Information Processing Letters 23 (1986) 289–293
North-Holland 3 December 1986

HALFPLANAR RANGE SEARCH IN LINEAR SPACE AND $O(n^{0.695})$ QUERY TIME

Herbert EDELSBRUNNER and Emo WELZL

Institute for Information Processing (IIG), Technical University of Graz and Austrian Computer Society, Schiesstattgasse 4a, A-8010 Graz, Austria

Communicated by L. Boasson
Received August 1985
Revised January 1986

Let S denote a set of n points in the Euclidean plane. A halfplanar range query specifies a halfplane h and requires the determination of the number of points in S which are contained in h. A new data structure is described which stores S in O(n) space and allows us to answer a halfplanar range query in $O(n^{\log_2(1+\sqrt{5})-1})$ time in the worst case, thus improving the best result known before. The structure can be built in O(n log n) time.

Keywords: Computational geometry, range search

1. Introduction

Let S denote a set of n points in the Euclidean plane. A *range search problem* requires to store S in a computer such that the number of points in S which are contained in a later specified region of the plane (called a *range*) can be determined efficiently. Usually, range search problems are classified by the kind of ranges which are specified. The classical *orthogonal range search problem* which prepares S for axis-parallel rectangles as ranges is already mentioned in [8].

We consider halfplanes as ranges, thus investigating the *halfplanar range search problem*. A new data structure, called the *conjugation tree* which will be introduced in Section 2, stores S in O(n) space such that $O(n^{0.695})$ time suffices to determine the number of points in S which lie in a query halfplane h. The application of the data structure is not restricted to computing the number of points in h. More generally, it can be used to compute the sum of values in some semigroup which are associated with the points contained in h.

The data structure also supports range queries for polygonal areas as query ranges. The query time remains the same if

(1) two values in the semigroup can be added in constant time, and

(2) the polygonal range is bounded by at most some constant number of edges.

In addition, the conjugation tree supports reporting the points in a halfplane or in a polygonal range bounded by at most a constant number of edges. The time required is $O(n^{0.695} + t)$ where t is the number of points to be reported. For reporting the points in a halfplane, this result is outperformed by an O(log n + t) query time and O(n) space data structure due to Chazelle et al. [2] (which, however, cannot be used for enumeration and polygonal ranges within this time bounds).

The best data structure for the halfplanar range search problem known before which requires only O(n) space is the polygon tree of Willard [12]. It stores n points in O(n) space and answers a range query of that kind in $O(n^{0.77})$ time. Thus, the conjugation tree is more efficient. Both trees are generalizations of the quad tree of Finkel and Bentley [5] and the kd-tree of Bentley [1] which were designed for orthogonal range queries. There exist data structures which are better than the conjugation tree as far as query time is concerned (see [3]). These data structures, however, require $\Omega(n^2)$ space for n points. It is also worthwhile to note that better results than the ones for the conjugation tree exist if only estimates of the number of points in a query halfplane are asked for. Data structures for these problems which achieve O(log n) query time and even sublinear space (depending on the required accuracy of the answer) are presented in [4]. For lower bounds on the halfplanar range search problem we refer to Fredman [6]. Unfortunately, it is not clear at this point how relevant Fredman's findings are for the material presented in this paper.

The organization of the paper is as follows: Section 2 describes the conjugation tree, analyzes the amount of time required to answer a halfplanar range query, and addresses a method for constructing the conjugation tree. Finally, Section 3 discusses some extensions and related problems.

2. The conjugation tree

This section introduces a new data structure, the so-called *conjugation tree*. The conjugation tree can be viewed as a binary tree which stores the points of a finite set in linear lists associated with its nodes. Preceding its detailed description, some geometric facts and notions are presented which are considered to be necessary for this purpose.

Intuitively, for a given point-set S and a line L, which divides S into two nearly equal-sized subsets, lines L′ are examined such that the four (infinite) regions defined by L and L′ contain nearly equal-sized subsets of S (see Fig. 1).

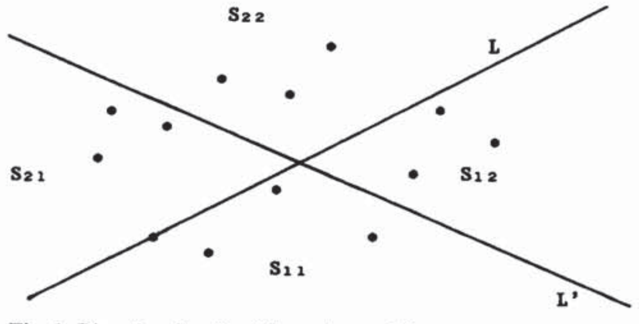

Fig. 1. Bisecting line L with conjugate L′.

Let S denote a set of n points in the Euclidean plane. For a directed line L we let left(L), on(L), and right(L) denote the number of points in S strictly to the left, on, and strictly to the right of L, respectively. L is said to *bisect* S if left(L) $\leq \frac{1}{2}$n and right(L) $\leq \frac{1}{2}$n.

L *induces* the partition of S into $S_{\text{left}}(L)$, $S_{\text{on}}(L)$, $S_{\text{right}}(L)$ with the obvious definitions. Note that $S_{\text{on}}(L)$ cannot be empty if n is odd; in this case, the bisecting line is unique if its slope (and direction) are given.

Let now L be a line which bisects S. We call another line L′ a *conjugate of* L (*for* S) if L′ bisects $S_{\text{left}}(L)$ and $S_{\text{right}}(L)$ (see Fig. 1). The important question of whether there is always a conjugate of a bisecting line is answered affirmatively, e.g., by Willard [12].

We now come back to the conjugation tree designed for the halfplanar range search problem.

Let S denote a set of n points in the Euclidean plane. Let L denote a line which bisects S and let L′ denote a conjugate of L. Unless S is empty, the *conjugation tree for* S (*and* L) is the tree

(i) with root w which stores L, |S|, and a linear array, on(w), which stores $S_{\text{on}}(L)$ in sorted order,

(ii) the conjugation tree with root left(w) for $S_{\text{left}}(L)$ and L′ as the left subtree of w, and

(iii) the conjugation tree with root right(w) for $S_{\text{right}}(L)$ and L′ as the right subtree of w.

Fig. 2 depicts the conjugation tree for a set of points as well as the dissection of the plane induced by the lines.

Obviously, the conjugation tree for a set of points is not unique since neither the first separating line nor the conjugate lines are unique (even if those inducing the same partition are considered to be the same).

We find it important to give here some meaningful comparisons between the conjugation tree and related data structures, the most famous of which is the quad tree. The most important properties of the conjugation tree, when compared with the quad tree as defined in [5], for point-sets are:

(i) the lines used by the conjugation tree to separate the point-set are not necessarily axis-parallel, and

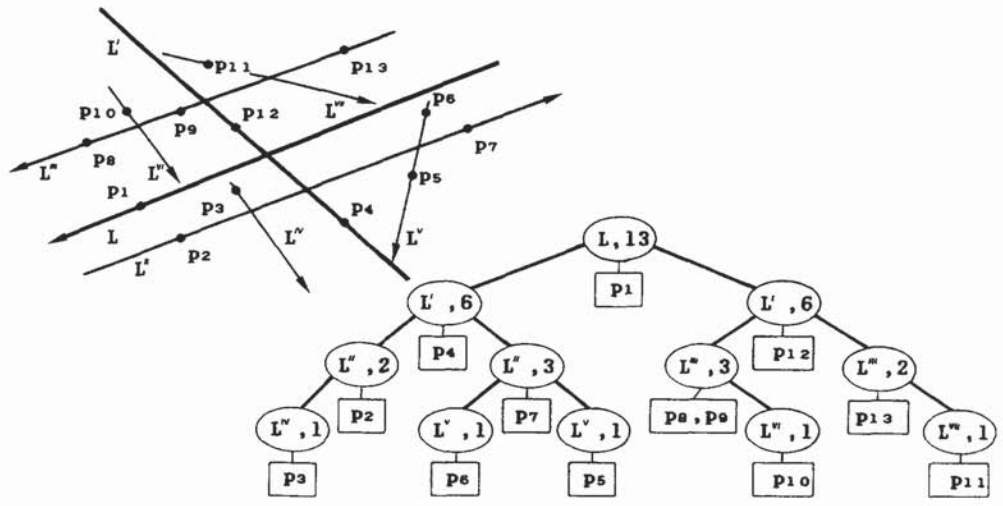

Fig. 2. A conjugation tree for thirteen points.

(ii) the partition of two point-sets just separated is not carried out independently.

Property (ii) makes the conjugation tree resemble the quad tree as it is used for storing digital pictures (see, e.g., [11]). The conjugation tree shares property (i) with the polygon tree of Willard [12]. This data structure stores n planar points in O(n) space and allows us to answer a halfplanar range query in $O(n^{0.77})$ time. In what follows, the conjugation tree is shown to be better than the polygon tree as far as efficiency is concerned. The gain in efficiency is essentially due to property (ii) which is not shared by the polygon tree.

Lemma 2.1. *Let* S *denote a set of* n *points in the Euclidean plane stored in a conjugation tree. Then* $O(n^{\log_2(1+\sqrt{5})-1})$ *time suffices to determine the number of points in* S *which are contained in a query halfplane.*

Proof. The assertion is verified by exhibiting and analyzing an algorithm which determines the number of points in S contained in a query halfplane h. There is no essential difference in the search strategy for h closed or open.

Let v be an arbitrary node of the conjugation tree T for S. We associate with v its *range* ran(v) such that the points in the subtree with root v are in ran(v). If v is the root of T, then ran(v) is the whole plane. Otherwise, let v be the left (right) son of its father f. Then ran(v) is the intersection of ran(f) with the open halfplane to the left (right) of the directed line stored in f. We let g denote the line which bounds h and are now ready to present the search algorithm which accumulates the result in a global variable COUNT.

Algorithm HALFPLANE QUERY. The search starts at the root w of T and with COUNT = 0. Note that the intersection of g and ran(w) is exactly g. Let v denote the current node in T and let int(v) be the intersection of g and ran(v). This line, ray, or line segment can be obtained by intersecting int(f), for v the left (right) son of f, with the open halfplane to the left (right) of the line stored in f. We distinguish two cases.

Case 1: int(v) is empty. Then ran(v) is either contained in h or has no intersection with h. In the former case, all points in the subtree of v are contained in h, thus, the number stored in v is added to a global variable COUNT. In the latter case, no action is taken.

Case 2: int(v) is not empty. We first increase COUNT by the number of points of on(v) that belong to h. Then the two sons of v are visited recursively (if they exist).

For the analysis of Algorithm HALFPLANE QUERY we need the following crucial observation: Let v denote an arbitrary node of T. Then the range of at least one of the grandsons of v is either totally contained in h or has no intersection with h. Thus, if Q(n) denotes the time required by the algorithm to search in a conjugation tree storing n points, then

$$Q(n) = Q(\tfrac{1}{2}n) + Q(\tfrac{1}{4}n) + O(\log n).$$

From the similarity of this recursive equation with the definition of the Fibonacci numbers we derive

$$Q(n) = O(n^{\log_2(1+\sqrt{5})-1}),$$

where $\log_2(1+\sqrt{5}) - 1$ is about 0.695. (See [7] for an analysis of the Fibonacci numbers and for a derivation of the following analytic description of the *kth* Fibonacci number.) Let $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for $k \geqslant 2$. Then

$$F_k = \left[\left(\tfrac{1}{2}(1+\sqrt{5})\right)^k - \left(\tfrac{1}{2}(1-\sqrt{5})\right)^k\right]/\sqrt{5}$$
$$= O\left(\left(\tfrac{1}{2}(1+\sqrt{5})\right)^k\right).$$

The result for Q(n) will now be verified. For the time being, we restrict our attention to numbers $n = 2^k$, k a nonnegative integer. We rewrite the relations on Q(n) as follows:

$$Q(2^k) \leqslant Q(2^{k-1}) + Q(2^{k-2}) + c_1 k \qquad (*)$$

and

$$Q(2^0) \leqslant c_2, \qquad (**)$$

and prove

$$Q(2^k) \leqslant (3c_1 + c_2)2^{k(\log_2(1+\sqrt{5})-1)} - c_1(k+3). \qquad (***)$$

Obviously, (* * *) holds for $k = 0$ because of (* *). Inequality (* * *) also holds for $k \geqslant 1$ by an inductive argument which uses (*) and the fact that

$$2^{(k-1)z} + 2^{(k-2)z} = 2^{kz},$$
$$\text{if } z = \log_2(1+\sqrt{5}) - 1.$$

Clearly, $Q(n) \leqslant Q(2^{\lceil \log_2 n \rceil})$, which completes the argument. □

Lemma 2.1 can be extended into various directions two of which are now addressed.

Let M be some mapping which associates each point p of the plane with a value M(p) in some commutative semigroup H. We assume that M(p) can be computed in constant time from p and that two numbers in H can be added in constant time. Then the conjugation tree allows us to compute in $O(n^{0.695})$ time the sum of M(p), for all p of a given set of n points which are contained in a query halfplane h. This result is obtained by assigning to each node v the sum of the values of all points stored in the subtree with root v. Notice that taking the set of positive integers together with the operation "+" for H and defining M(p) = 1 leads to computing the number as considered in Lemma 2.1. The conjugation tree also supports reporting the points in h. If t points of S belong to h, then $O(n^{0.695} + t)$ time is required for this activity. The bound is achieved via Algorithm HALFPLANE QUERY adapted to the case of reporting, that is, instead of adding some number to COUNT, all points stored in the respective subtree are reported.

More general than halfplanes, the conjugation tree permits arbitrary polygonal areas as query ranges. An analysis as presented in [12] shows that the time required for answering a query is still $O(n^{0.695})$ if the polygonal area is bounded by at most some constant number of edges, and if a search strategy analogous to the one of Algorithm HALFPLANE QUERY is applied.

For a complete analysis of the conjugation tree, we still have to analyze the space and time requirements for constructing it. While it is trivial that O(n) space suffices for the ultimate structure storing n points, an algorithm that constructs the tree in O(n log n) time is not obvious. Nevertheless, such an algorithm follows from a result of Megiddo [9]: For two sets of a total of n points in the Euclidean plane, a line that bisects both sets can be found in O(n) time if both sets can be separated by another line. This yields our main result.

Theorem 2.2. *Let* S *be a set of* n *points in the Euclidean plane. There exists a data structure which stores* S *in* O(n) *space and can be constructed in* O(n log n) *time, such that polygonal range queries can be answered in* $O(n^{\log_2(1+\sqrt{5})-1})$ *time.*

3. Discussion and extensions

Let us first give a review of the main contributions of this paper: A new data structure for point-sets in the plane is introduced. This data structure, called conjugation tree, is a binary tree which allows us to answer halfplanar range queries efficiently. More specifically, it stores a set S of n points in O(n) space and allows us to determine in $O(n^{0.695})$ time the number of points in S which are contained in a query halfplane (or polygon). The tree can be built in O(n log n) time. The conjugation tree improves the best O(n) space data structure for halfplanar range search known before, namely the polygon tree of Willard [12] which allows us to answer a halfplanar range query in $O(n^{0.77})$ time.

Unfortunately, the conjugation tree is inherently static, that is, it does not support insertions of new points or deletions of stored points. Therefore, we refer to [10] for general methods which dynamize static data structures. That is, these methods modify static data structures such that insertions and deletions can be carried out without prohibitive effort. Many of their dynamization methods apply to the conjugation tree.

Finally, we pose as an open problem either an improvement of the conjugation tree for halfplanar range queries, or a proof that, with O(n) space, no data structure can do better.

Acknowledgment

We thank W. Bucher for help in the analysis of the time complexity of the query algorithm.

References

[1] J.L. Bentley, Multidimensional binary search trees used for associative searching, Comm. ACM 18 (1975) 509–515.

[2] B. Chazelle, L. Guibas and D.T. Lee, The power of geometric duality, in: Proc. 24th Symp. on Foundations of Computer Science (1983) 217–225.

[3] H. Edelsbrunner, D.G. Kirkpatrick and H.A. Maurer, Polygonal intersection searching, Inform. Process. Lett. 14 (1982) 74–79.

[4] H. Edelsbrunner and E. Welzl, Constructing belts in two-dimensional arrangements with applications, SIAM J. Comput. 15 (1986) 271–284.

[5] R.E. Finkel and J.L. Bentley, Quad trees—a data structure for retrieval on composite keys, Acta Informatica 4 (1974) 1–9.

[6] M.L. Fredman, The inherent complexity of dynamic data structures which accommodate range queries, in: Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science (1980) 191–199.

[7] D.E. Knuth, Fundamental Algorithms: The Art of Computer Programming I (Addison-Wesley, Reading, MA, 1973) Chap. 1.

[8] D.E. Knuth, Sorting and Searching: The Art of Computer Programming III (Addison-Wesley, Reading, MA, 1973) Chap. 6.

[9] N. Megiddo, Partitioning with two lines in the plane, J. Algorithms, to appear.

[10] M.H. Overmars and J. van Leeuwen, Worst-case optimal insertion and deletion methods for decomposable searching problems, Inform. Process. Lett. 12 (1982) 168–173.

[11] H. Samet, Neighbor finding techniques for images represented by quadtrees, Comput. Graphics and Image Process. 18 (1982) 37–57.

[12] D.E. Willard, Polygon retrieval, SIAM J. Comput. 11 (1982) 149–165.