CAR–TR–600
CS–TR–2816

CCR–89–08901
AFOSR–91–0239
December 1991

COMPUTATIONALLY EFFICIENT ALGORITHMS FOR A HIGHLY ROBUST LINE ESTIMATOR

David M. Mount[1]
Nathan S. Netanyahu[2]

[1]Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD. 20742–3255

[2]Computer Vision Laboratory
Center for Automation Research
University of Maryland
College Park, MD. 20742–3411

ABSTRACT

The problem of fitting a straight line to a set of data points is an important task in many application areas (e.g., statistical estimation, image processing, and pattern recognition). Recently the computation of linear estimators that are *robust* has been recognized as important, since these estimators are insensitive to outlying data points, which arise often in practice. In this paper we study one such robust estimator, the *repeated median* line estimator (Siegel 1982), which achieves the highest possible breakdown point of 50%. We present the following results: (1) a simple practical randomized algorithm that runs in $O(n \log^2 n)$ time with high probability, and (2) a slightly more complex randomized algorithm which performs as well asymptotically, but empirical evidence shows that this algorithm performs in time $O(n \log n)$ on many realistic input distributions. We present empirical evidence for the efficiency of this algorithm under a number of input distributions.

[1]The support of the National Science Foundation under Grant CCR–89–08901 is gratefully acknowledged.

[2]The support of the Air Force Office of Scientific Research under Grant AFOSR–91–0239 is gratefully acknowledged.

1. Introduction

Fitting a straight line to a number of data points is a common task frequently encountered in numerous fields of science and engineering (e.g., statistical estimation, image processing, and pattern recognition). Let (x_i, y_i), $i = 1, \ldots, n$, denote a set of n distinct points in the plane which are hypothesized to lie on a straight line. The line estimator $y = \hat{\theta}_1 x + \hat{\theta}_2$, where

$$\hat{\theta}_1 = \underset{i}{\text{med}} \underset{j \neq i}{\text{med}} \frac{y_j - y_i}{x_j - x_i}$$
$$\hat{\theta}_2 = \underset{i}{\text{med}} (y_i - \hat{\theta}_1 x_i),$$

was proposed by Siegel (1982) as the *repeated median* (RM) line estimator fitting the given points. Computing $\hat{\theta}_1$ amounts to computing the median of the set of elements, $\{\theta_i, i = 1, \ldots, n\}$, where each θ_i corresponds to the median of the $n-1$ line slopes defined by joining the point (x_i, y_i) with each of the other $n-1$ points (x_j, y_j), $j \neq i$. Once $\hat{\theta}_1$ is known, $\hat{\theta}_2$ is easily computed in $O(n)$ time.

Siegel's estimator is among a general class of what are termed *robust* estimators, that are of great importance in many practical applications, where there may exist points which lie significantly outside the ideal line of fit. One of the most important parameters used to evaluate the *robustness* of a given estimator is its sensitivity to outlying data, or *breakdown point*. This parameter is roughly defined as the smallest fraction of contamination in the data that may cause the estimator to have an arbitrary deviant values from its original estimate (see Donoho and Huber 1983, or Rousseeuw and Leroy 1987, for an exact definition). Siegel's (line) estimator achieves a 50% breakdown point, which is the ultimate

level of robustness considered by statisticians.

Common simple regression (i.e., line) estimators include the L_2 (least squares) estimator, the L_1 (least absolute values) estimator, and the L_∞ (Chebyshev) estimator. The L_2 estimator can be computed in $O(n)$ time (see Dahlquist and Björck 1974, pp. 85–96). The L_1 line estimator can also be computed in $O(n)$ time by extending Megiddo's $O(n)$ algorithms for linear programming (Megiddo 1983b, 1984), as was shown in Imai, Kato, and Yamamoto (1989). The L_∞ estimator was shown to be computable in $O(n \log n)$ time by finding the convex hull of the point set (Shamos 1978). This (time) complexity is readily reduced to $O(n)$ using Megiddo's technique. Although the above mentioned estimators can be computed in optimal time (i.e., $O(n)$), they are all very sensitive to outlying data. More formally, each of them has a 0% breakdown point since a single outlier can bias the estimator arbitrarily (see Rousseeuw and Leroy 1987, Chapter 1).

$O(n)$ time line estimators achieving improved robustness levels are, for example, the line estimators of Tukey (1970/1971) (16.6%), Brown and Mood (1951) (25%), and Andrews (1974) (25%). Other line estimators having higher breakdown points are the following:

- The Theil (1950) and Sen (1968) estimator, $y = \hat{\theta}_1 x + \hat{\theta}_2$, such that

$$\hat{\theta}_1 = \underset{1 \le i < j \le n}{\text{med}} \frac{y_j - y_i}{x_j - x_i},$$

$$\hat{\theta}_2 = \underset{1 \le i < j \le n}{\text{med}} \frac{y_i x_j - y_j x_i}{x_j - x_i}.$$

This estimator has a breakdown point of $1-\sqrt{1/2}\approx 29.3\%$, and its computation can be considered in a more general context, that of the *slope selection* problem. (The slope selection problem is that of determining the k-th smallest (e.g., median) slope among the $\binom{n}{2}$ line slopes defined by joining each pair of distinct points.) Cole, Salowe, Steiger, and Szemerédi (1989) discovered an optimal (deterministic) $O(n\log n)$ algorithm for this problem. Their algorithm relies on the following ingenious but sophisticated techniques: Cole's improvement (Cole 1987) of Megiddo's technique for parametric search (Megiddo 1983a) based on parallel sorting schemes (e.g., the $O(\log n)$ depth sorting network of Ajtai, Komlós, and Szemerédi 1983), and an algorithm for computing approximate ranks of elements in an array. Alternatively, in Dillencourt, Mount, and Netanyahu (1991) we presented an optimal *randomized* algorithm of expected time complexity $O(n\log n)$. (This result was arrived at, independently, by Matoušek 1991.) Our alternative approach resulted in a simple practical algorithm and the techniques used are also applicable for efficient computation of the Siegel estimator. (See Section 2 for a brief review.)

- The *least median of squares* (LMS) estimator (Rousseeuw 1984), i.e., the line that minimizes the median of the squared residuals, was shown to have a 50% breakdown point. Souvaine and Steele (1987) derived an $O(n^2)$ time and $O(n^2)$ space algorithm for its computation. The space bound was later improved to $O(n)$ by Edelsbrunner and Souvaine (1990).

- The *MM*-estimator (Yohai 1987) and the τ-estimator (Yohai and Zamar 1988) are both highly robust (50% breakdown point) and statistically sound (e.g., have high statistical efficiency). Their computation consists of iterative numerical procedures but their overall computational complexity is determined by their initial robust estimates (e.g., LMS, RM).

In general, the more robust and sophisticated an estimator is, the higher its computational complexity. Therefore, to maintain practical estimators that are highly robust and statistically sound one should focus on deriving efficient algorithms for robust estimators. Specifically, in this paper we present computationally efficient algorithms for the RM line estimator. Although the LMS estimator consists of somewhat superior statistical properties (e.g., affine equivariance), pursuing the RM estimator has resulted in achieving a twofold goal; (1) the computationally efficient algorithms derived establish the best result(s) known for a 50% breakdown line estimator and (2) overall highly robust, statistically sound, and computationally efficient estimators (e.g., *MM*-, τ-) are immediate byproducts.

Underlying techniques used by Cole *et al.* [7] for deriving their deterministic slope selection algorithm were also employed by Stein and Werman (1992) to obtain a deterministic $O(n \log^2 n)$ algorithm for the RM line estimator. As commented previously, these techniques while ingenious are quite complicated and not really practical. Independently we have discovered a set of algorithms for this problem, which we feel are a significant improvement. We present the fol-

lowing results.

- A simpler description of a deterministic $O(n\log^2 n)$ algorithm for the repeated median estimator is presented. Unfortunately, although easy to describe, the algorithm is rather impractical.
- A simple $O(n\log^2 n)$ (expected) time *randomized* algorithm is presented.
- An alternative $O(n\log^2 n)$ (expected) time *randomized* algorithm is presented. Although slightly more sophisticated, this algorithm has the practical advantage that it appears to run in $O(n\log n)$ time for many realistic input distributions. We describe the actual implementation of this algorithm, and demonstrate its practical efficiency empirically on a large number of input sets and distributions.

The main features of this latter randomized algorithm are:

(1) The algorithm is fairly easy to implement, relying only on simple modifications of mergesort.

(2) The constants of proportionality hidden by the asymptotic notation are small.

(3) The $O(n\log^2 n)$ expected running time occurs with high probability on any input of size n (probability of failure decreasing exponentially as n increases). Moreover, empirical evidence gained by experimenting with our algorithm over a wide range of data sets strongly suggests that, on the average, its expected running time is actually $O(n\log n)$ for many realistic input sets (see further discussion in Sections 4 and 5).

(4) The (randomized) algorithm always terminates giving the *correct* output in worst case $O(n^2)$, although the probability of achieving this worst case is extremely small for large n.

(5) The algorithm is space optimal, i.e., it requires $O(n)$ storage.

Although the paper relates primarily to the RM estimator, it should be noted that the algorithms derived can easily be generalized to select an element of arbitrary rank from each line, and then selecting an element of arbitrary rank from this set. Thus we can generally compute the following estimator.

$$\hat{\theta}_1 = \underset{i}{k_i\text{-}th} \; \underset{j \neq i}{k_j\text{-}th} \; \frac{y_j - y_i}{x_j - x_i},$$

where $1 \leq k_i \leq n$ and $1 \leq k_j \leq (n-1)$.

The paper is organized as follows: Our algorithmic methodology for the computation of the RM line estimator is introduced in Section 2. In Section 3 we present a deterministic $O(n \log^2 n)$ algorithm, and in Section 4 we give *practical* randomized versions which result in $O(n \log^2 n)$ expected running time algorithms. Implementation and practical experience with the above algorithms are discussed in Section 5. Section 6 contains a discussion and concluding remarks.

2. General Approach and Review of Basic Techniques

As was stated in the introductory section, the computation of $\hat{\theta}_1$ amounts to computing the median of the set of elements, $\{\theta_i, i = 1, \ldots, n\}$, where each θ_i corresponds to the median of the $n-1$ line slopes defined by joining the point (x_i, y_i) with each of the other $n-1$ points (x_j, y_j), $j \neq i$. For cases where n is

odd (i.e., $n-1$ is even), computing each θ_i according to the very definition of a median (i.e., the arithmetic average of the two middle entries of a sorted set of elements) would result in a more complicated algorithm. To alleviate this difficulty, we will make the convention that the median element of a set containing n elements is the m-th smallest element if $n = 2m-1$, and the m-th (or the $(m+1)$-st) smallest element if $n = 2m$. In the latter case, choosing between the m-th smallest entry (i.e., the *low median*) and the $(m+1)$-st entry (i.e., the *high median*) would be done consistently with respect to each θ_i. (Although the resulting estimator is slightly biased, it can easily be shown that it retains the 50% breakdown property of the original estimator.) Once $\hat{\theta}_1$ is known, computing $\hat{\theta}_2$ becomes a trivial $O(n)$ task. A brute force computation of $\hat{\theta}_1$ results, obviously, in an $O(n^2)$ time algorithm. To improve upon this time bound, we outline the following general approach.

First we dualize the problem by mapping each point $p_i(x_i, y_i)$ to a line $D(p_i)$ given by $y = x_i x - y_i$ and vice versa. It is well known that such a transformation has the following properties:

(1) *Incidence Preservation.* Point p lies on line l if and only if point $D(l)$ lies on line $D(p)$.

(2) *Order Preservation.* Point p lies above (below) line l if and only if point $D(l)$ lies below (above) line $D(p)$.

A straightforward corollary of (1) is that the line l joining two distinct points, p_1 and p_2, is the dual transformation of the intersection point of the lines $D(p_1)$ and

$D(p_2)$ and vice versa. In particular, the x coordinate of the intersection point, henceforth the *intersection ordinate*, is equal to the slope of l. (In case two lines are parallel, we make the convention that they meet at $+\infty$. To maintain consistency we also assume that vertical lines in primal space have slope $+\infty$.)

From the above discussion, it follows that the problem of finding the repeated median slope is now reduced to finding the median of the set of elements m_i, $i = 1, \ldots, n$, where each m_i is the median of $n - 1$ intersection ordinates of the line l_i with each of the other $n - 1$ lines (i.e., $\hat{\theta}_1 = \underset{i=1}{\overset{n}{\text{med}}}\, m_i$). Figure 1(a) illustrates a line arrangement in dual space where $n = 4$. The median x-coordinates of the intersections lying on l_1, l_2, l_3, and l_4 are x_2, x_5, x_4, and x_4, respectively. The median of these ordinates (i.e., the repeated median intersection ordinate) is x_4.

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, $x_{lo} < x \leq x_{hi}$, and let $I(x_{lo}, x_{hi}]$ denote the set of intersection ordinates (i.e., the x-coordinates of the intersection points between pairs of lines) in this interval. (Note that this is a multi-set since multiple equal ordinates are possible.) To complete our notation, let L denote the set of lines $\{l_i\}$ whose median intersection ordinate, m_i, lies to the left of $(x_{lo}, x_{hi}]$. Similarly, let R denote the set of lines $\{l_j\}$ whose median intersection ordinate, m_j, lies to the right of the interval, and let C represent the set of lines $\{l_k\}$ whose median intersection ordinate, m_k, lies within the interval. For example, Figure 1(b) depicts the line

arrangement of Figure 1(a), where $L = \{l_1\}$, $C = \{l_3, l_4\}$, $R = \{l_2\}$, with respect to the interval $(x_{lo}, x_{hi}]$. We will maintain the invariant that the repeated median intersection ordinate is in $I(x_{lo}, x_{hi}]$. Equivalently, that is to say that $|L| < n/2$, $|R| < n/2$, and $|L| + |C| + |R| = n$. Initially, $x_{lo} = -\infty$, $x_{hi} = +\infty$, $|L| = |R| = 0$, $C = \{l_1, l_2, \ldots, l_n\}$ (i.e., $|C| = n$), and $I(x_{lo}, x_{hi}]$ contains all $\binom{n}{2}$ intersection ordinates because, by convention, no two lines intersect at $-\infty$.

We begin by describing how our algorithms work at an intuitive level. The algorithms operate in a series of stages. At the start of each stage the repeated median intersection ordinate lies within an interval $(x_{lo}, x_{hi}]$. The idea is to repeatedly contract the interval into a smaller interval which also contains the desired ordinate. There are two methods we use to perform this contraction. The first method selects the median element, x_{med}, from the set of intersection ordinates in the interval (this is a median of all the ordinates, and is not to be confused with the repeated median). This can be done deterministically in $O(n \log n)$ time using a slope selection algorithm. A more practical approach is to use random sampling to pick a random subset of, say $O(n)$, intersection ordinates from the the interval. We show that this can be done in $O(n \log n)$ time deterministically by a simple variant of mergesort. Using this set of ordinates we use their median as an estimate for the true median of the interval. Using x_{med} as a test value, we can determine whether the repeated median entry is contained in the left subinterval $(x_{lo}, x_{med}]$ or in the right subinterval $(x_{med}, x_{hi}]$. This is

done by counting the intersection ordinates in various intervals by the process of inversion counting which is sketched below. Since the number of intersection ordinates in the initial interval is $O(n^2)$, it follows that after $O(\log(n^2)) = O(\log n)$ steps we will converge on the desired element. (For the randomized version this holds with high probability.) In fact, for greater efficiency, our implementation actually stops this binary search when the number of remaining intersection ordinates in the interval drops to $O(n)$, at which point we simply enumerate all of the remaining ordinates in $O(n \log n)$ time and use a standard selection algorithm [5], [15], to find the desired element.

The second method attempts to speed up the convergence rate of the binary search for many practical instances of the problem. The idea is analogous to the method used in our slope selection paper [11]. We contract a given interval from both above and below simultaneously with the hope of eliminating much more than just a constant fraction of intersection ordinates. We select two parameters P and Q such that $P \cdot Q = O(n)$, e.g., $P = Q = O(\sqrt{n})$. By randomly sampling Q lines, and P intersection ordinates along each line we can construct a set of confidence intervals for the medians of the sampled lines, and then combine the confidence intervals of the sampled lines to form a confidence interval for the repeated median. The resulting subinterval will contain the repeated median ordinate with high probability and for typical problem instances has a very small (decreasing with n) fraction of the remaining intersection ordinates. This algorithm terminates in $O(n \log^2 n)$ time with high probability, but empirical evidence suggests that the running time is closer to $O(n \log n)$ time for many typical

input distributions. See Sections 4, 5, for a detailed discussion.

The key issues of both algorithms involve counting the number of intersection ordinates in an interval efficiently, sampling intersection ordinates efficiently, and contracting the interval in an effective manner. In Subsection 2.2 we review how to count efficiently the number of intersection ordinates in a given interval. As it turns out, by using a simple modification of mergesort, this task can be achieved in $O(n \log n)$ worst-case time. Once the count, *Count*, is known, a sample of size (up to) $O(n)$ (with replacement) can be generated in $O(n \log n)$ time by generating a collection of (up to) $O(n)$ random integers in the range from 1 to *Count*, sorting this collection of integers, and running a slightly modified version of the counting algorithm. A variant of this sampling technique is introduced in Subsection 2.3. Randomized contraction of a given interval in an efficient manner, and a probabilistic analysis establishing the theoretical basis for it are introduced in Section 4.

To simplify the presentation of the algorithms derived, we will assume that the given points in primal space are in a general position, e.g., no three points lie on the same line. Equivalently, that is to say that no two intersection ordinates in dual space are equal to one another. Also, we will assume that no two lines in dual space intersect the vertical lines $x = x_{lo}$ or $x = x_{hi}$ at the same y-coordinate. In Section 5 we discuss ways of handling these degeneracies.

2.1. Counting Intersection Ordinates

Counting the number of intersection ordinates in an interval is easily reducible to the task of determining the number of inversions in a list. Define an *inversion* in a list $a_1, a_2, \ldots, a_n$, to be a pair of values, a_i and a_j, where $i < j$ but a_i is greater than a_j. Index lines in order according to the y-coordinate at which they intersect the left end of the interval, $x = x_{lo}$. Label each line with the y-coordinate at which it intersects the right end of the interval, $x = x_{hi}$. Consider the resulting list of labels in index order. It is easily observed that two lines intersect within the interval if and only if the relative order of their intersection with the left and right ends of the interval are reversed, which means that there is an inversion in the resulting list (see Figure 2). Thus, in order to count the number of intersection ordinates in the interval $(x_{lo}, x_{hi}]$, it suffices to count the number of inversions in the resulting list.

In our slope selection paper [11], we describe the function Ord_Count, which returns the number of intersection ordinates between x_{lo} and x_{hi}. This function invokes the procedure Inv_Count to count the inversions in a list. In the context of the RM line estimator, we invoke, respectively, the modified versions RM_Ord_Count and RM_Inv_Count to count not only the global number of intersection ordinates in $(x_{lo}, x_{hi}]$, but also the number of intersection ordinates that lie on each specific line in the interval. (Note that each intersection ordinate is accounted for twice.) This information is stored in *line_status*, an array of n records whose i-th entry corresponds to the i-th line, l_i. We store the number of l_i's intersection ordinates in $(x_{lo}, x_{hi}]$ in $l_i.count$. Other fields of a record are

used to store additional information regarding the corresponding line (see Sections 3, 4). The global variable n is the number of points and the global arrays X and Y contain the coefficients of the line equations $y = x_i x - y_i$. The integer parameter *Count* keeps a running (global) count of the number of inversions. In the algorithm below we say that when we sort lines we permute the associated array of line coefficients X and Y in parallel. This would be quite costly in practice since this involves a lot of data movement. We actually use a permutation vector to keep track of the location of lines, and access lines indirectly through this vector. We have omitted this extra level of indirection here to make the algorithms easier to read.

```
function RM_Ord_Count(x_lo, x_hi, line_status);
(* Count intersection ordinates in (x_lo, x_hi]) *)
begin
    for i := 1 to n do
        Y_c[i] := X[i] · x_lo - Y[i];
    Sort Y_c and permute X and Y in parallel;
    for i := 1 to n do
        Y_c[i] := X[i] · x_hi - Y[i];
    (* Initialize *)
    Count := 0;
    for i := 1 to n do
        l_i.count := 0;
    RM_Inv_Count(Y_c, 1, n, Count, line_status);
    return(Count)
end;
```

The problem of counting the number of inversions in a list is closely related to sorting, and is discussed in Knuth (1973). We describe a simple adaptation of mergesort that solves this problem in $O(n \log n)$ time while maintaining the number of intersection ordinates per each line. Assume that recursively, we wish

to sort the subarray $Y_c[l..u]$ of reals. Assuming that $u > l$, let $m = \lfloor (l + u)/2 \rfloor$. We divide the list into left and right subarrays $Y_c[l..m]$ and $Y_c[m+1..u]$ 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 i, $l \leq i \leq m$, the i-th element in the left sublist implicitly discovers the index j of the next larger element in the right sublist. It follows that each of the $j-1$ smaller elements in the right sublist induces an inversion with the i-th element in the left sublist, and thus $l_i.count$ is increased by $j-1$. Likewise, 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 both the (global) inversion counter and $l_j.count$ 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 $x_{lo} < x \leq x_{hi}$.) Although each intersection ordinate is accounted for twice (as far as updating the lines' *count* field is concerned), observe that the global counter, *Count*, is incremented only once (e.g., when an element of the right sublist is merged). As in mergesort, this algorithm terminates in $O(n \log n)$ time.

The procedure RM_Inv_Count which counts the number of inversions in the portion of the array $Y_c[l..u]$ of reals is described below. It calls the procedure RM_Merge which merges two sorted lists $Y_c[l..m]$ and $Y_c[m+1..u]$ and counts

inversions. A parallel auxiliary array $Aux[l..u]$ is used to hold the sorted output. Proper handling of ties in sorting is briefly discussed in Section 5. (The post-increment operator $i{+}{+}$ returns the current value of i and then increments the variable.)

```
function RM_Inv_Count(Y_c, l, u, Count, line_status);
(* Count the inversions associated with each line in Y[l..u] *)
begin
    if l = u then return;
    m := (l + u) div 2;
    RM_Inv_Count(Y_c, l, m, Count, line_status);
    RM_Inv_Count(Y_c, m + 1, u, Count, line_status);
    RM_Merge(Y_c, l, m, u, Count, line_status)
end;

procedure RM_Merge(Y_c, l, m, u, Count, line_status);
(* Merge Y_c[l..m] and Y_c[m + 1..u] *)
begin
    (* Initialize *)
    i := l;
    j := m + 1;
    k := l;
    while (i ≤ m and j ≤ u) do begin
        if (Y_c[i] ≤ Y_c[j]) then begin
        (* Copy from left and increment line's intersection count *)
            Aux[k++] := Y_c[i++];
            l_i.count := l_i.count + (j - 1)
        end
        else begin
        (* Copy from right and increment intersection counters *)
            Aux[k++] := Y_c[j++];
            l_j.count := l_j.count + (m - i + 1);
            Count := Count + (m - i + 1)
        end
    end;
    (* Copy the remaining elements from the left or from the right *)
    while (i ≤ m) do begin
        Aux[k++] := Y_c[i++];
        l_i.count := l_i.count + (j - 1)
    end;
    while (j ≤ u) do
```

$Aux[k{+}{+}] := Y_c[j{+}{+}];$
Copy $Aux[l..u]$ to $Y_c[l..u]$
end;

The RM_Merge procedure as presented here suffers from the deficiency of having to copy back the contents of the auxiliary array to Y_c after each recursive call. In the actual implementation, this is avoided by using two working arrays which are switched between at alternate levels of the recursion.

2.2. Sampling Intersection Ordinates

As was suggested in Subsection 2.1, in order to select the next candidates for x_{lo} and x_{hi}, we will need to uniformly sample a subset of $I(x_{lo}, x_{hi}]$. (Actual contraction of the interval is discussed in Section 4.) Suppose that we have already counted the number of intersection ordinates per each line in $I(x_{lo}, x_{hi}]$. Let $l_i.count$ $(i = 1, \ldots, n)$ be this count. We apply the following adaptation of the above procedure to sample (with replacement) a subset of $O(n)$ intersection ordinates. Assume, for example, that we are interested in sampling P intersection ordinates from each line, for some constant P. Thus, we first generate the n element array IS whose i-th entry consists of a list of P random integers distributed uniformly in the range from 1 to $l_i.count$ (allowing duplicates) and sorted in increasing order. (This array of lists is implemented using a two dimensional array and using a array of n indices to indicate the current element of each list.)

Next, we rerun the mergesort algorithm of the preceding subsection but instead of counting intersections, for each line l_i, $1 \leq i \leq n$, and for each element $k \in IS[i]$, we select the k-th intersection ordinate counted by the counting

procedure to be part of the sample. More precisely, to perform this sampling we modify RM_Inv_Count as follows. Each time the line counter of an element from the left sublist is incremented in RM_Merge, $Y_c[i]$ induces an inversion with each of $Y_c[j']$, $m < j' < j$. This corresponds to increasing $l_i.count$ by $\Delta count_i = j - 1$. We check the list of random integers, $IS[i]$, for elements in the range $l_i.count + 1$ to $l_i.count + \Delta count_i$, and select for the sample the x-coordinate at which the corresponding pairs of lines intersect. Likewise, each time the line counter of an element from the right sublist is incremented, $Y_c[j]$ induces an inversion with each of $Y_c[i']$, $i \leq i' \leq m$. Selecting the sample in this case is done analogously to the description above (see code for RM_Merge2, below). We maintain an array $Samp[i]$, $1 \leq i \leq n$, which holds a list of the sampled intersection ordinates for line i.

We claim that the sample selection can be performed in $O(n \log n)$ time plus constant time per element selected. The reason is that for each step of the merging process, even though an element may induce a large number of inversions we need access only those elements of the sorted subarray which are to be sampled. Thus the total running time of this modified mergesort procedure is still $O(n \log n)$. The ordinate sampling procedure RM_Samp_Ord is essentially the same as the function RM_Ord_Count given earlier, but the statements incrementing $l_i.count$ and $l_j.count$ in RM_Merge are replaced with the code below. The list operator *Curr* returns the value of the current element of a list (initially the first element), *Extract_Curr* returns the current element and advances to the next element of this list, and *Add* adds an element to a list.

```
function RM_Merge2(Y_c, l, m, u, Count, line_status, IS_Indx, Samp);
begin
(* Same as RM_Merge but replace the statement(s) incrementing l_i.count with *)
    ...
    aux_count := l_i.count + (j - 1);
    while (Curr(IS[i]) ≤ aux_count) do begin
        j' := Extract_Curr(IS[i]) - l_i.count;
        (* Add new x-coordinate to Samp[i] *)
        Add(Samp[i],(Y[j'] - Y[i])/(X[j'] - X[i]));
    end;
    l_i.count := aux_count;
    ...
(* Same as RM_Merge but replace the statement(s) incrementing l_j.count with *)
    ...
    aux_count := l_j.count + (m - i + 1);
    while (Curr(IS[j]) ≤ aux_count) do begin
        i' := Extract_Curr(IS[j]) - l_j.count;
        (* Add new x-coordinate to Samp[j] *)
        Add(Samp[j],(Y[j] - Y[i'])/(X[j] - X[i']));
    end;
    l_j.count := aux_count;
    ...
end;
```

3. A Deterministic Algorithm

As mentioned earlier, the algorithm consists of a series of stages. During each stage we are given the sets of lines L and C, and a corresponding interval $(x_{lo}, x_{hi}]$ within which we seek the repeated median intersection ordinate. More specifically, we assume that for each line $l_i \in C$ we know the number of its intersections to the left of the interval, $l_i.L$, and the number of its intersections within the interval, $l_i.C$. This information is stored in the data structure *line_status*. (Obviously, the number of l_i's intersections to the right of the interval is $(n-1) - l_i.L - l_i.C$.)

Once $|C| = 1$ (i.e., there is one remaining line in the interval currently considered which contains the desired intersection ordinate), we apply a simple enumeration algorithm to locate the repeated median value. The enumeration procedure can simply be done by generating the list of $n-1$ intersection ordinates of the remaining line with each of the other $n-1$ lines. After this, an $O(n)$ selection algorithm (e.g., [5], [15]) can be invoked to obtain the $(\text{med}(n) - l_i.L)$-th smallest ordinate, i.e., the repeated median value. (By $\text{med}(n)$ we mean $n/2$ if n is even, and $(n+1)/2$ if n is odd.)

For efficiency it is actually faster to terminate the algorithm when "sufficiently few" intersections remain to be considered. For example, if $|C| \leq c$, or if $Tot\_Count \leq c_1 n$, where

$$Tot\_Count = \sum_{i=1}^{|C|} l_i.C,$$

for some constants c, $c_1 \geq 1$, we terminate the interval contraction and apply a different enumeration algorithm to locate the desired intersection ordinate. The algorithm is a simple modification of RM_Inv_Count (see Subsection 2.2). Beyond counting inversions associated with the remaining lines in C, the enumeration generates a list of inversion ordinates. This can be obtained by modifying the sampling procedure to sample *every* ordinate in the interval which belongs to the remaining lines. To be more precise, each element of the list is of the form $<i, x_{int}>$, where x_{int} is a (sampled) intersection ordinate of l_i. Lexicographic sorting of the list allows for $O(1)$ retrieval, for each of the $|C|$ sublists of the $(\text{med}(n-1) - l_i.L)$-th smallest ordinate of the line l_i. (Since the size of

the list is less than or equal to $|C|(n-1)$, i.e., $O(n)$, this operation requires $O(n \log n)$ time.) The repeated median value is obtained by invoking a fast selection algorithm to select the $(\text{med}(n) - |L|)$-th smallest element of these $|C|$ median ordinates.

If the termination condition does not hold, the algorithm proceeds by first picking an intersection ordinate in the current interval, e.g., the *median* intersection ordinate. This can be accomplished by invoking a slope selection algorithm to select, for example, the $(Left + Count/2)$-th smallest intersection ordinate (among the $\binom{n}{2}$ ordinates in $I(-\infty, +\infty]$), where $Left$ and $Count$ denote the number of intersections in $(-\infty, x_{lo}]$ and $(x_{lo}, x_{hi}]$, respectively. In principle, we may invoke the deterministic $O(n \log n)$ algorithm of Cole *et al.* [7], as a "black box", to obtain this ordinate. (In the next section we will discuss how to improve upon this purely theoretical approach.) Let x_{med} denote the value obtained. x_{med} serves as a test value with respect to the desired repeated median estimate. At this point, the algorithm performs inversion counting to determine whether $\hat{\theta}_1 \in (x_{lo}, x_{med}]$ or $\hat{\theta}_1 \in (x_{med}, x_{hi}]$. This can be done by invoking a modified inversion counting procedure with respect to the interval $(x_{lo}, x_{med}]$, for example. By (tentatively) updating C, a decision is reached as to what interval should be considered next; $(x_{lo}, x_{med}]$ or $(x_{med}, x_{hi}]$. The variables L, C, x_{lo}, x_{hi}, and *line_status* are updated appropriately, and the algorithm continues to loop until a termination condition is met.

The overall algorithm is described below. Initially, $x_{lo} = -\infty$, $x_{hi} = +\infty$, L is empty, $C = \{l_1, \ldots, l_n\}$, and the status of all lines is such that $l_i.L = 0$ and $l_i.C = n-1$, where $i = 1, \ldots, n$. C' is an auxiliary variable which serves for updating C.

```
function Repeated_Median(x_lo, x_hi, L, C, line_status);
begin
    while (| C | > 1) do begin
        C' := ∅;
        Left := Ord_Count(-∞, x_lo);
        Count := RM_Ord_Count(x_lo, x_hi, line_status);
        (* Invoke Cole et al.'s slope selection algorithm *)
        x_med = selected (Left + Count/2)-th smallest slope;
        for all l_i ∈ C do
            if l_i.L + l_i.count > med(n-1) then
            (* l_i's median ordinate ∈ (x_lo, x_med] *)
                C' := C' ∪ {l_i};
        if | L | + | C' | ≥ med(n) then begin
            (* Repeated median ∈ (x_lo, x_med] *)
            x_hi := x_med;
            C := C';
            (* Update line_status *);
            for all l_i ∈ C do
                l_i.C := l_i.count
        end
        else begin
            (* Repeated median ∈ (x_med, x_hi] *)
            x_lo := x_med;
            L := L ∪ C';
            C := C \ C';
            (* Update line_status *)
            for all l_i ∈ C do begin
                l_i.L := l_i.L + l_i.count;
                l_i.C := l_i.C - l_i.count
            end
        end
    end;
    (* | C | = 1 *)
    Enumerate all of the intersection ordinates of the remaining line l in C;
    Select the (med(n-1) - l.L)-th smallest entry;
```

Return this ordinate
end;

Lemma 3.1: The RM line estimator can be computed by a deterministic $O(n\log^2 n)$ time and $O(n)$ space algorithm.

Proof: Each while-loop consists of various procedures (e.g., modified inversion counting, slope selection, etc.) which are known to require $O(n\log n)$ time and $O(n)$ space. By definition of the interval contraction, (approximately) half of the intersection ordinates in the current interval are eliminated through the execution of one iteration. As the initial number of intersection ordinates is $O(n^2)$, it is obvious that $O(\log(n^2)) = O(\log n)$ iterations suffice to meet the termination condition. Hence, Lemma 3.1 follows. ■

4. A Randomized Approach

4.1. Randomized Binary Search

As indicated earlier, one may invoke the slope selection procedure of Cole *et al.* [7] as a "black box" for obtaining x_{med}, thus yielding a deterministic algorithm. Incorporating the techniques introduced by Cole *et al.* results, however, in a purely theoretical algorithm. In practice, to obtain the test value, x_{med}, we could invoke various other procedures.

One possibility is to replace the above "black box" with our randomized slope selection module [11]. Instead of selecting the *exact* median intersection ordinate in $I(x_{lo}, x_{hi}]$ as a test value, we can further simplify the above procedure by selecting at random an intersection ordinate from $(x_{lo}, x_{hi}]$. Although

the resulting algorithm may eliminate occasionally only a small fraction of the remaining intersection ordinates in the interval, it would still exhibit an *expected* convergence of $O(\log n)$ stages. (This is established by similar analysis to that provided for the randomized version of Quicksort [14], [8].) However, to get a single random intersection ordinate requires executing an $O(n \log n)$ sampling procedure, and in this much time we can get a better estimate, i.e., a test value that, with high probability, is "sufficiently close" to x_{med}.

The idea is to pick the median of the random sample of n elements in the interval. Let $x_{med}{}^*$ denote this alternative test value. As noted in the derivation of our slope selection algorithm, $x_{med}{}^*$ is expected to be "sufficiently close" to the actual median. More importantly, with high probability, this variant of the Repeated_Median procedure would still eliminate approximately half of the intersection ordinates in $I(x_{lo}, x_{hi}]$. This claim can be intuitively justified in view of the probabilistic analysis provided in Chapter 2. More specifically, observe that each randomly selected element is less than the median intersection ordinate with probability equal to 1/2. Thus the expected number of successes is given by the mean of a binomial distribution. Since the number of trials grows with n, it follows from Chernoff's bounds that sample median will be very close in rank to the true median for large n. Summarizing, we have the following:

Lemma 4.1: The RM line estimator can be computed, with high probability, in (expected) $O(n \log^2 n)$ time and $O(n)$ space.

4.2. Efficient Randomized Interval Contraction

In this subsection we introduce a refinement to further reduce the time complexity of the RM line estimator. We will focus mainly on deriving "more efficient" procedures for interval contraction. Put differently, we are interested in interval contraction that, with high probability, traps the repeated median ordinate and at the same time eliminates as many intersection ordinates as possible.

Consider an interval $(x_{lo}, x_{hi}]$ (in dual space) which is hypothesized to contain $\hat{\theta}_1$. Also, we assume the same notation presented in Section 2 (i.e., L, C, and R denote the sets of lines whose median intersection ordinates lie, respectively, to the left, inside, and to the right of the interval). We would like to contract this interval, as efficiently as possible, to a smaller subinterval $(x_{lo}', x_{hi}']$ which also contains $\hat{\theta}_1$. In an attempt to reduce the current number of $O(\log n)$ contraction stages, we pick two parameters P, Q, such that $P \cdot Q = O(n)$, e.g., $P = Q = O(\sqrt{n})$. (The choice of P and Q will affect the probabilistic performance of the algorithm, but not its correctness. In Section 5 we discuss issues in choosing these parameters, but for now we leave the discussion as general as possible.) Q lines (that belong to the set C) are selected at random and on each of these lines, P intersection ordinates in the interval $(x_{lo}, x_{hi}]$ are randomly sampled. (We allow for sampling with replacement.) Note that the sampler described is clearly a variant of our inversion counting/sampling routines mentioned in Section 2. In other words, it requires $O(n \log n)$ time and $O(n)$ space. For each of the Q sampled lines, an estimate is obtained for its median intersection point from the sample of P ordinates. (This is accomplished by invoking any selection

algorithm with respect to the line's P sampled intersection ordinates.)

Similar to the slope selection case, we can construct a confidence interval for each sampled line that will contain, with high probability, the line's (true) median intersection ordinate, given a sufficiently large sample. Let $(x_{lo_i}, x_{hi_i}]$ denote the confidence interval corresponding to the i-th sampled line $(i = 1, \ldots, Q)$. Now we repeat the process individually on the sets $\{x_{lo_i}\}$ and $\{x_{hi_i}\}$, determining for each a confidence interval for the median of these lower bounds and median of these upper bounds, respectively. In particular, let x_{lo}' be the lower bound on the confidence interval for the median of the set $\{x_{lo_i}\}$ and let x_{hi}' be the upper bound of the confidence interval for the median of the set $\{x_{hi_i}\}$. (To be precise, since the repeated median is not necessarily the median of the set C, we actually are looking for an confidence interval for the $(\text{med}(n) - |L|)$-th smallest element of C.) Since we have confidence that the median of each sampled line is trapped within the individual confidence intervals for each line, it is quite reasonable to expect that the true repeated median should be trapped within an interval formed by a lower bound for the set of lower bounds, and an upper bound for the set of upper bounds, that is $(x_{lo}', x_{hi}']$. In Subsection 4.3 we will show that (subject to the proper selection of P and Q) this subinterval contains, with high probability, the repeated median ordinate.

By balancing P and Q, e.g. $P = Q = O(\sqrt{n})$, the quality of the confidence intervals both within each line and among the different lines is quite good. However, it is also interesting to consider how the algorithm behaves in

the two extreme cases (1) $P = O(n)$, $Q = O(1)$, and (2) $P = O(1)$, $Q = O(n)$. In the first case, we randomly select a constant number of lines, perhaps only one, and randomly generate $O(n)$ (essentially all) intersection ordinates and generate a confidence interval for the median of these few lines. In the second case, we randomly select $O(n)$ lines (essentially all) and randomly select a constant number of intersection ordinates for each line, from which a confidence interval is constructed. In the first case we get a very good confidence interval from one line, and in the second case we get a number of fairly poor confidence intervals from every line. Between the two cases, the latter is intuitively better from a probabilistic standpoint since it allows the aggregation of many poor estimates to form a better estimate; whereas the former is doomed to poor performance if the chosen line is not a good representative. Expanding on this observation, note that it is not obviously necessary to even compute confidence intervals for the median of each line. For example, if even a single random intersection ordinate is sampled per line, it can still be argued that the median of these values is a reasonable estimate for the repeated median. (These observations will be justified further in Section 4.3.) In the algorithm described below we have two other parameters t_P and t_Q which are used in determining the size of the confidence intervals taking within the lines and between the lines, respectively. By setting $t_P = 0$ we can effectively shrink the confidence interval on each line to a single point, simulating this special case.

We can now describe the algorithm for sampling and constructing confidence intervals. The variable *Samp* denotes a $Q \times P$ array whose i-th row contains P

sampled intersection ordinates that lie on the i-th sampled line. (It is assumed that $Samp[i][1], \ldots, Samp[i][P]$, the elements of this sample, have been sorted in increasing order.) All of the $Q \times P$ ordinates sampled lie in the interval $(x_{lo}, x_{hi}]$, and a contraction procedure is invoked once *Samp* is known. Analogously to the slope selection case, confidence intervals are constructed by picking ordinates of various samples which lie several standard deviations (index-wise) on either side of the sample's mean (i.e., expected) index. The (global) parameters t_P and t_Q specify the numbers of such standard deviation shifts applied to sample sizes of P and Q, respectively. As mentioned earlier, by setting $t_P = 0$ we allow for the special case in which the confidence intervals for each line shrink to a single point. A theoretical justification of the probabilistic properties of this algorithm is presented in Subsection 4.3.

procedure Contract(*Samp*, $|L|$, $|C|$, *line_status*, x_{lo}', x_{hi}');
begin
(* Compute a standard deviation with respect to each line's sample *)
$\sigma_P := \sqrt{P}/2$;
(* Compute a standard deviation with respect to the line sample *)
$\sigma_Q := \sqrt{Q}/2$;
(* Construct confidence interval for each sampled line *)
for $i := 1$ **to** Q **do begin**
$k_i := \text{med}(n-1) - l_i.L$;
$k_{lo_i} := \max(1, \lfloor k_i - t_P\sigma_P \rfloor)$;
$x_{lo_i} := Samp[i][k_{lo_i}]$;
$k_{hi_i} := \min(P, \lceil k_i + t_P\sigma_P \rceil)$;
$x_{hi_i} := Samp[i][k_{hi_i}]$
end;
(* Construct confidence interval for the repeated median *)
$k := (\text{med}(n) - |L|) \cdot Q / |C|$;
(* Shift t_Q standard deviations *)
$k_{lo} := \max(1, \lfloor k - t_Q \cdot \sigma_Q \rfloor$;
$k_{hi} := \min(Q, \lceil k + t_Q \cdot \sigma_Q \rceil$;

$x_{lo}' := k_{lo}$–th smallest$\{x_{lo_i},\ i = 1, \ldots, Q\}$;
$x_{hi}' := k_{hi}$–th smallest$\{x_{hi_i},\ i = 1, \ldots, Q\}$

end;

The above contraction scheme is illustrated in Figure 3. Figure 3(a) illustrates the construction of a confidence interval $(x_{lo_i}, x_{hi_i}]$ for a sampled line l_i, and Figure 3(b) illustrates the construction of a confidence interval based on the values $\{x_{lo_i}\}, \{x_{hi_i}\}$ $(i = 1, \ldots, Q)$.

Once a confidence interval is constructed, the algorithm proceeds in a similar fashion described in our slope selection paper [11]. Specifically, a verification procedure is employed to test whether the repeated median is contained in $(x_{lo}', x_{hi}']$. If not, a different subinterval should be considered next, i.e., $(x_{lo}, x_{lo}']$ or $(x_{hi}', x_{hi}]$, and other variables such as L, C, and *line_status* are updated accordingly. The algorithm continues to loop until the termination condition is met. We prefer using the weaker termination condition $Tot\_Count \leq cn$ (for some constant $c \geq 1$) to $|C| \leq c$, for example, since it is more likely to result in less iterative steps without affecting the complexity bounds of the algorithm.

We call this algorithm the P-Q algorithm for the repeated median. This description allows for four separate parameters, whose choice affects the performance of the algorithm (although not its correctness). The first two parameters are Q and P, which determine the number of lines sampled, and the number of intersection ordinates sampled per line, respectively. The second two parameters specify the size of the confidence intervals taken for each sample about the estimated median. As in the Theil-Sen estimator, these parameters are given as

t_P and t_Q, where the radius of the confidence intervals are $t_P\sqrt{P}/2$ and $t_Q\sqrt{Q}/2$, respectively. For our implementation we have chosen $P = O(\sqrt{n})$, $Q = O(\sqrt{n})$, $t_P = 0$ and $t_Q = O(1)$. The choice $t_P = 0$ means that for each line we only return a single estimator of the lines' median, rather than a confidence interval. We have found that it suffices to use only the confidence interval specified by t_Q to trap the final repeated median with high probability. Our analysis of the algorithm's performance assumes this particular choice of parameters (and the fact that $t_P = 0$ is critical to our analysis).

The overall algorithm is described below. Initially, $x_{lo} = -\infty$, $x_{hi} = +\infty$, L is empty, $C = \{l_1, \ldots, l_n\}$, $Tot\_Count = n(n-1)$, and the status of all lines is such that $l_i.L = 0$ and $l_i.C = n-1$, where $i = 1, \ldots, n$. C' and C'' are auxiliary variables (initialized to the empty set) which serve for updating C.

function Randomized_Repeated_Median(x_{lo}, x_{hi}, L, C, *line_status*, *Tot_Count*);
begin
  while (*Tot_Count* $>$ *cn*) **do begin**
    $C' := \emptyset$;
    $C'' := \emptyset$;
    Generate a sorted list of Q random numbers from 1 to $|\,C\,|$,
      stored in $LI[1], \ldots, LI[Q]$;
    (* $LI[i]$ is the index of the i-th sampled line in C *)
    for all $l_{LI[i]}$ **do**
      Generate a sorted list of P random numbers from 1 to $l_{LI[i]}.C$,
      stored in $IS[i][1], \ldots, IS[i][P]$;
    (* IS is a $Q \times P$ array of sampled indices *)
    (* Sample $Q \times P$ ordinates from $I(x_{lo}, x_{hi}]$ and store them in *Samp* *)
    RM_Samp_Ord(x_{lo}, x_{hi}, *line_status*, IS, *Samp*);
    (* Contract the interval *)
    Contract(S, $|\,L\,|$, $|\,C\,|$, *line_status*, x_{lo}', x_{hi}');
    (* Verify whether or not the repeated median $\in (x_{lo}', x_{hi}']$ *)
    RM_Ord_Count(x_{lo}, x_{lo}', *line_status*);
    For all $l_i \in C$ copy $l_i.count$ to $line\_left\_count[i]$;

RM_Ord_Count(x_{lo}', x_{hi}', *line_status*);

for all $l_i \in C$ **do begin**

if $l_i.L + left\_line\_count[i] \geq \text{med}(n-1)$ **then**

(* l_i's median intersection $\in (x_{lo}, x_{lo}']$ *)

$C' := C' \cup \{l_i\}$

else if $l_i.L + left\_line\_count[i] + l_i.count < \text{med}(n-1)$ **then**

(* l_i's median intersection $\in (x_{hi}', x_{hi}]$ *)

$C'' := C'' \cup \{l_i\}$

end;

if $|L| + |C'| \geq \text{med}(n)$ **then begin**

(* Repeated median lies in the left subinterval $(x_{lo}, x_{lo}']$ *)

$x_{hi} := x_{lo}'$;

$C := C'$;

(* Update *line_status* *)

for all $l_i \in C$ **do**

$l_i.C := line\_left\_count[i]$

end

else if $|L| + |C \setminus C''| \geq \text{med}(n)$ **then begin**

(* Repeated median lies in the center subinterval $(x_{lo}', x_{hi}']$ *)

$L := L \cup C'$;

(* Update *line_status* *)

for all $l_i \in C$ **do begin**

$l_i.C := l_i.count$;

$l_i.L := l_i.L + line\_left\_count[i]$

end

end

else begin

(* Repeated median lies in the right subinterval $(x_{hi}', x_{hi}]$ *)

$L := L \cup (C \setminus C'')$;

$C := C''$;

(* Update *line_status* *)

for all $l_i \in C$ **do begin**

$l_i.C := l_i.C - (l_i.count + line\_left\_count[i])$;

$l_i.L := l_i.L + line\_left\_count[i] + l_i.count$

end

end

Update *Tot_Count*

end;

(* $Tot\_Count \leq cn$ *)

Enumerate all of the intersection ordinates of the remaining lines in C;

Sort lexicographically <line, intersection ordinate>;

for each remaining $l_i \in C$ **do**

Select its $(\text{med}(n-1) - l_i.L)$-th smallest intersection ordinate;

Select the $(\text{med}(n) - L)$-th smallest entry of these $|C|$ ordinates;

Return this ordinate
end;

4.3. Probabilistic Analysis

In this section we establish the probabilistic theory of the randomized P-Q algorithm that was just presented. We analyze two cases. First we show that the algorithm's expected running time is $O(n \log^2 n)$ with no assumptions on the structure of the input. Second we show that if the intersection ordinates along each line are assumed to be independent and identically distributed (i.i.d.), then the running time is $O(n \log n)$, and in particular it terminates after a constant number of iterations of $O(n \log n)$ time each. This latter result forms the theoretical basis for the claim that the algorithm's performance is better for realistic input sets (which is demonstrated empirically in the next section). Although it is not generally reasonable to assume that the intersection points on each line are i.i.d., observe that for many realistic inputs for robust estimators, the data typically consists of a significant constant fraction of points that lie near to the line of fit, and a constant fraction of outlying data points. We suspect that the algorithm performs so well because after a constant number of iterations, the intersection ordinates generated by the outlying data points are eliminated from consideration by the interval contraction algorithm. After this the remaining intersection ordinates are sufficiently close to i.i.d. that the algorithm terminates in an additional constant number of iterations.

Lemma 4.2: The P-Q randomized algorithm for computing the RM estimator

runs in $O(n \log^2 n)$ time with high probability.

Proof: We make use of the following results which were proven in [11], and which follow from a basic analysis of the binomial distribution. Given a universe of M distinct numbers, and an integer k, $1 \leq k \leq M$, and given a random subset of m elements from this universe, we can select two elements from the sample (together with the minimum and maximum elements of the universe) which form a confidence interval for the k-th smallest element of the universe, such that (1) the probability that this confidence interval contains the k-th smallest can be made arbitrarily high, (2) the number of elements from the sample contained in the confidence interval is $O(\sqrt{m})$, and (3) the number of elements of the universe contained within this confidence interval is $O(M/\sqrt{m})$ with arbitrarily high probability. The constants hidden by the asymptotic bounds are functions of the probability of (1) and (2). This confidence interval referred to here is exactly the one computed by the algorithm.

Let us assume that, among the C lines whose line median is known to lie within the interval $(x_{lo}, x_{hi}]$, at least half have their line median less than the repeated median. (The other case is symmetric.) Thus for a given sampled line the probability that its line median is not greater than the repeated median is at least $1/2$. Independent of this, because the median estimator for each sampled line is chosen symmetrically about the line's true median, with probability at least $1/2$ the estimated median for each sampled line is not greater than the line's median. Combining these observations it follows that with probability at least

1/4 the estimated median for each sampled line is not greater than the repeated median. If we consider a sample to be a success if this occurs, then among the Q lines sampled, we expect at least $Q/4 - O(\sqrt{m})$ sampled lines to be successes with high probability. Furthermore, since there are only $O(\sqrt{m})$ sampled lines in the contracted confidence interval, it follows that almost all of these successes are less than x_{lo}', the lower bound for the confidence interval. With probability at least 1/2 the repeated median is greater than x_{lo}' (since this value is greater than an unbiased estimate for the repeated median). Whenever this is the case, we eliminate these roughly $Q/4$ lines from further consideration. Now applying the result stated in the previous paragraph with $M = C$ and $m = P = O(\sqrt{n})$, it follows that with high probability, at least $C/4 - O(C/\sqrt{n})$ (i.e. roughly $C/4$ for large n) lines are eliminated from further consideration.

Thus for roughly half of the iterations we eliminate a constant fraction of the lines. Since the initial number of lines is n, the expected number of iterations will be $O(\log n)$, as desired. The probability that we exceed this expected number of iterations by a given constant factor is at most $(1/2)^{O(\log n)}$ which is function of the form $1/(n^e)$ for some constant $e > 0$. Thus this result holds with high probability for large n. ■

Since each iteration takes $O(n \log n)$ time, and $O(n)$ space, we have the following.

Theorem 4.3: The P-Q randomized algorithm for computing the RM estimator

runs in time $O(n \log^2 n)$ with high probability and $O(n)$ space.

We now show that the algorithm terminates in a constant number of iterations if the intersection ordinates for each line are assumed to be i.i.d.

Lemma 4.4: The P-Q randomized algorithm for computing the RM estimator terminates after a constant number of iterations with high probability.

Proof: It suffices to prove two facts. First, that with high probability each iteration of this algorithm succeeds in trapping the repeated median within the newly contracted interval $(x_{lo}', x_{hi}']$, and second, that each time we do this we eliminate all but a fraction of $O(1/(n^{1/4}))$ of the remaining intersection ordinates from further consideration. Let us call an iteration successful if both of these conditions hold. By establishing these two facts, it follows that after k successful iterations the number of remaining intersection ordinates will be reduced by a fraction of $O(1/(n^{k/4}))$. Since the initial number of intersection ordinates is $O(n^2)$, after roughly 5 iterations the number of remaining intersection ordinates will be reduced to a quantity that is sublinear in n, implying that the enumeration phase can complete the process.

Let μ denote the median of the distribution. To simplify the presentation we will assume that the probability density of the distribution, restricted to the interval $(x_{lo}', x_{hi}']$ is evenly distributed about μ. (The general case is similar, replacing 1/2 by the corresponding probability.) To establish the first fact we observe that for each intersection ordinate sampled on each line the probability that it is less than μ is 1/2. Thus the median of a given sample of ordinates on

each line is an unbiased estimator for μ, and hence it is less than μ with probability 1/2. The proof of the remarks in Lemma 4.2 is based on this same fact. It follows directly from these remarks that we expect μ to lie within this contracted confidence interval with high probability.

To establish the second fact, let M_i denote the number of intersection ordinates remaining for the line l_i within the interval $(x_{lo}, x_{hi}]$. Because the distribution of a line's intersection ordinates is i.i.d. this expected value will be consistent over the entire universe of lines. Using the notions introduced in the remarks made in Lemma 4.2, imagine that for each line l_i we construct a confidence interval around the estimated line median (although our algorithm does not actually construct such an interval). From these remarks applied to this line's intersection ordinates, it follows that with high probability (1) μ lie within this confidence interval, and (2) there are $O(M_i/\sqrt{P})$ intersection ordinates of l_i lying within this confidence interval. Let us call a sampled line a success if these two conditions are met. Observe that successful sampled lines will tend to have estimates close to the value μ, and hence we expect their estimates to be closest to the median of the sample of estimates.

Because the estimated line median lies within each confidence interval (indeed the confidence interval is built symmetrically outwards around the estimate), it follows that for each line the number of intersection ordinates lying between μ and the line's estimate is not greater than $O(M_i/\sqrt{P})$, with high probability. Since this holds with high probability for all lines, and since the endpoints of the contracted interval are drawn from two "central" estimates (lying

within $O(\sqrt{Q})$ of the median of the sample of Q estimates), it follows that the number of intersection ordinates on each line between the lower and upper endpoints of the contracted interval will be no greater than $O(M_i/\sqrt{P})$, with high probability. Summing over all successful lines shows that the number of intersection ordinates remaining in the interval drops by a fraction of $O(1/\sqrt{P}) = O(1/(n^{1/4}))$ on the average. ■

Since each iteration requires $O(n \log n)$ time, and $O(n)$ space, we arrive at the following:

Theorem 4.5: The P-Q randomized algorithm for computing the RM estimator runs in time $O(n \log n)$, with high probability, and $O(n)$ space if the intersection ordinates on each line are independent and identically distributed.

5. Implementation and Experimental Results

In this section we briefly mention several implementation issues that were left unspecified in the preceding presentation. Such issues include the following:

(1) How to handle various degenerate and special cases.

(2) How to deal with limited numerical precision.

(3) How to choose the constant parameters referred to in earlier sections which affect the probabilistic behavior and running time of the algorithm(s).

In addition to the above we also describe the experimental behavior of an implementation of our algorithm.

5.1. Degeneracies

Adequate handling of special cases and degeneracies is an important task concerning any (computational) geometry algorithm. Since the algorithm deals exclusively with lines, we could pass all the handling of degeneracies to a general purpose system, e.g., the *simulation of simplicity* technique introduced by Edelsbrunner and Mücke (1990). However, for the sake of efficiency and due to the limited nature of the degeneracies, we choose to handle degeneracies on a case by case basis.

Degenerate and special cases arise in the following cases: (1) When values to be sorted or ranked (i.e., y-coordinates at x_{lo} and x_{hi}) are equal to one another. (2) When an intersection ordinate occurs multiple times within $I(x_{lo}, x_{hi}]$. The first type of degeneracies can be handled by applying specific tie-breaking rules to the special case when $x_{lo} = -\infty$ or $x_{hi} = +\infty$ and to the case when x_{lo} and x_{hi} are finite. The tie-breaking rules introduced in our slope selection paper [11] also apply to our randomized algorithm(s) for the RM line estimator. (See *ibid.* for a detailed discussion.)

The second type of degeneracies introduces the following problem. Suppose that we seek the "median" intersection ordinate of each of the sampled lines of C within the interval $(x_{lo}, x_{hi}]$, and just over half of the ordinates within $I(x_{lo}, x_{hi}]$ are equal to x_{hi}. In sampling ordinates, we expect that about half will be equal to x_{hi} for the lines sampled. It is not unlikely, therefore, that for the next stage a new subinterval, $(x_{lo}', x_{hi}']$, will be generated such that $x_{hi}' = x_{hi}$.

In other words, since the interval is closed on the right side, no intersection ordinates will be eliminated from that side of the "median" ordinates.

As in the slope selection case, we note that the problem arises only when the "median" ordinates of sampled lines coincide with the right endpoint of the current interval. (Since the interval is open on the left, degeneracies at x_{lo} are ignored.) We solve the problem by altering the procedure RM_Inv_Count (Subsection 2.2) so that instead of counting the number of intersection ordinates $l_i.count$ for each sampled line in the half-open, half-closed interval $(x_{lo}, x_{hi}]$, it returns two counts for each sampled line, one for the open interval (x_{lo}, x_{hi}), denoted $l_i.count_1$, and one for the single point x_{hi}, denoted $l_i.count_2$. Obviously, $l_i.count = l_i.count_1 + l_i.count_2$. If $l_i.count_1 < \text{med}(n-1) - l_i.L$, then the median intersection ordinate of l_i is x_{hi}. Let C_2 denote the set of all lines in C which satisfy the above condition. If $|L| + |C_2| > n/2$, then we conclude that the RM ordinate is x_{hi} and terminate. Otherwise the algorithm is applied to the open interval (x_{lo}, x_{hi}).

The above modification can be accommodated with little additional effort. Applying RM_Inv_Count only to the half-open half-closed interval requires using a slightly different tie-breaking rule at $x = x_{hi}$. (The exact details are given in [11].) This modified RM_Inv_Count procedure returns the counts $l_i.count_1$ for $i = 1, \ldots, |C|$. To obtain the number of intersection ordinates at x_{hi} per each line in C, we scan the sorted list of y-coordinates at which the lines intersect the vertical line $x = x_{hi}$. These y-coordinates can be grouped into *bunches*

of equivalence classes in $O(n)$ time. It is important to note that each bunch of size m corresponds to m lines intersecting at a common point with x-coordinate x_{hi}, thereby implying that the counts $l_i.count_2$ for each of these lines should be set to $(m-1)$.

As in [11], the use of open intervals should not affect the proofs of the algorithm's correctness. Moreover, the asymptotic running time cannot deteriorate as a consequence, since processing the open interval can only reduce the number of ordinates to be considered at later stages of the algorithm. In practice, the above modification has little effect on the performance of the algorithm since the probability of degeneracies is small. In cases of (nearly) exact data (i.e., many degeneracies are present), the algorithm may terminate after fewer iterations than in nondegenerate cases, since intersection ordinates bunch heavily around the repeated median point.

5.2. Numerical Precision

The proofs of the algorithm's correctness have been based on the assumption that all calculations have been carried out using exact arithmetic. In practice, where calculations are typically performed using floating point representation, this assumption may not be valid. However, analogously to the treatment in our slope selection paper [11], it can be shown that if the input is presented to b bits of precision, $2b+3$ bits of precision suffice to determine the solution exactly. Even if (single precision) floating point calculations are carried out throughout, we can guarantee that the algorithm will succeed in finding an (imprecise) solu-

tion within the specified time complexity. (See [11] for a more detailed discussion concerning the handling of roundoff errors associated with floating point calculations.)

5.3. Choice of Parameters

We now discuss how to choose the various parameters which determine the probabilistic performance of the algorithm.

(1) Selecting P and Q was done in accordance with the intuitive guidelines described in Subsection 4.2. To maintain a certain statistical significance, i.e., to ensure reliable, representative samples both within each line and amongst the different lines, it is desirable to pick large enough values for P and Q. Picking P and Q in a balanced manner, one may choose, in principle, $P = K_P\sqrt{n}$ and $Q = K_Q\sqrt{n}$ for some large constants K_P, K_Q. Note, however, that picking P and Q values that are too large results in an additional (running time) overhead that could outweigh the advantage(s) gained by generating reliable samples in the first place.

(2) In principle it is possible to take $t_P \neq 0$, $t_Q \neq 0$, in order to maintain a "conservative" confidence interval. Our own experience shows, however, that $t_P = 0$ performs rather efficiently. Also, our empirical experience indicates that picking $t_Q = 3$ by analogy to the Theil-Sen case results in a reasonably good performance.

(3) The parameter c which determines whether the termination condition holds, i.e., whether or not $Tot\_Count \leq cn$ is met, should be determined by

similar considerations discussed in [11]. In principle, one faces the following tradeoff. One more iteration incurs the additional cost of counting and sampling, but then enumeration can be applied to a smaller set of intersection points.

In view of the above discussion and based on preliminary experiments we have carried out over various value ranges for the parameters in question, we chose the following set of values which seems to work reasonably well: $P = \sqrt{n}$, $Q = 2\sqrt{n}$, $t_P = 0$, $t_Q = 3$, and $c = 10$.

5.4. Experimental Results

We ran the algorithm on numerous data sets to study its overall performance and to establish a certain degree of statistical validity for the empirical results obtained. (The following input descriptions are stated in terms relevant to primal space.) The experiment considered 100 different data instances for each value of n. In each case, a set of n points associated with one of the following data distribution types was generated inside a unit square.

(1) Linear plus normal perturbation, i.e., a set of points associated with a specific line (characterized by its slope and vertical intercept) whose y-coordinates are perturbed with respect to the line by the value of a Gaussian randomly generated variable with standard deviation σ. (See Knuth 1981, pp. 116–117, for the algorithm used to produce these values.)

(2) Linear plus one-sided perturbation; like the above mentioned distribution only that the perturbation is one-sided, e.g., determined by the value of a

Weibull randomly generated variable with parameters γ and p [3]. (Generating values for a Weibull random variable is fairly simple since the (inverse of the) Weibull cumulative distribution function can be expressed in closed form.)

(3) Bimodal plus normal perturbation, i.e., a set of n points, n_1 of which are associated with one specified line, and n_2 $(= n-1)$ of which are associated with another specified line. The y-coordinate of each point is perturbed (with respect to one of the lines) by the value of a normal random variable with standard deviation σ.

(4) Circular, i.e., a set of points corresponding to a circular arc characterized by its center coordinates and radius.

(5) Uniform, i.e., a set of points uniformly (spatially) distributed in the unit square considered.

For each value of n, we ran our algorithm on 20 instances for each of the above types. The data instances generated correspond to lines of various slopes (and intercepts) and to various degrees of perturbation (e.g., $\sigma = 10^{-6}, 10^{-3}, 10^{-2}, 10^{-1}$). Examples of various data instances for $n = 225$ are shown in Figures 4(a)–(e). These figures depict data instances of the distribution types (1)–(5), respectively. Figures 4(a)–(b) correspond to the line $y = 0.5x + 0.25$, with perturbations $\sigma = 0.1$, and $<\gamma, p> = <100, 2>$, respectively, 4(c) corresponds to the lines $y = 0.5x + 0.25$ and $y = -0.5x + 0.75$ with a perturbation $\sigma = 0.01$, 4(d) corresponds to the circle $(x-0.5)^2 + (y-0.5)^2 = 0.25^2$, and 4(e) shows an

instance of uniform distribution. (The corresponding RM values given are the those computed by the algorithm for the repeated median estimate.)

For each data instance, we invoked our algorithm to find the RM estimate, and recorded the running times and the values of various parameters of interest (e.g., the number of iterations per each run, the number of times the algorithm has failed to capture the desired intersection ordinate, etc.) For comparison, we have also recorded the running times of the $O(n^2)$ brute force algorithm (which simply enumerates, for each point, all of its $(n-1)$ pairwise slopes, selects the median of each such subset, and then selects the median of the resulting n medians).

The results obtained are summarized in Figure 5. Specifically, Figure 5(a) depicts the (average, minimum, and maximum) number of iterations recorded per each value of n. Notice that for (relatively) small n values (e.g., $n \leq 225$), the (average, minimum, and maximum) number of iterations is greater than the respective number obtained for larger n values. Although the theoretical proofs provided in Subsection 4.3 suggest that the P-Q algorithm should terminate within four iterations (plus an enumeration stage), these proofs apply only in the "asymptotic" sense. For small n values, samples of size $O(\sqrt{n})$ are not statistically valid, i.e., the results obtained may not adequately reflect the algorithm's overall performance. (Note that for such cases, situations where $x_{lo}' = x_{lo}$ and/or $x_{hi}' = x_{hi}$ arise more often, i.e., more iterations are expected to take place.) To establish reliable running time models, we discarded, therefore, running times recorded for $n \leq 225$ from computing the models below. As in [11],

we used least squares to derive linear models for the running times of both algorithms. As only a constant number of iterations (per run) were recorded for n values ranging up to 40000, we assume an $O(n \log n)$ model for the running time of our algorithm. Letting $T_I(n)$ and $T_B(n)$ denote the running times (in milliseconds) of our inversion counting algorithm and the brute force algorithm, respectively, we obtained the following relations:

$$\frac{T_I(n)}{n} = 1.2\log_{10} n + 7.2,$$

$$\frac{T_B(n)}{n} = 0.05n + 1.9.$$

The running times have been scaled down by a factor of n to reduce the residuals for large n.

The above results are illustrated in Figures 5(b)–(d). (Logarithms are base 10.) Figure 5(b) shows $T_I(n)/n$, 5(c) shows $T_B(n)/n$, and 5(d) shows both fits superimposed. We conclude from these models that for $n > 10^{2.19} \approx 155$, our algorithm performs faster than the brute force algorithm. (To be more precise, we should extrapolate running times obtained for small n values and compare them with the brute force model. Carrying this out reveals that our algorithm performs faster for a slightly greater threshold, i.e., $n > 175$.)

As was mentioned previously, we also recorded the number of times the contraction stage has failed to trap the RM ordinate. With the choice of parameters made (see Subsection 5.3), it was observed that the RM ordinate was trapped successfully for over 99% of the cases.

6. Discussion

In this paper we presented several efficient algorithms for the 50% breakdown point RM line estimator. First, a simple presentation of a deterministic $O(n\log^2 n)$ time and $O(n)$ space algorithm was given. Noting the impracticality of underlying techniques this algorithm was based on, we introduced alternative randomized versions for computing the estimator. The randomized algorithms introduced always terminate giving the *correct* output within machine precision. We have shown that the algorithms derived require $O(n\log^2 n)$ expected running time. The characteristic features of our algorithmic approach are summarized as follows:

- The algorithms are fairly easy to implement, relying only on simple modifications of mergesort.
- The constants of proportionality hidden by the asymptotic notation are small.
- The $O(n\log^2 n)$ expected running time occurs with extremely high probability on any input of size n. Moreover, empirical results for a large number of data sets strongly suggest that, the expected running time of our P-Q algorithm is essentially $O(n\log n)$. We have proved that if the distributions of intersection ordinates on each line are assumed to be i.i.d., then the expected running time will indeed be $O(n\log n)$.
- The algorithms are space optimal, i.e., they require $O(n)$ storage.

In a companion paper (Mount and Netanyahu 1991), we have extended the approach pursued in this paper to higher dimensions. Specifically, we have shown that a d-dimensional RM estimator can be computed in (expected) $O(n^{d-1}\log^2 n)$ time and $O(n)$ space, for fixed d. Based on our encouraging experience regarding the two-dimensional case, we believe that our extended algorithmic approach is practical also in higher dimensions, at least for low values of d (e.g., $d = 3$).

Acknowledgements

We would like to thank Mike Dillencourt and Azriel Rosenfeld for their helpful remarks and valuable suggestions, Peter Rousseeuw for his encouragement and genuine interest in our research, and Chris Welsh for providing useful assistance in processing our experimental results.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi (1983), Sorting in $c \log n$ Parallel Steps, *Combinatorica*, **3**, pp. 1–19.

[2] D.F. Andrews (1974), A Robust Method for Multiple Linear Regression, *Technometrics*, **16**, pp. 523–531.

[3] I.N. Bronshtein and K.A. Semendyayev (1985), *Handbook of Mathematics*, English translation edited by K.A. Hirsch, Van Nostrand, New York.

[4] G.W. Brown and A.M. Mood (1951), On Median Tests for Linear Hypotheses, in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, edited by J. Neyman, University of California Press, Berkeley and Los Angeles, pp. 159–166.

[5] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan (1973), Time Bounds for Selection, *Journal of Computer and System Sciences*, **7**, pp. 448–461.

[6] R. Cole (1987), Slowing Down Sorting Networks to Obtain Faster Sorting Algorithms, *Journal of the ACM*, **34**, pp. 200–208.

[7] R. Cole, J.S. Salowe, W.L. Steiger, and E. Szemerédi (1989), An Optimal-Time Algorithm for Slope Selection, *SIAM Journal on Computing*, **18**, pp. 792–810.

[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest (1990), *Introduction to Algorithms*, MIT Press, McGraw-Hill, New York.

[9] G. Dahlquist and A. Björck (1974), *Numerical Methods*, translated by N. Anderson, Prentice-Hall, Englewood Cliffs, New Jersey.

[10] D.L. Donoho and P.J. Huber (1983), The Notion of Breakdown Point, in *A Festschrift for Eric L. Lehman*, edited by P.J. Bickel, K. Doksun, and J.L. Hodges, Jr., Wadsworth International Group, Belmont, California, pp. 157–184.

[11] M.B. Dillencourt, D.M. Mount, and N.S. Netanyahu (1991), A Randomized Algorithm for Slope Selection, in *Proceedings of the Third Canadian Conference on Computational Geometry*, Vancouver, Canada, August 1991, pp. 135–140; CS-TR-2431, Center for Automation Research, University of

Maryland, March 1990; to appear in *International Journal on Computational Geometry and Applications.*

[12] H. Edelsbrunner and E.P. Mücke (1990), Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, *ACM Transactions on Graphics*, **9**, pp. 66–104.

[13] H. Edelsbrunner and D.L. Souvaine (1990), Computing Median-of-Squares Regression Lines and Guided Topological Sweep, *Journal of the American Statistical Association*, **85**, pp. 115–119.

[14] C.A.R. Hoare (1962), Quicksort, *Computer Journal*, **5**, pp. 10–15.

[15] C.A.R. Hoare (1970), Proof of a Program: FIND, *Communications of the ACM*, **13**, pp. 39–45.

[16] H. Imai, K. Kato, and P. Yamamoto (1989), A Linear-Time Algorithm for Linear L_1 Approximation of Points, *Algorithmica*, **4**, pp. 77–96.

[17] D.E. Knuth (1973), *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, Massachusetts.

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

[19] J. Matoušek (1991), Randomized Optimal Algorithm for Slope Selection, *Information Processing Letters*, **39**, pp. 183–187.

[20] N. Megiddo (1983a), Applying Parallel Computation Algorithms in the Design of Serial Algorithms, *Journal of the ACM*, **30**, pp. 852–865.

[21] N. Megiddo (1983b), Linear Time Algorithms for Linear Programming in R^3 and Related Problems, *SIAM Journal of Computing*, **12**, pp. 759–776.

[22] N. Megiddo (1984), Linear Programming in Linear Time when the Dimension is Fixed, *Journal of the ACM*, **31**, pp. 114–127.

[23] D.M. Mount and N.S. Netanyahu (1991), Computationally Efficient Algorithms for High-Dimensional Robust Estimators, Technical Report in preparation, Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland.

[24] P.J. Rousseeuw (1984), Least Median of Squares Regression, *Journal of the American Statistical Association*, **79**, pp. 871–880.

[25] P.J. Rousseeuw and A.M. Leroy (1987), *Robust Regression and Outlier Detection*, John Wiley & Sons, New York.

[26] P.K. Sen (1968), Estimates of the Regression Coefficient Based on Kendall's Tau, *Journal of the American Statistical Association*, **63**, pp. 1379–1389.

[27] M.I. Shamos (1978), *Computational Geometry*, Ph.D. Thesis, Yale University.

[28] A.F. Siegel (1982), Robust Regression Using Repeated Medians, *Biometrika*, **69**, pp. 242–244.

[29] D.L. Souvaine and J.M. Steele (1987), Time- and Space-Efficient Algorithms for Least Median of Squares Regression, *Journal of the American Statistical Association*, **82**, pp. 794–801.

[30] A. Stein and M. Werman (1992), Finding the Repeated Median Regression Line, to appear in *Proceedings of the Third Annual Symposium on Discrete Algorithms*, Orlando, Florida, January 1992.

[31] H. Theil (1950), A Rank-Invariant Method of Linear and Polynomial Regression Analysis (Parts 1–3), *Nederlandse Akademie Wetenschappen Series A*, **53**, pp. 386–392, 521–525, 1397–1412.

[32] J.W. Tukey (1970/1971), *Exploratory Data Analysis* (Limited Preliminary Edition), Addison-Wesley, Reading, Massachusetts.

[33] V.J. Yohai (1987), High Breakdown-Point and High Efficiency Robust Estimates for Regression, *Annals of Statistics*, **15**, pp. 642–656.

[34] V.J. Yohai and R.H. Zamar (1988), High Breakdown-Point Estimates of Regression by Means of the Minimization of an Efficient Scale, *Journal of the American Statistical Association*, **83**, pp. 406–413.

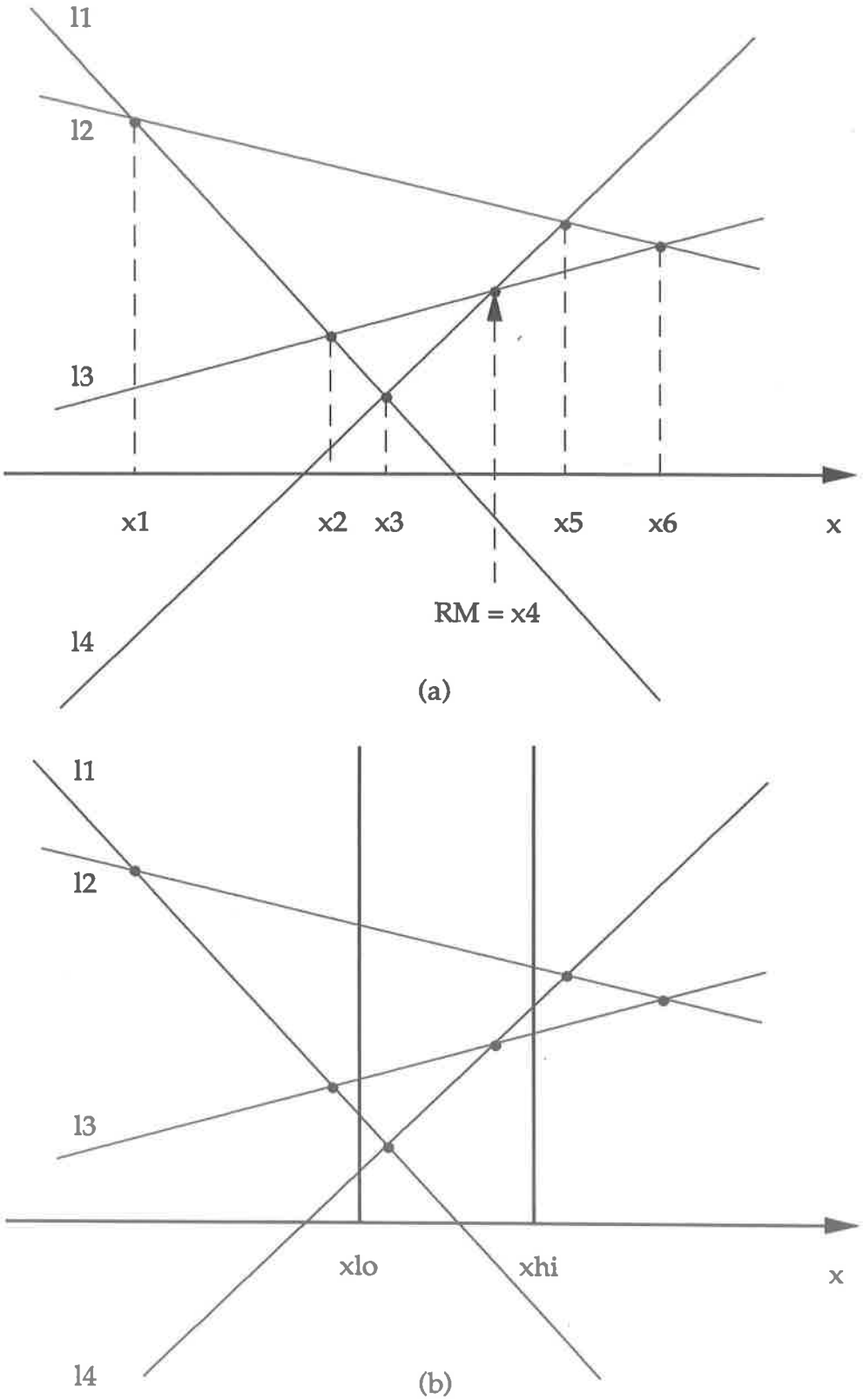


Figure 1: Finding the RM in dual space

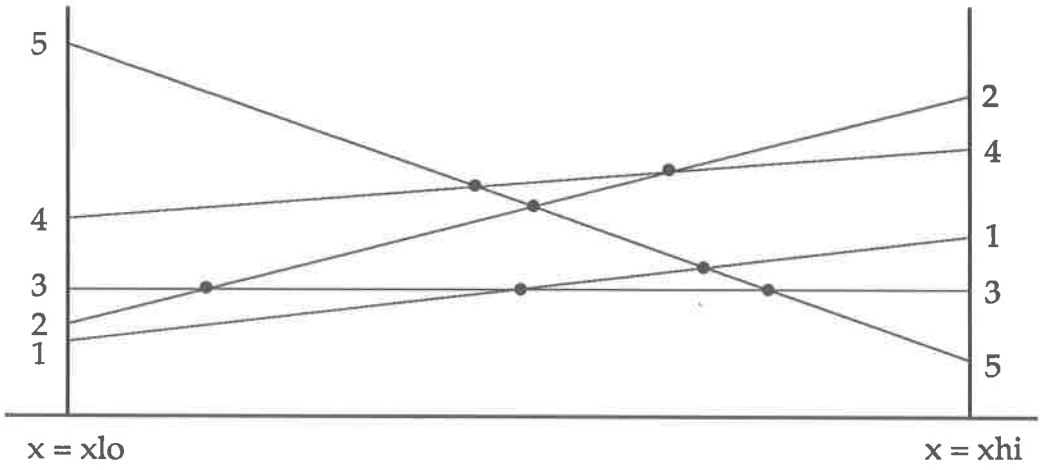
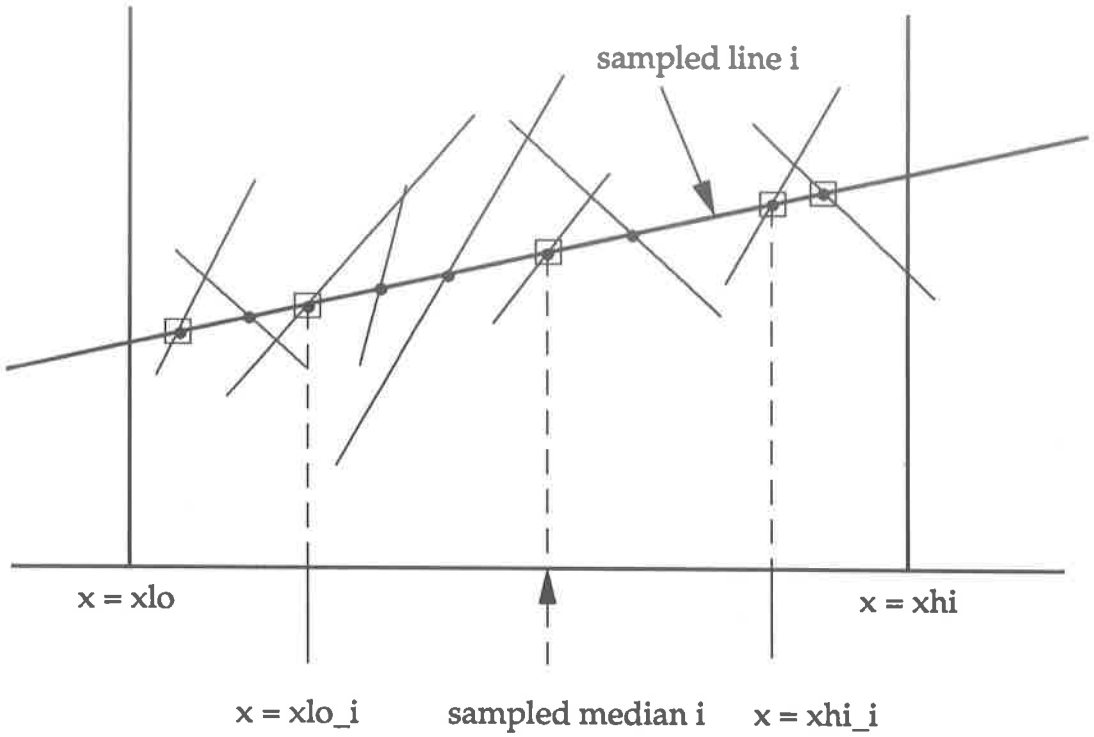


Figure 2: Inversions and line intersections



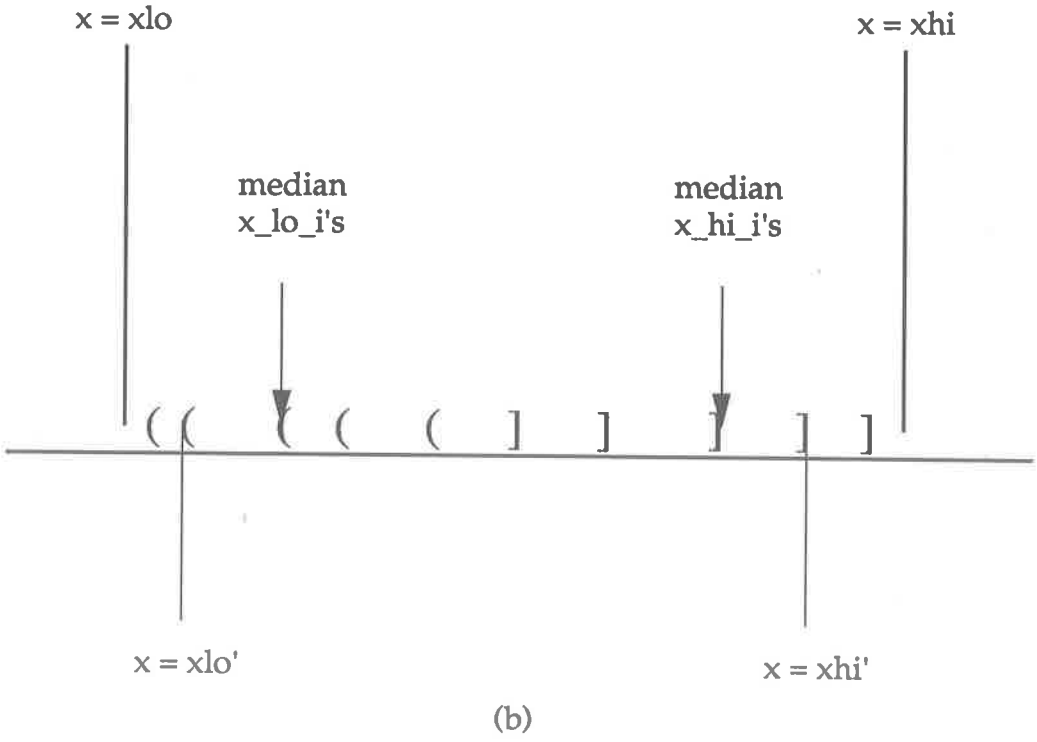


Figure 3: Interval contraction for the RM line estimator

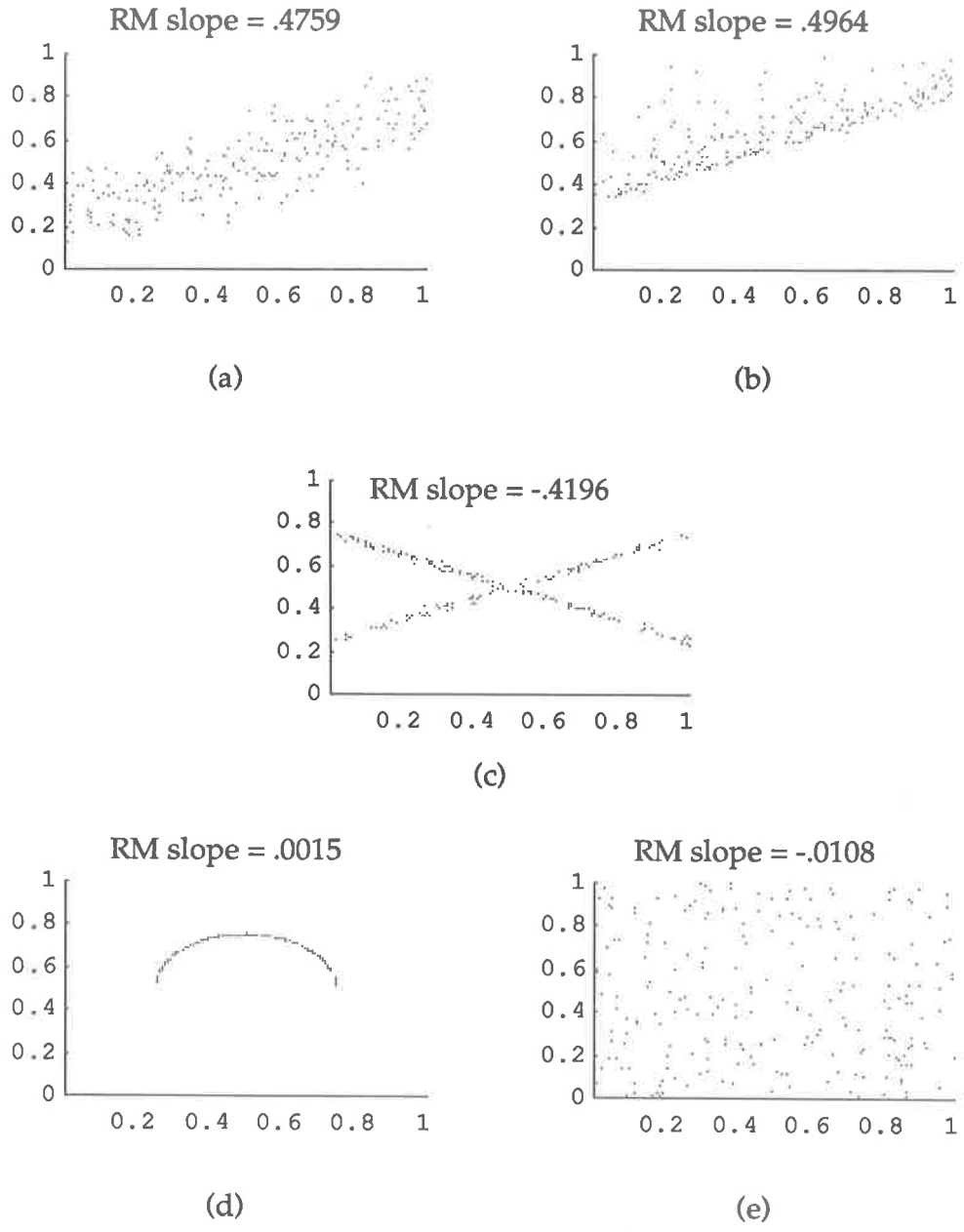
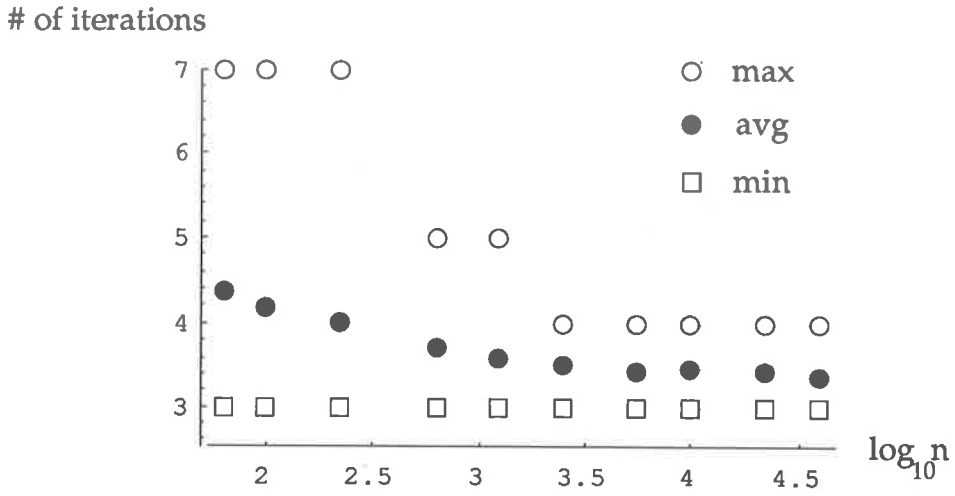
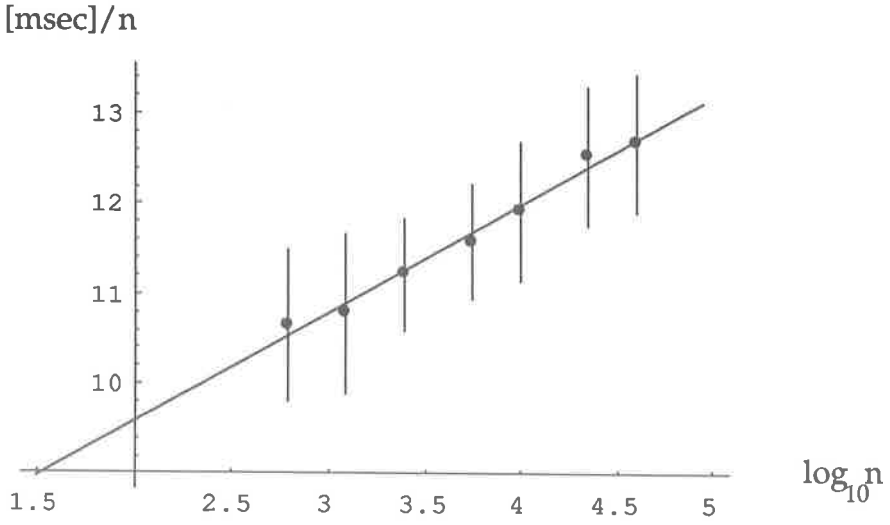


Figure 4: Examples of experimental data sets

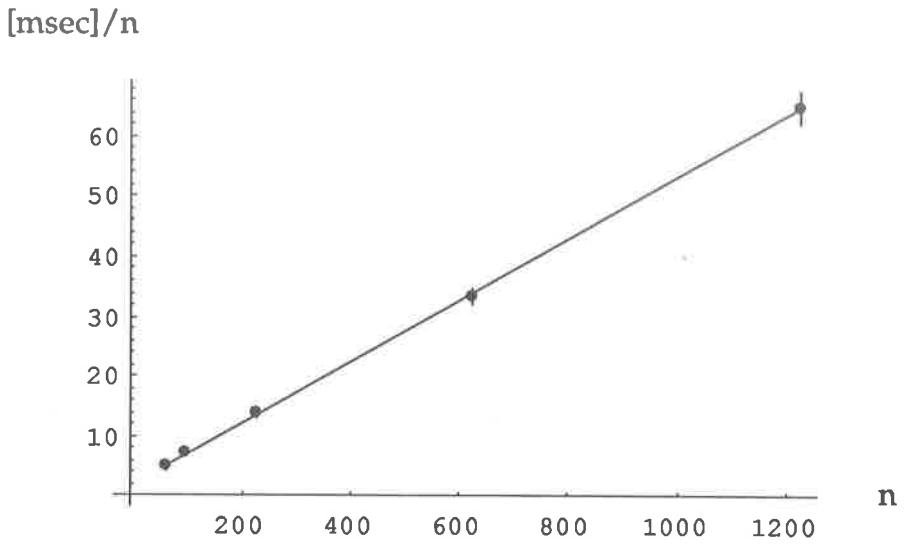


(a)

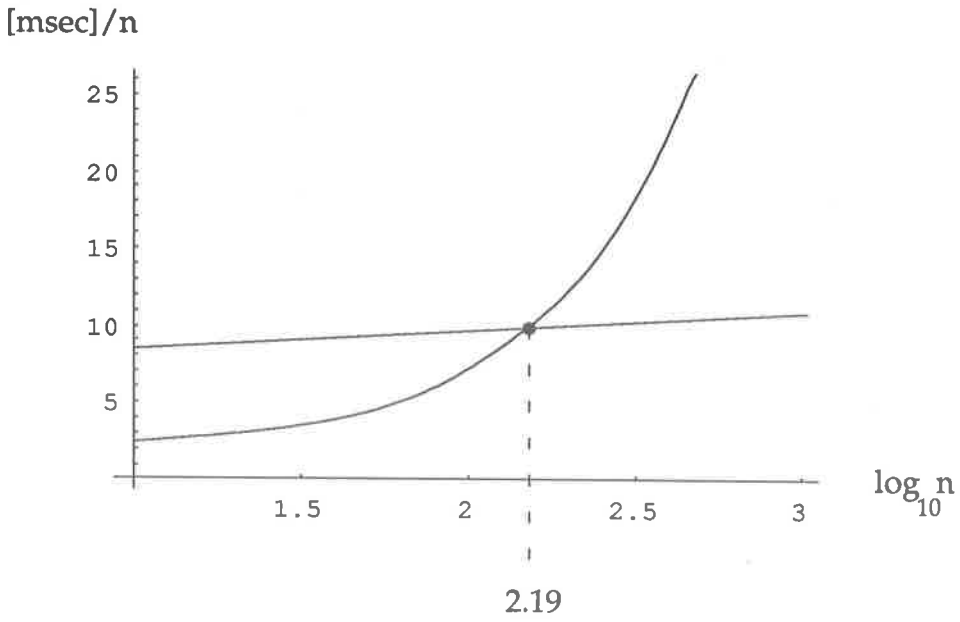


(b)

Figure 5: Number of iterations and comparison between the P-Q and brute force algorithms.



(c)



(d)

Figure 5 continued