JOURNAL OF ALGORITHMS 6, 430–433 (1985)

NOTE

Partitioning with Two Lines in the Plane*

NIMROD MEGIDDO

Stanford University, Stanford, California 94305, The IBM Almaden Research Center, San Jose, California 95120, and Tel Aviv University, Israel

Received April 5, 1984; revised June 26, 1984

An $O(n)$ algorithm is presented for the problem of partitioning a set of n points in the plane into four equal parts by means of two straight lines.

1. INTRODUCTION

The following problem was raised in a paper by Dan Willard [6]: Given n points (a_i, b_i) $(i = 1, \ldots, n)$, find two straight lines that partition the plane in such a way that each (closed) quadrant contains at least $\lfloor n/4 \rfloor$ points. Cole, Sharir, and Yap [1] propose an algorithm which is based on the scheme for applying parallel algorithms in the design of serial ones that is presented in [3]. Their algorithm runs in $O(n \log^2 n)$ but the authors argue that an $O(n \log n)$ algorithm is possible if the guiding routine is a sorting network of depth n with bounded degree. Here we show that the ideas presented in [2, 4] apply to the two-line partitioning problem as well, so we obtain here an $O(n)$ algorithm for the latter.

2. PRELIMINARIES

We first argue that a crucial subproblem is a special case of what is called the "ham sandwich problem." The special case is the following: Given two finite sets of points in the plane, whose convex hulls are disjoint, find a

*Supported in part by the National Science Foundation under Grants MCS-8300984, ECS-8218181, and ECS-8121741.

0196-6774/85 $3.00

straight line that simultaneously bisects both of the sets. The relation of the latter to the original partitioning problem is as follows. Given the set of n points, first find any straight line that bisects it. The problem now reduces to bisecting the two halves simultaneously by means of a second line. Obviously, the convex hulls of the two halves are disjoint if the first line does not contain any of the input points.

Without loss of generality it is sufficient to deal with the following problem: Given are two sets of points $P = \{(a_1, b_1), \ldots, (a_m, b_m)\}$, where $b_i > 0$ $(i = 1, \ldots, m)$, and $Q = \{(c_1, d_1), \ldots, (c_n, d_n)\}$, where $d_i < 0$ $(i = 1, \ldots, n)$. We have to find an x such that the median of the set $P_s = \{(a_i - x)/b_i : i = 1, \ldots, m\}$ equals the median of the set $Q_s = \{(c_i - x)/d_i : i = 1, \ldots, n\}$. It is thus more convenient (at least to the author's taste) to consider the problem in the following equivalent form. Given are two sets of linear functions. One consists of m increasing linear functions while the other consists of n decreasing ones. Find the point where the two point-wise median-functions intersect. Obviously, the first median-function is monotone increasing while the second is monotone decreasing. Thus, we are searching for a well-defined point in the plane, namely, the unique intersection point of the two median-functions. We will develop a linear-time algorithm for this search problem.

The linear-time search here resembles the search procedure for the solution of a three-variable linear programming problem [2, 4]. The reader is strongly advised to refer to [2, 4] for more detail and illustrations. A rough description is as follows. A vital subroutine for the algorithm is a procedure that determines, for any straight line in the plane, on what side of the line lies the intersection point of the two median-functions. The problem is then solved by repeatedly discarding lines from the sets. It turns out that by testing two special straight lines one can tell the "correct" side of $(m' + n')/8$ lines, where m' and n' are the cardinalities of the current sets of lines. For the reader who is familiar with the technique in [2, 4], all we need to show is how to implement the line-query and how to select the two lines during an iteration of the algorithm.

3. The Line Query

In this section we address the following problem. Given are two sets of linear functions $P_l = \{y = A_i x + B_i : i = 1, \ldots, m\}$ and $Q_l = \{y = C_i x + D_i : i = 1, \ldots, n\}$ such that all the A_is are positive while all the C_is are negative. Also, let k $(1 \le k \le m)$ and r $(1 \le r \le n)$ be given. Denote by $P_k(x)$ the kth largest value in the set $\{A_i x + B_i : i = 1, \ldots, m\}$ and by $Q_r(x)$ the rth largest value in the set $\{C_i x + D_i : i = 1, \ldots, n\}$. Let (x^*, y^*) denote the point at which the two quantile-functions intersect, that is,

$y^* = P_k(x^*) = Q_r(x^*)$. Given any line L, determined by an equation $y = \alpha x + \beta$, we need to decide whether $y^* < \alpha x^* + \beta$, $y^* = \alpha x^* + \beta$ or $y^* > \alpha x^* + \beta$. For a vertical line of the form $x = x'$, recognizing whether $x^* < x'$, $x^* = x'$ or $x^* > x'$ is easy so we omit the details. Similarly, the case of a horizontal line ($\alpha = 0$) is easy.

We discuss here only the case where $\alpha > 0$. The case of $\alpha < 0$ is analogous. The first step is to find the intersection of the line L with $Q_r(x)$. Our assumptions imply that there is a unique intersection point whose x value, x', is precisely the rth largest of the x values of the intersection points of lines in Q_l with L. Thus, x' can be found in linear time. Let $y' = \alpha x' + \beta$. Next, we compute $y^P = P_k(x')$ in linear time. Obviously, if $y^P < y'$ then $x^* > x'$. However, since $Q_r(x)$ is decreasing and L is increasing, it follows that $y^* < \alpha x^* + \beta$. Analogously, if $y^P > y'$ then $y^* > \alpha x^* + \beta$ and if $y^P = y'$ then $y^* = \alpha x^* + \beta$. We conclude that the position of (x^*, y^*) relative to any straight line can be determined in linear time.

4. The Search Algorithm

An iteration of the search algorithm is organized as follows. Consider the set of lines which are active at the beginning of the iteration. A line is active if we have not determined on which side of the line the point (x^*, y^*) lies. At the start of the algorithm all the lines are active and each iteration deactivates $\frac{1}{8}$ of the active lines. Let N denote the number of active lines at the beginning of the current iteration. The first step is to partition the active lines according to their slopes into two sets, namely, those with slope larger than and smaller than the median slope s. This is accomplished in linear time. Second, the active lines are paired so that each pair has a member with a larger (than s) slope and a member with a smaller slope. Now, the x values of the intersection points of paired lines are computed and the median x_m of these values is found. Next, the line $x = x_m$ is tested in the sense of Section 3. Suppose, for example, we found that $x^* < x_m$. We now consider only those pairs whose intersection points have x values greater than or equal to x_m. There are at least $\lfloor N/4 \rfloor$ such pairs. The second line that we query is one that has a slope of s and such that it divides the intersection points under consideration into two equal sets. Thus, the intercept of this line equals the median of the set $\{y - sx\}$, where (x, y) runs over the intersection points still under consideration. By testing this line we identify a quadrant Q in which (x^*, y^*) lies; the $\lfloor N/8 \rfloor$ pairs of lines associated with the "opposite" quadrant have the property that each pair has one member that does not cross through Q. Each such member is a line for which we know on which side (x^*, y^*) lies. Hence, such a line can

be deactivated at this point and the problem is reduced to a similar one (with different values for the ranks k and r) on no more than $7N/8$ lines. This implies that the whole algorithm runs in linear time.

REFERENCES

1. R. COLE, M. SHARIR, AND C. YAP, On k-hulls and related problems, *in "Proceedings of the 16th Annual* ACM *Symposium on Theory of Computing,"* Assoc. Comput. Mach., pp. 154–66, New York, 1984.
2. M. E. DYER, Linear time algorithms for two- and three-variable linear programs, *SIAM J. Comput.* **13** (1984), 31–45.
3. N. MEGIDDO, Applying parallel computation algorithms in the design of serial algorithms, *J. Assoc. Comput. Mach.* **30** (1983), 852–865.
4. N. MEGIDDO, Linear time algorithms for linear programming in R^3 and related problems, *SIAM J. Comput.* **12** (1983), 759–776.
5. N. MEGIDDO, Linear programming in linear time when the dimension is fixed, *J. Assoc. Comput. Mach.* **31** (1984), 114–127.
6. D. E. WILLARD, Polygon retrieval, *SIAM J. Comput.* **11** (1982), 149–165.
7. F. F. YAO, A 3-space partition and its applications, *in "Proceedings of the 15th Annual* ACM *Symposium on Theory of Computing,* Assoc. Comput. Mach., pp. 258–263, New York, 1983.