A Randomized Algorithm for Slope Selection

Michael B. Dillencourt[*]
Department of Information and Computer Science
University of California, Irvine

David M. Mount[†]
Department of Computer Science
University of Maryland at College Park

Nathan S. Netanyahu[‡]
Center for Automation Research
University of Maryland at College Park

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 simple randomized algorithm for selecting the k-th smallest slope of such a set of lines which runs in expected $O(n \log n)$ time.

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 determining the k-th smallest slope among these lines. This problem was posed by Shamos [12]. It was subsequently considered by Cole, Salowe, Steiger, and Szemeredi [3], who discovered an optimal (deterministic) $O(n \log n)$ algorithm for this problem. Their algorithm relies on two ingenious but complicated techniques: Cole's improvement of Meggido'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. In the full paper [5], we also describe an implementation of the algorithm, and demonstrate the algorithm's practical efficiency. While our time bound is theoretically weaker than the deterministic bound of [3], we feel that our algorithm will be of greater practical interest, 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 on any input of size n. The algorithm always terminates giving the correct output in worst case $O(n^2)$ time (although the probability of achieving this worst case is extremely small for large n).

The slope selection 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 as a robust estimator of the slope of the line fitting the points [14]. 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 percentage of outlying data points that may cause the estimator to take on an arbitrarily large aberrant value. (See [6] or for an exact definition.) It can easily be shown that the Theil estimator has a breakdown point of $1 - \sqrt{1/2} \approx 29.3\%$. Other common estimators include the L_2 (least squares) estimator,

[*]The support of the National Science Foundation under Grant IRI-88-02457 is gratefully acknowledged.

[†]The support of the National Science Foundation under Grant CCR-8908901 is gratefully acknowledged.

[‡]The support of the Defense Advanced Research Projects Agency (ARPA Order No. 6989) and the U.S. Army Engineer Topographic Laboratories under Contract DACA76-89-C-0019 is gratefully acknowledged.

the L_1 estimator, and the L_∞ (Chebyshev) estimator. These estimators can be computed in $O(n)$ time (see [4,9,10]) but each has a breakdown point of zero because a single outlier can bias the estimator arbitrarily. The line minimizing the median of the squared residuals, *least median of squares* (LMS) [11], was shown to be computable in $O(n^2)$ time and $O(n^2)$ space in [13], and the space bound was improved to $O(n)$ in [8]. The breakdown point for LMS is 50%. Although this is better than the Theil estimator, the computation time is significantly larger. Our motivation for studying this problem arises from the problem in computer vision of fitting a line to a set of points in an image.

Our algorithm is based on the subtasks of counting and sampling from a set of inversions in a list. 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.

2 Inversion Counting and Sampling

Let (a_i, b_i), for $1 \le 1 \le n$, 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 make the convention that a vertical line has the largest possible slope, $+\infty$. As observed in [3] 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 $y = a_ix - b_i$ and $y = a_jx - b_j$, respectively, intersect at the x-coordinate $x = (b_i - b_j)/(a_i - a_j)$. 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 $x = +\infty$. (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 to compute the median y-intercept of the set of lines determined by all pairs of points. Selection of parameters for other line representations can be derived by appropriate use of other dual transformations.)

We have now reduced the slope selection problem to the following dual form. Given a set of n lines $y = a_ix - b_i$, $1 \le i \le n$, 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.

Our basic approach is to maintain two x-values, x_{lo} and x_{hi}, $-\infty \le x_{lo} \le x_{hi} \le +\infty$. Let $(x_{lo}, x_{hi}]$ denote the half-open, half-closed interval of points x, $x_{lo} < x \le x_{hi}$. 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 can 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: (a) no two intersection ordinates are equal to one another, and (b) 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 the full paper [5] we discuss how to augment the algorithm to handle these degeneracies properly.

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 $y_1, y_2, \ldots, y_n$, to be a pair of values, y_i and y_j where $i < j$ but y_i is greater than y_j. Index lines in order according to the y-coordinate at which they intersect the right end of the interval, $x = x_{hi}$. Label each line with the y-coordinate at which it intersects the left end of the interval, $x = x_{lo}$. Consider the resulting list of labels in index order. Observe 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 Fig. 1).

The number of inversions can be counted in $O(n \log n)$ worst-case time, using a simple modification of mergesort. Once the count, say C, is known, a sample of size n (with replacement) can be generated in $O(n \log n)$ time by (1) generating a collection of n random integers in the range from 1 to C (allowing duplicates), (2) sorting this collection of integers, and (3) running a slightly modified version of the counting algorithm. The algorithms for counting and sampling inversions are not described in this extended abstract due to space limitations.

3 Contracting the Interval

In this section we introduce the necessary probability-theoretic groundwork on which the algorithm is based. We omit the proofs of the two lemmas of this section, which can be found in [5]. 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 $I(x_{lo}, x_{hi}]$, and that $C \geq n$. (The case when C is smaller will be discussed in the next section.) We begin by applying the sampling procedure of Section 2 to sample (with replacement) a subset of n intersection ordinates from $I(x_{lo}, x_{hi}]$. Let S denote this subset. Let x^* denote the (unknown) k-th smallest intersection ordinate in $I(x_{lo}, x_{hi}]$, and let k^* denote the nearest integer to kn/C. Because the sample is random, the k^*-th smallest element in the sample should be "close" to x^*. More precisely, let $S[1], S[2], \ldots, S[n]$ denote the elements of the sample, sorted in increasing order. For some positive constant t (whose exact value is discussed in the full version of the paper) 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:

$$k_{lo} = \left\lfloor \frac{kn}{C} - t\frac{\sqrt{n}}{2} \right\rfloor \quad \text{and} \quad k_{hi} = \left\lceil \frac{kn}{C} + t\frac{\sqrt{n}}{2} \right\rceil .$$

(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 show in the full paper [5], 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 $x'_{lo} = S[k_{lo}]$ and $x'_{hi} = S[k_{hi}]$. If $k_{lo} < 1$ then let $x'_{lo} = x_{lo}$ and if $k_{hi} > n$ let $x'_{hi} = x_{hi}$. The process is roughly illustrated in Fig. 2. The following lemma says that, 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 increases, and hence the running time of the algorithm also increases.

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 intersection ordinate, x^*. 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, as a function of ϵ, $t \in O(\sqrt{\ln(1/\epsilon)})$.*

In order 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 states that for any constant probability bound, 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 *Assume the same hypotheses as in 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.*

4 Complete Algorithm

As we mentioned earlier, the algorithm consists of a series of stages. At the start of each stage we are given an interval $(x_{lo}, x_{hi}]$ within which we seek the k-th smallest intersection ordinate. We are also 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 enumerate all inversion ordinates in the interval. We then use Hoare's $O(n)$ expected-time selection algorithm (using a randomly chosen pivot element) [1] to determine the k-th smallest value.

If, on the other hand, $C > cn$, we apply the sampling procedure described earlier to select a sample S of n intersection ordinates. We sort these intersection ordinates, and we set x'_{lo} to be the k_{lo}-th smallest and 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 $I(x'_{lo}, x'_{hi}]$. It is an easy matter to apply the inversion counting procedure to ascertain whether the k-smallest ordinate occurs within $I(x'_{lo}, x'_{hi}]$, as expected, or to the right or left of this subinterval. If it lies within this central subinterval, we invoke the algorithm recursively on $I(x'_{lo}, x'_{hi}]$, and if not then we apply the algorithm to either the left subinterval $I(x_{lo}, x'_{lo}]$ or the right subinterval $I(x'_{hi}, x_{hi}]$ as appropriate. The overall algorithm follows from this description. We omit details from this extended abstract.

In order 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. As we mentioned in Section 2, the running time of each stage can be bounded above by $en \log n$ for some constant e. 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

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

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).

This suggests the following mode of operating the algorithm. By selecting t and d such that the probability of success of any one stage of the algorithm is very high, say 0.99, we are assured that after two stages the algorithm terminates successfully with very high probability, say 0.98. In the complete paper we discuss the choice of these parameters for our implementation. Once t and d have been selected, c is selected so that $c \geq d^2/2$. With high probability, all stages will then be successful and 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). Hence, the running time of the algorithm is given by the recurrence

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

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 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 [2] we have experimented extensively with the implementation in order to establish its efficiency and robustness. (See [5] for a detailed discussion.)

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[2] J. L. Bentley. Experiments on traveling salesman heuristics. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 91–99, San Francisco, CA, January 1990.

[3] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, August 1989.

[4] G. Dahlquist and A. Björck. *Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1974. Translated by N. Anderson.

[5] M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. Computer Science Technical Report CS-TR-2431, University of Maryland, College Park, MD, March 1990.

[6] D. L. Donoho and P. J. Huber. The notion of breakdown point. In P. J. Bickel, K. Doksun, and J. L. Hodges, Jr., editors, *A Festschrift for Erich L. Lehman*, pages 157–184. Wadsworth International Group, Belmont, CA, 1983.

[7] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric altorithms. *ACM Transactions on Graphics*, 9(1):66–104, January 1990.

[8] H. Edelsbrunner and D. L. Souvaine. Computing median-of-squares regression lines and guided topological sweep. *Journal of the American Statistical Association*, 85(409):115–119, March 1990.

[9] H. Imai, K. Kato, and P. Yamamato. A linear-time algorithm for linear L_1 approximation of points. *Algorithmica*, 4(1):77–96, 1989.

[10] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, November 1983.

[11] P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871-880, December 1984.

[12] M. I. Shamos. Geometry and statistics: Problems at the interface. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 251–280. Academic Press, 1976.

[13] D. L. Souvaine and J. M. Steele. Time- and space-efficient algorithms for least median of squares regression. *Journal of the American Statistical Association*, 82(399):794–801, September 1987.

[14] H. Theil. A rank-invariant method of linear and polynomial regression analysis I. *Proc. Koninklijke Nederlandse Akademie van Wetenschappen*, 53(3):386–392, 1950.

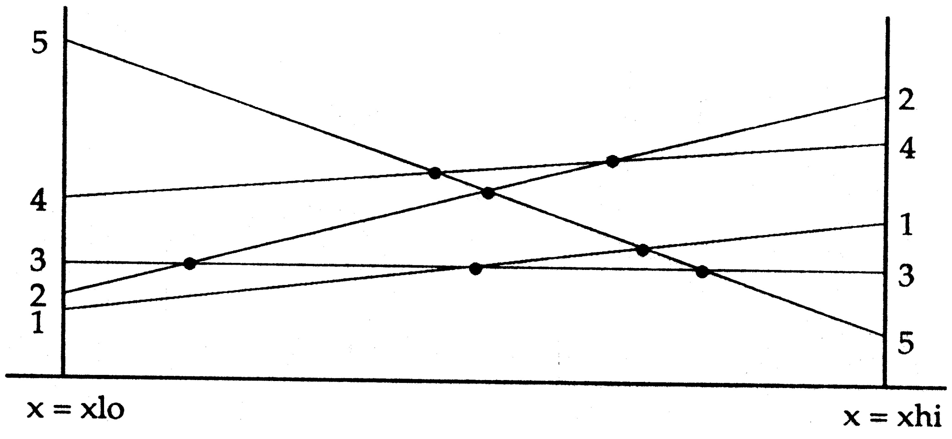

Figure 1: Inversions and line intersections.

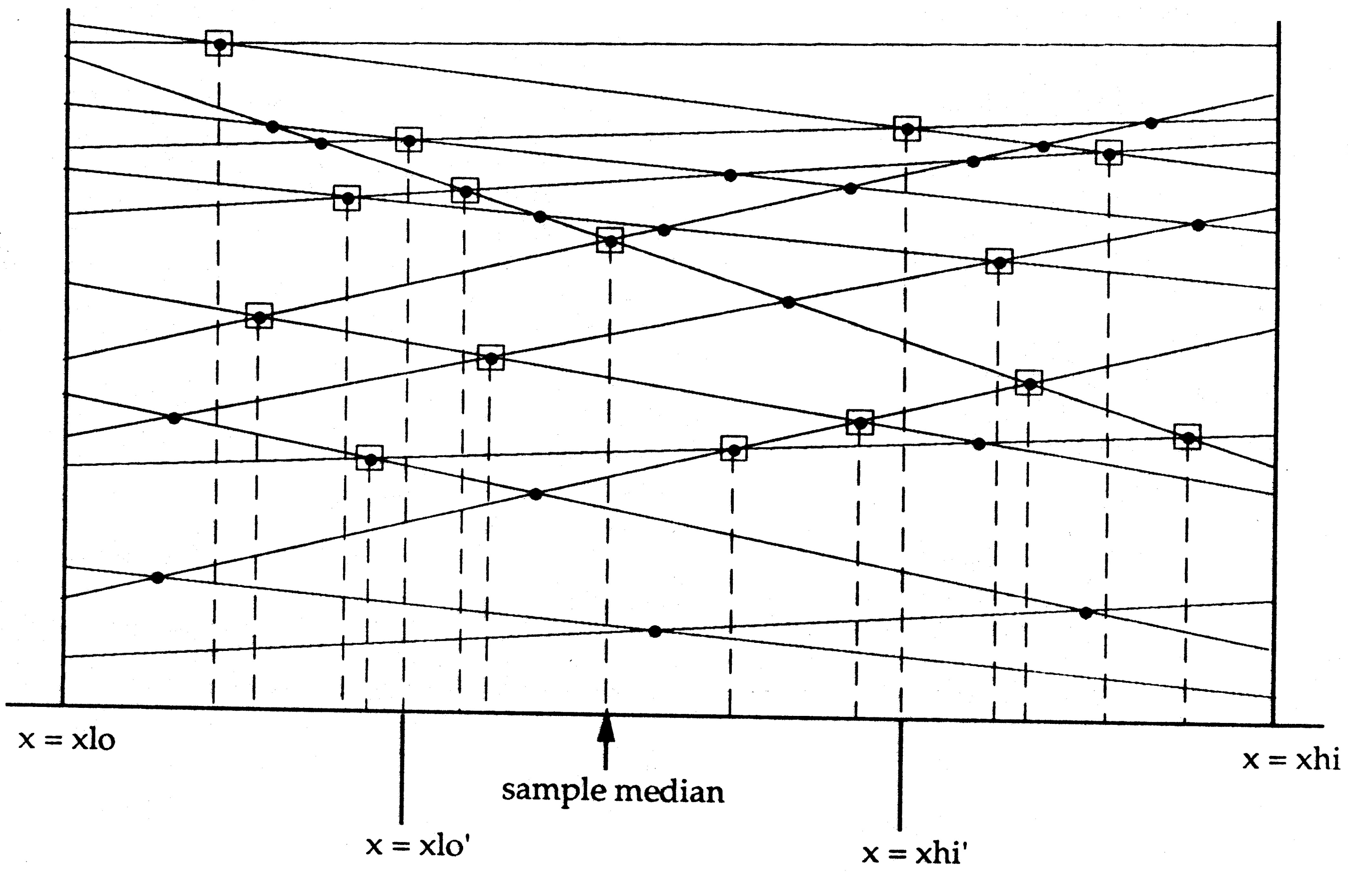

Figure 2: Illustration of the sampling procedure.