y_j have intersected somewhere between $x = x_{lo}$ and $x = x_{hi}$. It follows that the number of inversions in P is exactly equal to the number of intersection ordinates that fall within the interval $(x_{lo}, x_{hi}]$. (See Figure 1.)

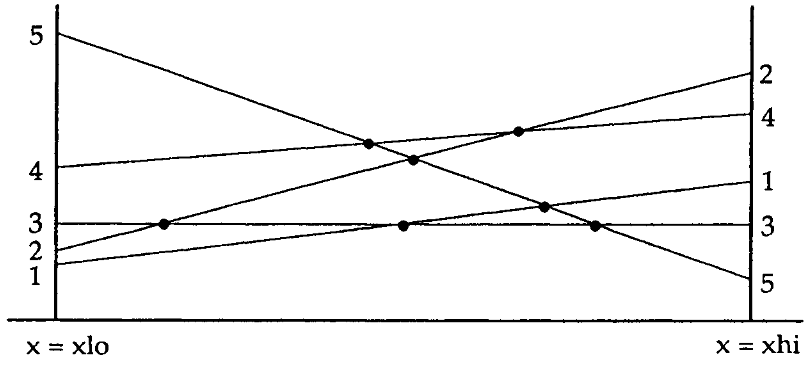

Figure 1: Inversions and line intersections.

Thus, in order to count the number of intersections in the interval $(x_{lo}, x_{hi}]$ it suffices to count the number of inversions in the list P. We describe a function Ord_Count which returns the number of intersection ordinates between x_{lo} and x_{hi}. It invokes the procedure Inv_Count (described later) to count the inversions in a list. The global variable n is the number of points and the global arrays A and B

contain the coefficients of the line equations $y = a_i x - b_i$. The integer parameter C keeps a running count of the number of inversions.

```
function Ord_Count($x_{lo}, x_{hi}$);   (* Count intersection ordinates in $(x_{lo}, x_{hi})$ *)
begin
    for $i := 1$ to $n$ do $P[i] := A[i] * x_{lo} - B[i]$;        (* evaluate at $x = x_{lo}$ *)
    Sort $P$ and permute $A$ and $B$ in parallel;
    for $i := 1$ to $n$ do $P[i] := A[i] * x_{hi} - B[i]$;        (* evaluate at $x = x_{hi}$ *)
    $C := 0$;
    Inv_Count($P, 1, n, C$);
    return($C$)
end;
```

The problem of counting the number of inversions in a list is closely related to sorting, and is discussed by Knuth.[18] We describe a simple adaptation of mergesort that solves this problem in $O(n \log n)$ time. Assume that recursively, we wish to sort the subarray $P[l..u]$ of reals. Assuming that $u > l$, let $m = \lfloor (l+u)/2 \rfloor$. We divide the list into left and right subarrays $P[l..m]$ and $P[m+1..u]$ and sort these subarrays recursively and at the same time count the number of inversions within each sublist. Finally these two sorted subarrays are merged together into a single sorted list. In the process of merging for each j, $m < j \leq u$, the j-th element in the right sublist implicitly discovers the index i of the next larger element in the left sublist. It follows that each of the $m - i + 1$ larger elements in the left sublist induces an inversion with the j-th element of the right sublist, and so we increase the inversion counter by $m - i + 1$. Equivalently the line associated with the j-th element in the right sublist intersects each of the lines associated with the k-th elements of the left sublist for $i \leq k \leq m$ in the vertical strip $x_{lo} < x \leq x_{hi}$. As in mergesort, this algorithm terminates in $O(n \log n)$ time.

The procedure Inv_Count which counts the number of inversions in the portion of the array $P[l..u]$ of reals is described below. It calls the procedure Merge which merges two sorted lists $P[l..m]$ and $P[m+1..u]$ and counts inversions. A parallel auxiliary array $Aux[l..u]$ is used to hold the sorted output. Proper handling of ties in sorting is discussed in Section 5. (The post-increment operator i++ returns the current value of i and then increments the variable.)

```
procedure Inv_Count($P, l, u, C$);          (* Count the inversions in $P[l..u]$ *)
begin
    if $l = u$ then return;
    $m := (l + u)$ div 2;
    Inv_Count($P, l, m, C$);
    Inv_Count($P, m + 1, u, C$);
    Merge($P, l, m, u, C$)
end;
procedure Merge($P, l, m, u, C$);           (* Merge $P[l..m]$ and $P[m + 1..u]$ *)
begin
    $i := l$;      $j := m + 1$;      $h := l$;
    while ($i \leq m$ and $j \leq u$) do begin
```

```
            if ($P[i] \leq P[j]$) then                          (* copy from left *)
                $Aux[h{+}{+}] := P[i{+}{+}]$
            else begin        (* copy from right and increment inversion count *)
                $Aux[h{+}{+}] := P[j{+}{+}]$;
                $C := C + (m - i + 1)$
            end
        end
        while ($j \leq u$) do $Aux[h{+}{+}] := P[j{+}{+}]$;
        while ($i \leq m$) do $Aux[h{+}{+}] := P[i{+}{+}]$;
        copy $Aux[l..u]$ to $P[l..u]$
    end;
```

The Merge procedure as presented here suffers from the deficiency of having to copy back the contents of the auxiliary array to P after each recursive call. In the actual implementation, this is avoided by using two working arrays and switching between them at alternate levels of the recursion.

2.2. Sampling Intersection Ordinates

In order to select the next candidates for x_{lo} and x_{hi}, we will need to do more than simply count the number of intersection ordinates—we will need to uniformly sample a subset of $I(x_{lo}, x_{hi}]$. Suppose that we have already counted the number of intersection ordinates in $I(x_{lo}, x_{hi}]$. Let C be this count. We apply the following adaptation of the above procedure to sample (with replacement) a subset of n intersection ordinates. First we generate a set IS of n random integers distributed uniformly in the range from 1 to C (allowing duplicates), and sort these integers. Since C is $O(n^2)$ sorting can be done in $O(n)$ time by two passes through radix sort.[1]

Next, we run the mergesort algorithm of the preceding section again but instead of counting intersections, for each $IS\_Indx \in IS$, we sample the IS_Indx-th intersection ordinate. To perform this sampling we modify the inversion-counting procedure as follows. Each time we increase the inversion counter in procedure Merge, we have discovered that $P[j]$ induces an inversion with each of $P[i']$, $i \leq i' \leq m$. This corresponds to the action of increasing the inversion counter C by some amount $\Delta C = m - i + 1$. We check the list of random integers for elements in the range $C + 1$ to $C + \Delta C$ and select for the sample the x-coordinate at which the corresponding pairs of lines intersect. Recall that two lines $y = a_i x - b_i$ and $y = a_j x - b_j$ intersect at the x-coordinate $x = (b_j - b_i)/(a_j - a_i)$.

The ordinate sampling procedure, Samp_Ord(x_{lo}, x_{hi}, IS) is essentially the same as the inversion-counting procedure given earlier, but the statement $C := C + (m - i + 1)$ is replaced with the code given below. The argument IS_Indx holds the next index to be sampled from the global array IS. Initially $IS\_Indx := 1$. We assume a sentinel value of $C + 1$ is stored in the location $IS[n + 1]$.

```
procedure Merge2($P, l, m, u, C, IS\_Indx$);
    (* Same as Merge but replace the statement $C := C + (m - i + i)$ with: *)
begin
    ...
    $new\_C := C + (m - i + 1)$;
    while ($IS[IS\_Indx] \leq new\_C$) do begin
        $i' := i + IS[IS\_Indx] - C$;
        Add $x := (B[j] - B[i'])/(A[j] - A[i'])$ to the sample;
        $IS\_Indx$++
    end;
    $C := new\_C$;
    ...
end;
```

The total running time of the sampling procedure is $O(n \log n)$. Because each selected pair of lines induces an inversion, these intersection points will all lie within the interval $(x_{lo}, x_{hi}]$ (recalling our assumptions about general position).

3. Contracting the Interval

In the previous section we introduced the necessary counting and sampling tools. In this section we introduce the necessary probability-theoretic groundwork on which the algorithm is based. Recall that our task has been transformed to that of computing the k-th smallest intersection ordinate in the set $I(x_{lo}, x_{hi}]$. (Note that the value of k may not be the same as the algorithm's original input.) The set $I(x_{lo}, x_{hi}]$ may contain $O(n^2)$ elements, but we have an $O(n \log n)$ procedure that counts the number of intersection ordinates in the set, and we have an $O(n \log n)$ procedure which samples n intersection ordinates (with replacement) at random. In this section we show how to use the two procedures to contract the interval into a smaller subinterval $(x'_{lo}, x'_{hi}]$ which still contains the k-th smallest intersection ordinate.

We assume that we have applied the counting procedure of the previous section to determine the number of intersection ordinates C in $I(x_{lo}, x_{hi}]$. We will assume that $C \geq n$. The case when C is smaller will be discussed in the next section. Throughout this section we will make the same general position assumptions of the previous section that the intersection ordinates in $I(x_{lo}, x_{hi}]$ are distinct. Handling degenerate cases in which this condition is violated is discussed in Section 5.

We begin by applying the sampling procedure of Section 2.2 to sample, with replacement, a subset of n intersection ordinates from $I(x_{lo}, x_{hi}]$. Let $S[1], S[2], \ldots, S[n]$ denote the elements of the sample, sorted in increasing order, and let k^* denote the nearest integer to kn/C. We claim that $S[k^*]$ should be "close" in some sense to the desired intersection ordinate in the original set. This can be informally justified as follows. Let x^* denote the (unknown) k-th smallest intersection ordinate in $I(x_{lo}, x_{hi}]$. Whenever we select a random element from $I(x_{lo}, x_{hi}]$ (assuming replacement), with probability k/C we select an intersection ordinate that is less than or equal to x^*. If we think of such an event as a "success," then it is as if we are performing n independent Bernoulli trials, each with a probability of success equal to $p = k/C$. Let m denote the number of successes in n trials. The value m is a binomial random variable with the well known probability density function

$$b(m; n, p) = \binom{n}{m} p^m q^{n-m} \qquad (q = 1 - p).$$

The mean of this distribution is $\mu = pn = kn/C$, and the standard deviation is $\sigma = \sqrt{npq} \leq \sqrt{n}/2$. Since we expect kn/C elements of the sample to be less than or equal to x^*, the k^*-th smallest element of the sample is a good estimator for x^*.

Although it is quite unlikely that the k^*-th smallest element of the sample will actually equal x^*, we can use the sample to place a bound on the location of x^* with high probability. It is well known that when n is large and p is not too close to zero or one, the binomial distribution can be approximated by the normal distribution. A large portion of the normal distribution lies within a few standard deviations of the mean. In particular, the probability that a given point sampled from a normal distribution lies within, say, t standard deviations of the mean rapidly approaches unity as t increases

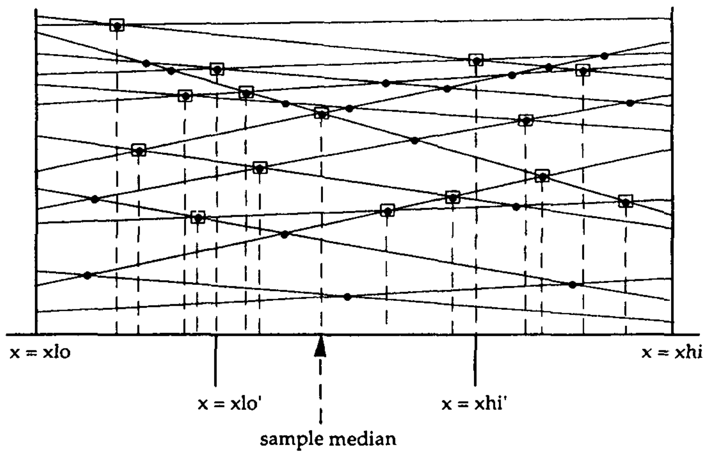

Figure 2: Illustration of the sampling procedure.

For some positive constant t (whose exact value will be discussed in Section 5) we define k_{lo} and k_{hi} to be indices of the ordinates in the sorted sample which lie at least t standard deviations on either side of the mean:

$$\begin{aligned} k_{lo} &= \left\lfloor \frac{kn}{C} - t\frac{\sqrt{n}}{2} \right\rfloor \\ k_{hi} &= \left\lceil \frac{kn}{C} + t\frac{\sqrt{n}}{2} \right\rceil . \end{aligned}$$

(Notice that we do not use the exact value of the standard deviation of $\sqrt{npq}$ but the upper bound of $\sqrt{n}/2$. As we will see in the proof of Lemma 3.1 below, the use of this bound provides more robust performance when p is close to zero or one.) If $k_{lo} \geq 1$ and $k_{hi} \leq n$ then define

$$\begin{aligned} x'_{lo} &= S[k_{lo}] \\ x'_{hi} &= S[k_{hi}]. \end{aligned}$$

If $k_{lo} < 1$ then let $x'_{lo} = x_{lo}$ and if $k_{hi} > n$ let $x'_{hi} = x_{hi}$. The process is illustrated in Figure 2. (Observe that since both x'_{lo} and x'_{hi} each coincide with at least one intersection ordinate, our general position assumptions will be violated since the two lines intersecting above this ordinate will generate a degeneracy at the next stage. This will be addressed in Section 5.)

We claim that as the constant t increases, the probability that x^* is located between x'_{lo} and x'_{hi} rapidly approaches unity. To justify this claim formally we will make use of two theorems from probability theory: Chebyshev's inequality[11] and Chernoff's bounds on the sum of binomial probabilities (Ref 10, p. 18). At this point, the presentation becomes significantly more technical, and the casual reader may wish to skip to the next section.

- Chebyshev's inequality: Let X be a random variable with mean $\mu = E(X)$ and variance $\sigma^2 = \mathrm{Var}(X)$. Then for any $r > 0$

$$Pr(|X - \mu| \geq r) \leq \frac{\sigma^2}{r^2}.$$

- Chernoff's bounds: (This formulation is given in (Ref. 2)). For any positive integer n and any reals $0 \leq p \leq 1$ and $0 \leq \beta \leq 1$

$$\begin{aligned} \sum_{m \leq (1-\beta)np} b(m; n, p) &\leq \exp(-\beta^2 np/2) \\ \sum_{m \geq (1+\beta)np} b(m; n, p) &\leq \exp(-\beta^2 np/3). \end{aligned}$$

  We assume the convention that $b(m; n, p) = 0$ for $m > n$ or $m < 0$.

Lemma 3.1 *Let $I(x_{lo}, x_{hi}]$ be the set of $C \geq n$ intersection ordinates between x_{lo} and x_{hi} from which we wish to select the k-th smallest. Let S be a random sample of n intersection ordinates from $I(x_{lo}, x_{hi}]$, and let $p = k/C$. For any $\epsilon > 0$, we can select t such that for all sufficiently large n the probability that the k-th smallest element of $I(x_{lo}, x_{hi}]$ lies within the subinterval $I(x'_{lo}, x'_{hi}]$ is at least $1 - \epsilon$. Furthermore, we can choose t so that, as a function of ϵ, $t \in O(\sqrt{\ln(1/\epsilon)})$.*

Proof. Let x^* denote the k-th smallest element of $I(x_{lo}, x_{hi}]$. As we argued earlier, the number of sampled elements that are less than or equal to the k-th smallest element is a binomial random variable X with mean $\mu = np$ and standard deviation $\sigma = \sqrt{npq}$ where $q = 1 - p$. By symmetry, it suffices to consider the case when $p \leq 0.5$ (for otherwise, we can imagine we are looking for the $(C - k)$-th largest element instead).

Let P_{lo} denote the probability that x^* is less than x'_{lo}, and let P_{hi} denote the probability that x^* is greater than x'_{hi}. Clearly x^* is less than x'_{lo} if and only if fewer than k_{lo} sampled elements are less than x^*, and x^* is greater than x'_{hi} if and only if more than $n - k_{hi}$ sampled elements are greater than x^*. The respective probabilities of these events are

$$\begin{aligned} P_{lo} &= \sum_{m<k_{lo}} b(m;n,p) \le \sum_{m\le k_{lo}} b(m;n,p), \text{ and} \\ P_{hi} &= \sum_{m>k_{hi}} b(m;n,p) \le \sum_{m\ge k_{hi}} b(m;n,p). \end{aligned}$$

Case 1: If $p > t/(2\sqrt{n})$ then letting $\beta = t/(2p\sqrt{n})$ we have

$$\begin{aligned} (1-\beta)np &= \left(1 - \frac{t}{2p\sqrt{n}}\right) np \\ &= np - \frac{t\sqrt{n}}{2} \\ &= \frac{kn}{C} - \frac{t\sqrt{n}}{2}. \end{aligned}$$

Thus $\lfloor (1-\beta)np \rfloor = k_{lo}$, and similarly $\lceil (1+\beta)np \rceil = k_{hi}$. Under our hypothesis it is easy to see that $0 \le \beta \le 1$, thus we can apply Chernoff's bounds giving

$$\begin{aligned} P_{lo} &\le \exp(-\beta^2 np/2) = \exp(-t^2/(8p)) \le \exp(-t^2/4) \\ P_{hi} &\le \exp(-\beta^2 np/3) = \exp(-t^2/(12p)) \le \exp(-t^2/6). \end{aligned}$$

The probability that the k-th smallest element lies between x'_{lo} and x'_{hi} is just one minus the sum of these two probabilities,

$$1 - (P_{lo} + P_{hi}) \ge 1 - \exp(-t^2/4) - \exp(-t^2/6) \ge 1 - 2\exp(-t^2/6).$$

Thus, by selecting

$$t \ge \sqrt{6 \ln(2/\epsilon)}, \qquad (*)$$

we achieve the desired probability bound in this case for all n.

Case 2: If $p \le t/(2\sqrt{n})$ then

$$\sigma^2 = npq \le np \le \frac{t\sqrt{n}}{2}.$$

The probability that the k-th smallest element lies between x'_{lo} and x'_{hi} is at least the probability that the number of sampled successes differs from the mean of np by no more than $t\sqrt{n}/2$. By Chebyshev's inequality we have

$$\begin{aligned} 1 - (P_{lo} + P_{hi}) &\ge 1 - \frac{\sigma^2}{(t\sqrt{n}/2)^2} \\ &\ge 1 - \frac{t\sqrt{n}/2}{(t\sqrt{n}/2)^2} \\ &= 1 - \frac{2}{t\sqrt{n}}. \end{aligned}$$

Thus, by chosing t to satisfy inequality $(*)$ above, and then selecting n such that $\sqrt{n} \geq 2/(t\epsilon)$ the probability of trapping the k-th smallest element in all cases is at least $(1-\epsilon)$. □

Thus given any prespecified probability bound, we can select a large enough sample range such that this range contains the k-th smallest element, x^*, with at least the desired probability. Of course we face a tradeoff since as t increases the size of the bounding interval, and hence the running time of the algorithm, also increases. In Section 5 we discuss the choice of parameter t.

To complete the probability theory needed to justify our claims on the algorithm's running time, we need to consider the number of intersection ordinates that remain to be processed in the subinterval $(x'_{lo}, x'_{hi}]$. Because the subsample between x'_{lo} and x'_{hi} represents essentially $t\sqrt{n}$ elements of the n elements sampled, we would expect that of the original C elements in $I(x_{lo}, x_{hi}]$, a fraction of about $t\sqrt{n}/n = t/\sqrt{n}$ would lie in the subinterval. The following result and its corollary state that for any constant probability bound, we can assert that there are asymptotically $O(Ct/\sqrt{n})$ ordinates within the subinterval with at least this probability. The constant on the asymptotic bound depends on the probability bound.

Lemma 3.2 *Consider the same hypotheses of Lemma 3.1. Given any $\epsilon > 0$ and any constant $d > t$, for all sufficiently large n the probability that more than $dC/\sqrt{n}$ elements of $I(x_{lo}, x_{hi}]$ lie within the open subinterval (x'_{lo}, x'_{hi}) is at most ϵ.*

Proof. Define a sample of intersection ordinates to be *good* if no more than $dC/\sqrt{n}$ elements of $I(x_{lo}, x_{hi}]$ lie within the open subinterval (x'_{lo}, x'_{hi}), and define a sample to be *bad* otherwise. Suppose that a sample is bad, containing $C' > dC/\sqrt{n}$ ordinates in the open subinterval. Since each sample point is generated independently with replacement we know that, excluding x'_{lo} and x'_{hi} (which are part of the sample), the remaining $n-2$ sampled ordinates form a random sample of $I(x_{lo}, x_{hi}]$ where each ordinate has an equal probability of being sampled. We also know that the number of sampled ordinates in the interval (x'_{lo}, x'_{hi}) is at most $k_{hi} - k_{lo} - 1 \leq \lceil t\sqrt{n} \rceil$.

Thus the probability of generating a bad sample is no greater than the probability of generating at most $\lceil t\sqrt{n} \rceil$ sample ordinates in the interval (x'_{lo}, x'_{hi}) given that there are C' ordinates within the subinterval. (To be formally correct here we should also factor in the additional knowledge that there are at least $k_{lo} - 1$ sample elements less than x'_{lo}, but since the sampling is uniform, this does not affect the conditional probability.) If we consider each sampled ordinate in the interval (x'_{lo}, x'_{hi}) to be a success and each other sampled ordinate to be a failure, we see that the probability of generating a bad sample is equal to the probability of generating at most $\lceil t\sqrt{n} \rceil$ successes in $n-2$ trials, where the probability p of success is $C'/C > d/\sqrt{n}$. This is the tail of a binomial distribution.

Since $d > t$ we can select a constant $\delta > 0$ such that $(1-\delta)d > (1+\delta)t$. Let

$$\beta = 1 - \frac{(1+\delta)t}{(1-\delta)d}.$$

Clearly $0 < \beta < 1$. For all sufficiently large n, $n-2 \geq (1-\delta)n$ and $\lceil t\sqrt{n} \rceil \leq$

$(1+\delta)t\sqrt{n}$. Thus

$$\begin{aligned}(1-\beta)(n-2)p &\geq (1-\beta)(1-\delta)n(d/\sqrt{n})\\ &\geq (1-\beta)(1-\delta)d\sqrt{n}\\ &\geq (1+\delta)t\sqrt{n}\\ &\geq \lceil t\sqrt{n}\rceil .\end{aligned}$$

Thus the probability of generating a bad sample is no greater than the probability of generating at most $(1-\beta)(n-2)p$ successes in $(n-2)$ trials with probability p of success. By Chernoff's bounds this probability is bounded above by

$$\sum_{m\leq(1-\beta)(n-2)p} b(m;n-2,p) \leq \exp(-\beta^2(n-2)p/2).$$

Since β is a constant and $(n-2)p$ is $O(\sqrt{n})$ this probability rapidly tends to zero as n increases. For any ϵ, there is $n_0 \in \Omega(\ln^2(1/\epsilon))$, such that for all $n \geq n_0$ the lemma is satisfied. □

This is not exactly the desired result, because it only bounds the number of ordinates in the open subinterval, (x'_{lo}, x'_{hi}). However, under our general position assumptions, the elements of $I(x_{lo}, x_{hi}]$ are distinct, hence there can be at most one additional element by closing the right side of the interval.

Corollary *Given the same hypotheses of the previous lemma, with the additional constraint that the elements of $I(x_{lo}, x_{hi}]$ are distinct, the probability that more than $dC/\sqrt{n}$ elements of $I(x_{lo}, x_{hi}]$ lie within the half-open, half-closed subinterval $I(x'_{lo}, x'_{hi}]$ is at most ϵ.*

4. Complete Algorithm

At this point we have the necessary tools to describe the entire algorithm. As mentioned earlier, the algorithm consists of a series of stages. During each stage we are given an interval $(x_{lo}, x_{hi}]$ within which we seek the k-th smallest intersection ordinate. We also assume that we have been given a count C of the number of intersection ordinates in this interval, and we assume that $k \leq C$.

If C is sufficiently small, in particular, if $C \leq cn$, for some constant $c \geq 1$, we apply a simple enumeration algorithm to locate the k-th smallest value. (In Section 5 we discuss the choice of the constant c.) The enumeration algorithm is a simple modification to the inversion-counting procedure, which beyond counting inversions, generates a list of inversion ordinates. (This can be accomplished by modifying the sampling procedure to sample *every* ordinate in the interval.) After this, any selection algorithm can be applied to determine the k-th smallest value (e.g. Hoare's $O(n)$ expected time algorithm[1] modified to select the pivot element at random).

If, on the other hand, $C > cn$, we apply the sampling procedure described earlier to select a sample S of n intersection ordinates. As always let us assume that all the intersection ordinates are distinct. We will consider the degenerate case in Section 5.

We sort the sampled intersection ordinates, and set x'_{lo} to be the k_{lo}-th smallest and the set x'_{hi} to be k_{hi}-th smallest elements from the sample, where k_{lo} and k_{hi} were defined in the previous section. By Lemma 3.1, with high probability, we expect the k-th smallest intersection ordinate to lie within $I(x'_{lo}, x'_{hi}]$. To verify that this is the case, we apply the counting procedure twice to determine the number of elements in the left subinterval $I(x_{lo}, x'_{lo}]$ and the center subinterval $I(x'_{lo}, x'_{hi}]$. Let L and C' be these respective values. (The number of ordinates in the right subinterval $I(x'_{hi}, x_{hi}]$ can be derived as the remainder $C - L - C'$.) If $L < k \leq L + C'$, then indeed, the k-th smallest element lies within the desired subinterval, and we make a recursive call to locate the $(k - L)$-th smallest element in $I(x'_{lo}, x'_{hi}]$. In the rare case $k \leq L$ or $k > L + C'$, we either make a recursive call to find the k-th smallest element of the left subinterval $I(x_{lo}, x'_{lo}]$, or we make a recursive call to find the $k - (L + C')$-th smallest element of the right subinterval $I(x'_{hi}, x_{hi}]$.

The overall algorithm is described below. The global variable n contains the number of input points. The line coefficients are stored in the global arrays $A[1 \ldots n]$ and $B[1 \ldots n]$. $+INF$ and $-INF$ are special values whose intended interpretation is $+\infty$ and $-\infty$, and c and t are constants which are used to tune the performance of the algorithm (see Section 5 for details). The argument C contains the number of ordinates in $I(x_{lo}, x_{hi}]$, and k is the rank of the desired ordinate. The initial call is Select_Slope($-INF, +INF, \binom{n}{2}, k$).

```
function Select_Slope($x_{lo}, x_{hi}, C, k$);
begin
    if $C \leq cn$ then begin
        Enumerate the ordinates in $I(x_{lo}, x_{hi}]$;
        Select the $k$-th smallest by any selection algorithm;
        Return this ordinate
    end
    else begin
        Generate a sorted list of $n$ random numbers from 1 to $C$, $IS[1 \ldots n]$;
                                        (* Sample ordinates from $I(x_{lo}, x_{hi}]$ *)
        $S[1..n]$ := Samp_Ord($x_{lo}, x_{hi}, IS$);
        Sort $S[1..n]$;
        $k_{lo} := \max(1, \lfloor kn/C - t\sqrt{n}/2 \rfloor)$;        $x'_{lo} := S[k_{lo}]$;
        $k_{hi} := \min(n, \lceil kn/C + t\sqrt{n}/2 \rceil)$;        $x'_{hi} := S[k_{hi}]$;
        $L$ := Ord_Count($x_{lo}, x'_{lo}$);        (* Count left subinterval ordinates *)
        $C'$ := Ord_Count($x'_{lo}, x'_{hi}$);   (* Count center subinterval ordinates *)
        if $k \leq L$ then                          (* Search left subinterval *)
            Select_Slope($x_{lo}, x'_{lo}, L, k$)
        else if $k \leq L + C'$ then             (* Search center subinterval *)
            Select_Slope($x'_{lo}, x'_{hi}, C', k - L$)
        else                                      (* Search right subinterval *)
            Select_Slope($x'_{hi}, x_{hi}, C - (L + C'), k - (L + C')$)
    end
end;
```

To analyze the running time of this procedure notice that the only probabilistic aspect of the algorithm is the number of stages which the algorithm performs (the number of recursive calls to Select_Slope). Let $T(n, C)$ denote the expected running time of the procedure for n lines on any interval $(x_{lo}, x_{hi}]$ containing C intersection ordinates. We define a stage of the procedure to be *successful* if the k-th smallest intersection ordinate is located in the central interval $(x'_{lo}, x'_{hi}]$ (as expected), and if this central interval has no more than $dC/\sqrt{n}$ elements, where d is a constant to be defined later. The running time of each stage can be bounded above by $en \log n$ for some constant e, since as we have shown, each of the algorithm's basic tasks can be reduced to variations of mergesort. If all stages are successful, then the running time of the algorithm is $T(n, \binom{n}{2})$, where $T(n, C)$ is given by the recurrence

$$\begin{array}{rcll} T(n,C) & \leq & en \log n & \text{if } C \leq cn \\ T(n,C) & \leq & T\left(n, \frac{dC}{\sqrt{n}}\right) + en \log n & \text{otherwise.} \end{array}$$

After k successful stages, the remaining number of intersection ordinates is at most

$$\left(\frac{d}{\sqrt{n}}\right)^k \binom{n}{2} \leq \frac{d^k}{2} n^{2-(k/2)}.$$

Thus, if c and d are chosen such that $d^2/2 \leq c$, then after only two successful stages, the remaining number of intersection ordinates falls below cn. From Lemmas 3.2 and 3.1, by adjusting the parameters d and t (which in turn constrain the value of c) we can make the probability of a successful stage arbitrarily high. Because success is independent from one stage to another (depending only on the random number generator) we can select c and d sufficiently large so that the algorithm terminates within two stages with arbitrarily high probability (while simultaneously slowing the algorithm down by a constant factor).

The preceding considerations suggest the following mode of operating the algorithm: select t and d such that the probability of success of any one stage of the algorithm is very high, say 0.99, which implies that after two stages the algorithm terminates successfully with very high probability, say 0.98. In Section 5 we discuss the choice of these parameters for our implementation. The values of these parameters should be derived empirically, because the bounds provided by Lemmas 3.1 and 3.2 are rather pessimistic. Once t and d have been selected, c is selected so that $c \geq d^2/2$. As argued earlier, with high probability (depending on the choice of d and t) in two stages the number of intersection ordinates $I(x_{lo}, x_{hi}]$ will fall below cn, after which we apply the simple enumeration algorithm. If all goes as expected, the running time will be $O(n \log n)$.

Even if all stages are not successful, the *expected* running time of the procedure is still $O(n \log n)$, provided that d and t are chosen so that the probability p of a successful stage is bounded away from zero. If the stage is not successful, then we recurse on either the left or right subinterval. To obtain an upper bound we assume that in each unsuccessful stage no ordinates at all are eliminated (which is pessimistic, since at least $O(\sqrt{n})$ sample points will always be eliminated). If the stage is not successful the running time of the algorithm is given by the recurrence

$$\begin{aligned} E(n,C) &\leq en\log n && \text{if } C \leq cn \\ E(n,C) &\leq pE\left(n, \tfrac{dC}{\sqrt{n}}\right) + (1-p)E(n,C) + en\log n && \text{otherwise.} \end{aligned}$$

It is straightforward to prove by induction that $E(n,C)$ satisfies

$$E(n,C) \leq (en\log n)\max\left(1, \frac{\log C}{p\log(\sqrt{n}/d)}\right).$$

Since C is $O(n^2)$ the last factor is $O(1)$. Thus the algorithm's expected running time is $O(n\log n)$.

5. Implementation

In this section we describe a number of implementation issues concerning the algorithm that were left unspecified in the preceding presentation. In particular we consider the following issues:

- How to handle various degenerate and special cases. Degenerate cases arise, for example, when values to be sorted or ranked are equal to one another.
- How to deal with limited numerical precision.
- How to choose the constant parameters referred to in earlier sections which affect the probabilistic behavior and running time of the algorithm.

In addition to this we discuss a specific implementation of the algorithm which we have used to test the algorithm's performance on various data sets.

5.1. Degeneracies

Proper handling of special cases and degeneracies is an important aspect of the implementation of any geometric algorithm. Since the algorithm deals exclusively with lines, we could pass all the handling of degeneracies to a general purpose system, such as Edelsbrunner and Mücke's *simulation of simplicity* in which coincident line intersections are broken in a systematic manner.[8] Because of the restricted nature of degeneracies in this problem, and the fact that in application areas like computer vision, digitized degenerate cases are often the norm rather than the exception, we decided to process the degeneracies on a case by case basis for greater efficiency.

There are two places in the algorithm where degenerate and special cases can arise. The first is when we sort the dual lines according to the y-coordinate of their intersection with one of the vertical lines $x = x_{lo}$ or $x = x_{hi}$. Recall that each line is represented in slope and intercept form: $y = a_i x - b_i$. One special case which needs to be considered is when $x_{lo} = -\infty$ or $x_{hi} = +\infty$. At these extremes we simply sort the lines by decreasing slope for $-\infty$ and by increasing slope for $+\infty$. If two lines share the same slope, then we sort them by increasing y-intercept at $-\infty$ and by decreasing y-intercept at $+\infty$. By sorting in reverse order of y-intercept at $+\infty$

we insure that two parallel lines will induce an inversion in some interval $(x_{lo}, x_{hi}]$ if and only if $x_{hi} = +\infty$. (Recall our convention that two parallel lines intersect at $x = +\infty$.)

For finite values of x_{lo} and x_{hi} we still need to consider the possible degeneracy that two or more lines may intersect one of the vertical lines $x = x_{lo}$ or $x = x_{hi}$ at the same y-coordinate. In breaking ties we recall that we are interested in counting the number of intersections in the half-closed interval $(x_{lo}, x_{hi}]$. This is equivalent to imagining that we are sorting along the vertical lines $x = x_{lo} + \epsilon$ and $x = x_{hi} + \epsilon$, for some sufficiently small $\epsilon > 0$. If two lines intersect the vertical line $x = x_{lo}$ at the same y-coordinate, then their intersections with the line $x = x_{lo} + \epsilon$ (from bottom to top) will be in order of increasing slope. In other words, we sort lexicographically by the pair $\langle a_i x_{lo} - b_i, a_i \rangle$. Since we assume that the pairs (a_i, b_i) are distinct, this will break all ties. Similarly, in order to break ties along the vertical line $x = x_{hi}$, we sort lexicographically by the pair $\langle a_i x_{hi} - b_i, a_i \rangle$.

A second type of degeneracy that may arise is a multiple occurrence of the same intersection ordinate within $I(x_{lo}, x_{hi}]$. The problem here is considerably subtler. Suppose that we seek the median intersection ordinate within the interval $(x_{lo}, x_{hi}]$, and just over half of these ordinates are equal to x_{hi}. In sampling ordinates, we expect that about half will be equal to x_{hi}. For the next stage it is not unlikely that we will generate a new subinterval $(x'_{lo}, x'_{hi}]$ where $x'_{hi} = x_{hi}$. Because the interval is closed on the right side, we will fail to eliminate a single ordinate on that side of the median!

To handle this kind of degeneracy, we note that the problem arises only when the k-th smallest ordinate coincides with the right endpoint of the current interval. (There is not a problem on the left endpoint, since the interval is open on the left, so that degeneracies at x_{lo} are ignored.) In order to solve this problem, we alter our inversion-counting procedure so that instead of counting the number of intersection ordinates C in the half-open, half-closed interval $(x_{lo}, x_{hi}]$, we create two counts, one for the open interval (x_{lo}, x_{hi}), denoted C_1, and one for the single point x_{hi}, denoted C_2. Clearly $C = C_1 + C_2$. If $C_1 < k$, then we conclude that the k-th smallest intersection ordinate is x_{hi} and terminate (recall that the algorithm maintains the invariant $k \leq C$). Otherwise we apply our algorithm on the open interval (x_{lo}, x_{hi}) rather than on the half-closed interval $(x_{lo}, x_{hi}]$.

This two stage counting can be performed with very little additional effort. Rather than applying the inversion-counting routine to the half-closed interval $(x_{lo}, x_{hi}]$, we apply it only to the open interval (x_{lo}, x_{hi}). The only change to the procedure is to modify the tie-breaking rule given above so that we sort along the line $x = x_{hi} - \epsilon$, for some sufficiently small $\epsilon > 0$. This is equivalent to sorting lexicographically on the pair $\langle a_i x_{hi} - b_i, -a_i \rangle$. The resulting count is C_1. The intersection ordinates at x_{hi} can be counted in only $O(n)$ additional time as follows. We scan the sorted list of y-coordinates at which the lines intersect the vertical line $x = x_{hi}$. (This is the same as the sorted order along $x_{hi} - \epsilon$ which was just computed.) These y-coordinates can be grouped into *bunches* of equivalence classes in $O(n)$ time (by ignoring the second component of the lexicographical sort). Let

$m_1, m_2, \ldots, m_k$ denote the sizes of these bunches. Each bunch of size m corresponds to m lines intersecting at a common point with x-coordinate x_{hi}, implying that there are $m(m-1)/2$ intersection ordinates at $x = x_{hi}$. By computing the sum $\sum_{i=1}^{k} m_i(m_i+1)/2$ we have computed C_2, the number of degenerate ordinates at x_{hi}.

This use of an open interval does not affect any of the proofs of the algorithm's correctness. (The proof of Lemma 3.1 made no assumptions about the interval endpoints, and Lemma 3.2 assumed an open interval.) The asymptotic running time cannot deteriorate as a consequence, since processing the open interval can only reduce the number of ordinates which need to be considered at later stages of the algorithm. Our practical experience has shown that when the input data is perturbed by random noise, this modification has little effect on the algorithm's performance since the probability of degeneracies is small. In cases where the data is nearly exact and many degeneracies are present, the algorithm often terminates after only one iteration (rather than the expected two) because intersection ordinates bunch heavily around the median point.

5.2. *Limited Numerical Precision*

The proofs of the algorithm's correctness have been based on the assumptions that all calculations have been carried out exactly. In practice, where calculations are typically performed using floating point representation, this assumption may not be valid. In our algorithm there are two places where roundoff errors may affect the correctness of the final result. One situation occurs when computing the intersection ordinate for two lines, say $y = a_1x - b_1$ and $y = a_2x - b_2$:

$$x = \frac{b_1 - b_2}{a_1 - a_2}.$$

The second situation arises when computing the y-coordinate at which a line, $y = a_3x - b_3$, intersects either the vertical line $x = x_{lo}$ or $x = x_{hi}$. Since the values of x_{lo} and x_{hi} are themselves intersection ordinates, the typical y-coordinate results from an expression of the following form:

$$y = a_3\frac{b_1 - b_2}{a_1 - a_2} - b_3.$$

These y-values are not involved in further computations except for the purposes of comparisons in sorting.

If the input values are represented in fixed-point notation or as integers containing at most d bits of precision (a very reasonable assumption in the application area of computer vision where the input data consists of pixel indices), this computation can be restructured so that it is performed exactly with at most $2d+3$ bits of precision. In particular, each intersection ordinate is represented as a pair $\langle p, q\rangle$, where $p = b_1 - b_2$ and $q = a_1 - a_2$. Each of p and q requires at most $d+1$ bits. We can compare two intersection ordinates p_1/q_1 and p_2/q_2 by determining the sign of the difference $p_1q_2 - p_2q_1$, which requires at most $2(d+1)+1 = 2d+3$

bits. Suppose that one of the endpoints of the current interval, say x_{lo} is given by p/q. In order to sort the y-coordinates at which the lines intersect $x = p/q$, namely $y = a_3(p/q) - b_3$, we may equivalently sort the values $yq = a_3p - b_3q$, which requires at most $2d + 3$ bits.

If floating point calculations are performed throughout, the final output will be imprecise, but as mentioned above the degree of the relative error will be bounded because the depth of the computations is bounded. A more serious problem is that roundoff errors associated with floating point computations can cause the algorithm to fail altogether, because the inversion-counting and intersection sampling may be inconsistent. For example, the inversion-counting algorithm may determine that two lines $y = a_1x - b_1$ and $y = a_2x - b_2$ intersect within some interval $(x_{lo}, x_{hi}]$ but the computed intersection ordinate $x = (b_1 - b_2)/(a_1 - a_2)$ lies outside of this interval. (The error may have occurred in either computation.) This phenomenon has been observed on actual data, and in certain cases can cause the algorithm to loop infinitely because the interval size may oscillate.

One way to avoid this problem is to compute the intersection ordinate by a more robust formula so that if an inversion is detected, then the computed intersection ordinate must lie within the interval. This can be done by computing the x-ordinate as a weighted average between x_{lo} and x_{hi} where the relative weights are proportional to the differences in the y-coordinates at which the lines intersect $x = x_{lo}$ and $x = x_{hi}$. Let

$$\begin{aligned}\Delta lo &:= (a_1x_{lo} - b_1) - (a_2x_{lo} - b_2);\\ \Delta hi &:= (a_1x_{hi} - b_1) - (a_2x_{hi} - b_2).\end{aligned}$$

These equations should not be optimized algebraically. Notice that an inversion will be detected in the interval $(x_{lo}, x_{hi}]$ if and only if either $\Delta hi = 0.0$ exactly, implying that the intersection ordinate is determined to lie at $x^* = x_{hi}$, or else if Δlo and Δhi are both nonzero and have opposite signs. In the latter case observe that the relative distance of the intersection ordinate x^* between x_{lo} to x_{hi} is proportional to the ratio $R = \Delta lo/(\Delta lo - \Delta hi)$, which is guaranteed to be between 0 and 1 by our assumption that these quantities have different signs. (This is true for any reasonable model of floating point calculations.) Thus by computing x^* by the following formula we will generate an intersection point between x_{lo} and x_{hi}:

$$\begin{aligned}R &:= \Delta lo/(\Delta lo - \Delta hi);\\ x^* &:= x_{lo} + R * (x_{hi} - x_{lo}).\end{aligned}$$

Although the intersection ordinates may be computed to lower accuracy than before (because their computation depends on floating point quantities Δlo and Δhi), they will be consistent with the inversion sampling in the sense that if an inversion is generated then the sampled ordinate will lie within the interval $(x_{lo}, x_{hi}]$. Even so, when the size of the interval $(x_{lo}, x_{hi}]$ becomes very small (relative to floating point precision) the algorithm may fail to contract the interval at all, or may contract it only slightly. We suggest that if floating point calculations are used, then an extra test should be included which terminates the algorithm if either the interval size shrinks to within some prespecified accuracy or if the number of intersection ordinates does not decrease by at least one half over two successful iterations.

5.3. Choice of Parameters

In Section 4 we mentioned that there is a tradeoff between the constant factors in the algorithm's running time and the probability that each stage of the algorithm is successful. The constants on which this tradeoff is based are: (1) t, which is used in the definition of k_{lo} and k_{hi} (see Lemma 3.1); (2) c, the factor such that when the number of intersection ordinates falls below cn we simply enumerate the remaining inversions; and (3) d, the factor such that $dC/\sqrt{n}$ is the bound on the number of remaining ordinates after a successful stage (see Lemma 3.2). The value of d is really not used by the algorithm (only in its analysis) but its value constrains the choices of c and t. It was shown in Section 4 that the algorithm requires only two stages, provided that c and d satisfy $c \geq d^2/2$. In Lemma 3.2 it was assumed that d was strictly larger than t.

Although in theory we could apply Lemmas 3.1 and 3.2 directly to determine bounds on t and c which would guarantee termination in two stages with some specified probability, the resulting bounds would be pessimistic because these lemmas do not provide tight bounds. To see why this is, observe that (1) we are always dealing with reasonably large n (e.g. $n \geq 50$), since for small n we can simply apply the $O(n^2)$ brute force algorithm, and (2) the values of p tend to be close to 0.5 in many instances because we select each subinterval symmetrically about the expected location of the k-th smallest element. For large n and when p is reasonably close to 0.5, Chernoff's bounds are somewhat pessimistic. Tighter bounds may be achievable by better approximating functions, such as that given by the DeMoivre-Laplace Theorem with error terms (Ref. 31, p. 129). However, we chose instead to use our experimental experience with the implementation to (conservatively) determine suitable values.

In our experiments with the algorithm, which involved over 1,000 invocations of the algorithm on input sizes n ranging from 50 to 50,000, we observed that setting $t = 3$ was sufficient to capture the k-th smallest intersection ordinate in over 99% of the cases run. The reason is as follows. If p is near 0.5, then the normal distribution is a good approximation to the binomial distribution, and three standard deviations from the mean captures most of the area under the normal curve. On the other hand, if p is very close to 0 or 1, then the standard deviation $\sigma = \sqrt{npq}$ is much smaller than the upper bound of $\sqrt{n}/2$ which we use in the algorithm. So even though the normal distribution is a much poorer approximation for the skewed binomial distribution, we are taking a much larger number of standard deviations on either side of the mean.

Our next task is to determine the value of c, such that when the number of intersection ordinates falls below cn we switch to the simple enumeration algorithm. We face the following tradeoff. By iterating one more time, we incur the additional cost of sampling and counting, but we then can apply enumeration to a smaller set of points. Thus, the value of c depends on the relationship between the running times of each inversion-counting phase and the enumeration phase. Let $en(n, C)$ denote the running time needed to determine the k-th smallest intersection ordinate out of the remaining C ordinates by simple enumeration. Let $inv(n)$ denote the running time needed for one stage of basic inversion sampling and counting (this quantity is independent of C). Our goal is to determine the maximum c, such that for all $C \leq cn$ the time needed to solve the problem by brute force enumeration is less than the expected time needed to iterate through one more stage followed by an enumeration on a smaller list. Because the expected decrease in the number of intersection ordinates is $3/\sqrt{n}$, the choice of c should satisfy

$$en(n, C) \leq inv(n) + en\left(n, \frac{3C}{\sqrt{n}}\right).$$

The running time for each iteration is $O(n \log n)$, and the running time for enumeration is $O(n \log n + C)$. We used the following models for the running time of these two parts of the algorithm:

$$\begin{aligned} inv(n) &= I_1 n \log n + I_2 n, \\ en(n, C) &= E_1 n \log n + E_2 n + E_3 C, \end{aligned}$$

where I_1, I_2, and E_1, E_2, E_3 are constants depending on the particular implementation. These values can be derived empirically. Substituting the model into the equation above yields

$$C\left(1 - \frac{3}{\sqrt{n}}\right) E_3 \leq I_1 n \log n + I_2 n.$$

By setting C to cn we have

$$c \leq \frac{I_1 \log n + I_2}{E_3\left(1 - \frac{3}{\sqrt{n}}\right)}.$$

We timed the inversion-counting algorithm and the enumeration algorithm with n ranging from 100 to 10,000 and with values of C varying between $n/2$ and $40n$. Based on these running times (in milliseconds), we used least squares to fit running times to the models above and determined that $I_1 \approx 0.3$, $I_2 \approx 0.8$ and $E_3 \approx 0.09$. As a function of n, c is rather flat, ranging between 20 and 30 over all "practical" values of n $(50 \leq n \leq 100,000)$. We conclude that $c = 20$ is a conservative choice. It should be mentioned that it may be advantageous to make c smaller if memory is tight at the potential expense of running an extra iteration. The reason is that cn words of memory must be used for the enumeration. Our experience is that for $n \geq 100$, when $c = 10$ the algorithm failed to terminate after two iterations only when it failed to trap the k-th smallest intersection by sampling. Finally, note that the expected number of ordinates remaining after two iterations is $9n/2$ (assuming that $t = 3$), so values of c smaller than, say 7, have a greater risk of running a third iteration.

5.4. Experimental Results

We ran the algorithm on a number of data sets in order to study the overall performance of the algorithm and also to study the statistical behavior of various parameters on which the running time depends. In order to establish a certain degree of statistical validity to the empirical results obtained, we experimented with numerous sets of data. We ran two types of experiments: a *standard* experiment in which the algorithm was run over data sets which we felt were likely to arise in practice, and a *special case* experiment in which the algorithm was run on more skewed data sets. The following input descriptions are stated in terms of the original problem of selecting the k-th smallest slope determined by a set of n points (as opposed to the dual problem of selecting the k-th smallest intersection ordinate).

The standard experiment considered nearly 100 different data instances for each value of n. In each case a set of n points was generated inside a unit square with respect to a specific line equation, characterized by its slope and vertical intercept. To avoid degeneracies we perturbed the y-coordinate of each point by adding to it the value of a Gaussian randomly generated variable multiplied by some specified standard deviation, σ. We used the algorithm of (Ref. 17, pp. 116–117) to produce these values. Notice that $\sigma = 0$ corresponds to an "exact fit case" which is considered in the special case experiment below. For each data instance, we invoked our algorithm to find the 50% quantile, an approximation to the median element, and recorded the running time, and the values obtained for the various parameters of interest.

For comparison, the running times of the brute force algorithm (which simply enumerates all $O(n^2)$ pairs of points and selects the k-th smallest by Hoare's selection algorithm[1]) have also been recorded. This could be done only for data instances where $n \leq 1000$ because of the limitations of main memory and the fact that this algorithm needs $O(n^2)$ space. Letting $T_I(n)$ denote the running time of our inversion-counting algorithm and $T_B(n)$ denote the running time of the brute force algorithm, we used least squares to derive the following linear models for the running times of these algorithms in milliseconds. The running times have been scaled down by a factor of n to reduce the residuals for large n.

$$\begin{aligned} \frac{T_I(n)}{n} &= 0.7\log_{10} n + 5.2, \\ \frac{T_B(n)}{n} &= 0.03n + 2.5. \end{aligned}$$

These results are summarized in Figure 3. Logarithms are base 10. Figure 3(a) shows $T_B(n)/n$, (b) shows $T_I(n)/n$, and (c) shows both fits superimposed. From these models we conclude that for values of n greater than $10^{2.13} \approx 135$ our algorithm performs better than the brute force algorithm.

The other purpose of the standard experiments was to investigate the statistical behavior of some of the algorithm's parameters. It was mentioned in Lemma 3.2 that between successive iterations of the inversion-counting algorithm the number of intersection ordinates is expected to decrease by a factor of $3/\sqrt{n}$. Thus if C and C'

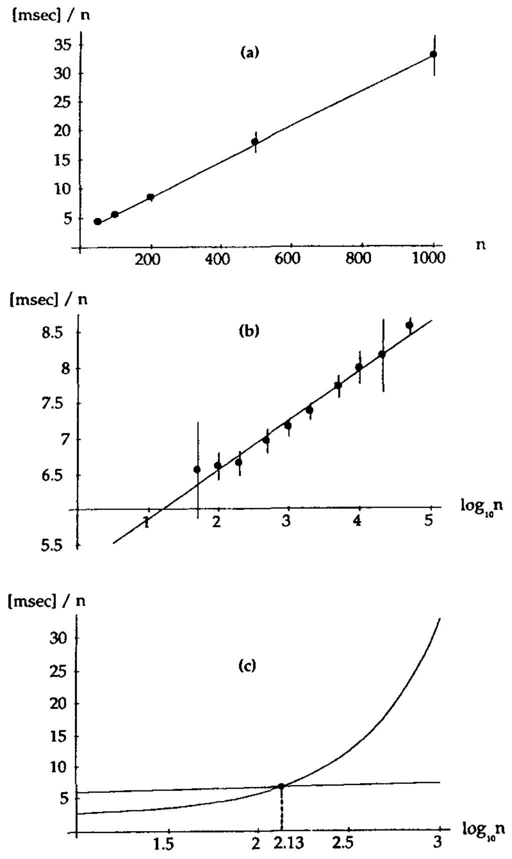

Figure 3: Comparison of inversion counting and brute force algorithms.

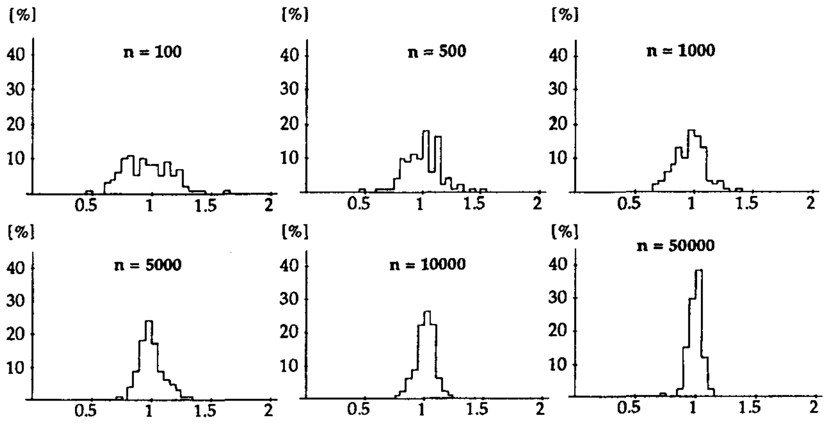

Figure 4: Histograms of the ratio $(C''/C) : (9n)$.

represent the number of intersection ordinates before and after sampling, we would expect that the ratio $(C'/C) : (3/\sqrt{n})$ should be close to 1, and the approximation should be closer for larger values of n. Letting C'' be the number of ordinates remaining after two iterations, we would expect that the ratio $(C''/C) : (9/n)$ should be close to 1. This ratio is important because it indicates how many intersection ordinates (above the expected $9n/2$) remain to be enumerated after two iterations. We recorded this ratio after every iteration that was successful in trapping the k-th smallest intersection ordinate by sampling. We found that this ratio never exceeded 2 for $n \geq 100$ and did not exceed 1.5 for $n \geq 1000$. Histograms of these ratios are shown in Figure 4 for values of n ranging from 100 to 50000. Even with skewed data sets (mentioned below) this ratio only tended to be smaller.

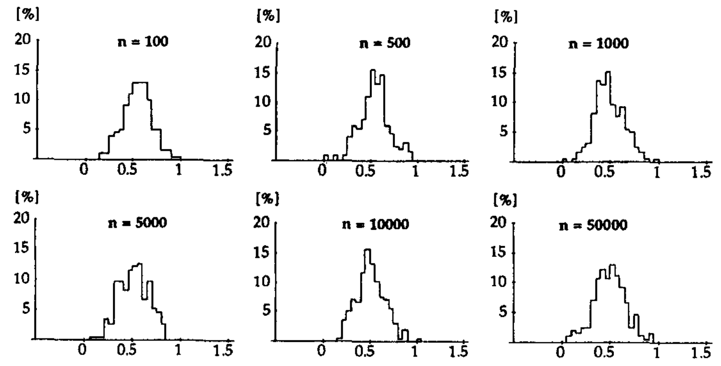

Figure 5: Histograms of relative ranks of the true k-th smallest ordinate.

It was mentioned earlier that one of the reasons that the algorithm performs faster than the bounds suggested in Lemmas 3.1 and 3.2 is that the k-th smallest intersection ordinate tends to lie in the middle of the interval $(x_{lo}, x_{hi}]$. This is of interest because the normal approximation to the binomial is best when the k-th smallest element is not too close to one endpoint or the other. We recorded the relative rank of the true k-th smallest element with respect to the contracted set of intersection ordinates $I(x_{lo}, x_{hi}]$ at the end of the first and second iterations for the standard experiment (where the median element was sought). This quantity is between 0 and 1 if the k-th smallest element lies within the interval, and it is equal to 0.5 if the k-th smallest element is the median element of the interval. Histograms for this statistic are shown in Figure 5 for values of n ranging from 100 to 50000. Notice that if k approaches the extreme values of 1 or $\binom{n}{2}$ the location of the k-th smallest element tends to be closer to the extremes of the interval during the early stages.

The special case experiments involved the invocation of our implementation to find various k-th smallest elements per each data instance in a number of instances which we felt would be less likely in practice (given our application of line fitting), but these cases are important in checking that the algorithm's performance does not vary significantly over its performance for nominal data sets. Letting $C_0 = \binom{n}{2}$

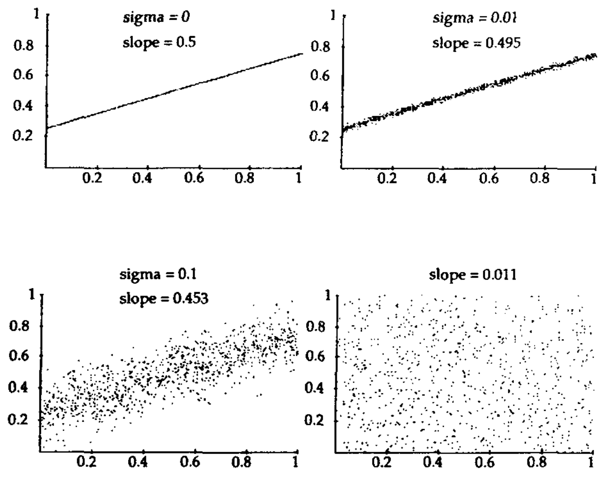

Figure 6: Examples of experimental data points.

denote the maximum range of k, we sought values of k in the set

$$\{1,\ C_0^{1/4},\ \sqrt{C_0},\ \frac{1}{2}C_0,\ C_0 - \sqrt{C_0},\ C_0 - C_0^{1/4},\ C_0\}$$

for each $n \in \{100, 1000, 10000\}$. We experimented with data instances corresponding to several slopes of lines (including 0° and 90°) and with various degrees of perturbation (e.g., $\sigma = 0, 10^{-6}, 10^{-2}, 10^{-1}$) and on uniformly random data. Examples of data instances for $n = 1000$ and various σ values with respect to the line $y = 0.5x + 0.25$ are shown in Figure 6. (The corresponding slope values given are those computed by the algorithm for the 50% quantile.) In cases of exact data with very small perturbations the algorithm would usually terminate after only one inversion sampling stage largely because of the fact that slopes "bunch up" significantly more in these cases and are caught by the bunch counting mentioned earlier in this section. In other cases the algorithm's performance was not significantly different from the standard cases.

6. Concluding Remarks

We have described a randomized $O(n \log n)$ expected time algorithm for selecting the k-th smallest line slope determined by all pairs of n points in the plane. Our emphasis has been on designing an algorithm which is provably efficient (with very high probability), which handles degenerate cases correctly, and which has a simple and efficient implementation. In much the spirit of Bentley's work on the traveling salesman problem,[3] we have experimented extensively with the implementation in order to establish its efficiency and robustness.

One problem suggested by this work is the extension of this technique to higher order or higher dimensional estimators. For example, one could fit a set of data points in the plane to a quadratic curve, $y = ax^2 + bx + c$, by generating all triples of data points and for each triple deriving the equation of the resulting curve. From these $O(n^3)$ curves we could then select the median of each of the three coefficients. This generates a three-dimensional instance of the Theil-Sen estimator. Similarly, a d-dimensional Theil-Sen estimator can be defined to fit a function of d parameters. An $O(n^{d-1}\log n)$ time and $O(n)$ space algorithm was recently proposed for the d-dimensional generalization of the Theil-Sen estimator.[21,22]

Another question regards the choice of the median slope as a line estimator. Although the median slope is preserved under translation and scaling, the median slope of a rotated set of points need not be equal to the rotated median slope. For example, when points are nearly vertical, we may find two clusters of slopes at $+\infty$ and $-\infty$. An estimator with the property that the estimator of a rotated point set is always equal to the rotated estimator is called a *rotation-equivariant* estimator. Stein and Werman have proposed a rotation-equivariant generalization of the Theil-Sen estimator and shown that it can be computed in $O(n\log^2 n)$ time.[29] Can this result be improved to $O(n\log n)$?

7. Acknowledgements

An algorithm for slope selection similar to the one presented in this paper was discovered, independently, by Matoušek.[19] We would like to thank Peter Cucka, Azriel Rosenfeld, Joseph Sanjour, and Chris Welsh for their suggestions and valuable assistance.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.
2. D. Angluin and L. G. Valiant, "Fast probabilistic algorithms for Hamiltonian circuits and matchings," *J. Comput. System Sci.* 19 (1979), 155–193.
3. J. L. Bentley, "Experiments on traveling salesman heuristics," in *Proc. First ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, California, 1990, 91–99.
4. R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi, "An optimal-time algorithm for slope selection," *SIAM J. Comput.* 18 (1989), 792–810.
5. G. Dahlquist and A. Björck, *Numerical Methods*, translated by N. Anderson, Prentice Hall, Englewood Cliffs, New Jersey, 1974.
6. D. L. Donoho and P. J. Huber, "The notion of breakdown point," in *A Festschrift for Erich L. Lehman*, eds. P. J. Bickel, K. Doksun, and J. L. Hodges, Jr., Wadsworth International Group, Belmont, California, 1983, 157–184.
7. R.O. Duda and P.E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Comm. ACM* 15 (1972), 11–15.
8. H. Edelsbrunner and E. P. Mücke, "Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms," *ACM Trans. Graphics* 9 (1990), 66–104.
9. H. Edelsbrunner and D. L. Souvaine, "Computing median-of-squares regression lines and guided topological sweep," *J. Amer. Stat. Assoc.* 85 (1990), 115–119.
10. P. Erdös and J. Spencer, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

11. W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. 1, John Wiley, New York, 1950.

12. M. A. Fischler and R. C. Bolles, "Random sampling consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Comm. ACM* 24 (1981), 381–395.

13. R. V. Hogg and E. A. Tannis, *Probability and Statistical Inference*, third edition, MacMillan, New York, 1983.

14. P. V. C. Hough, "Method and means for recognizing complex patterns," U. S. Patent 3,069,654, 1962.

15. H. Imai, K. Kato, and P. Yamamoto, "A linear-time algorithm for linear L_1 approximation of points," *Algorithmica* 4 (1989), 77–96.

16. B. Kamgar-Parsi, B. Kamgar-Parsi, and N. S. Netanyahu, "A nonparametric method for fitting a straight line to a noisy image," *IEEE Trans. Patt. Anal. Mach. Intell.* 11 (1989), 998–1001.

17. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, second edition, Addison-Wesley, Reading, Massachusetts, 1981.

18. D. E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, Massachusetts, 1973.

19. J. Matoušek, "Randomized optimal algorithm for slope selection," *Info. Proc. Letters* 39 (1991), 183–187.

20. N. Megiddo, "Linear time algorithms for linear programming in R^3 and related problems," *SIAM J. Comput.* 12 (1983), 759–776.

21. D. M. Mount and N. S. Netanyahu, "Computationally efficient algorithms for high-dimensional robust estimators," technical report in preparation, University of Maryland, College Park, 1992.

22. N. S. Netanyahu, "Computationally Efficient Algorithms for Robust Estimation," Ph.D. Thesis, University of Maryland, College Park, 1991.

23. P. J. Rousseeuw, "Least median-of-squares regression," *J. Amer. Stat. Assoc.* 79 (1984), 871–880.

24. P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, John Wiley and Sons, New York, 1987.

25. P. K. Sen, "Estimates of the regression coefficients based on Kendall's Tau," *J. Amer. Stat. Assoc.* 63 (1968), 1379–1389.

26. M. I. Shamos, "Computational Geometry," Ph.D. Thesis, Yale University, 1978.

27. M. I. Shamos, "Geometry and statistics: Problems at the interface," in *Algorithms and Complexity: New Directions and Recent Results*, ed. J. F. Traub, Academic Press, New York, 1976, 251–280.

28. D. L. Souvaine and J. M. Steele, "Efficient time and space algorithms for least median of squares regression," *J. Amer. Stat. Assoc.* 82 (1987), 794–801.

29. A. Stein and M. Werman, "Robust statistics in shape fitting," in *Proc. Eighth Israeli Symp. on Artificial Intelligence and Computer Vision*, Ramat-Gan, Israel, 1991, 285–302.

30. H. Theil, "A rank-invariant method of linear and polynomial regression analysis," *Proc. Kon. Ned. Akad. v. Wetensch. A* 53 (1950), 386–392.

31. J. V. Uspensky, *Introduction to Mathematical Probability*, McGraw Hill, New York, 1937.