アルゴリズム 33−11
(1993. 5. 29)

Partial Construction of an Arrangement of Lines and Its Application to Optimal Partitioning of Bichromatic Point Set *

Tetsuo Asano
Osaka Electro-Communication University

Takeshi Tokuyama
IBM Research, IBM Tokyo Research Laboratory

SUMMARY This paper presents an efficient algorithm for constructing some portion of an arrangement of lines in the plane in time roughly proportional to the complexity of the partial arrangement, i.e., the number of intersections in the arrangement. Then, we apply the algorithm to a bipartitioning problem of a bichromatic point set: For m red points and n blue points in the plane and a directed line L, the figure of demerit $f_d(L)$ associated with L is defined to be the sum of the number of blue points below L and that of red ones above L. The problem we are going to consider is to find an optimal partitioning line to minimize the figure of demerit.

直線のアレンジメントの部分構成アルゴリズムと 2色点集合の最適分割問題への応用

浅野哲夫 (大阪電気通信大学工学部応用電子工学科)
徳山豪 (日本IBM東京基礎研究所)

あらまし 本論文では平面上の直線のアレンジメントの一部分をほぼその複雑度、すなわちアレンジメント中の交点の個数に比例する時間で構成する効率のよいアルゴリズムを提案する。さらに、このアルゴリズムを次のように定義される2色の点からなる集合の分割問題に適用する：平面上に m 個の赤点と n 個の青点が与えられたとき、有向直線 L の下にある青点の個数と L より上にある赤点の個数の和を直線 L のデメリット $f_d(L)$ ということにするとき、このデメリットを最小にする最適な分割線を求めよ。

*This work was partially supported by Grant in Aid for Scientific Research of the Ministry of Education, Science and Cultures of Japan.

1 Introduction

Duality transform is a kind of algorithmic techniques to solve geometric problems not in the original setting but in the dual setting. An example is a problem of finding a largest-area empty triangle for a set of points. Given a set of n points in the plane, we want to find a traingle containing no other point in it that has the largest area. A brute-force algorithm which examines every combination of three points obviously takes $O(n^4)$ times. The use of duality transform reduces the time to $O(n^2)$. That is, a point (a, b) is mapped into a line $y = ax + b$ and a line $y = -cx + d$ is mapped into a point (c, d). Since this mapping preserves relative vertical position between points and lines, the problem is reduced to evaluating triangles corresponding to three lines in the dual plane such that one line lies immediately above or below the intersection of the remaining two. A key to the solution is to construct an arrangement of lines, a kind of graph structure which represents every incidence relationship among lines and intersections. A simple incremental algorithm(3,6) which constructs the arrangement of n lines in $\Theta(n^2)$ time is known. This algorithm is optimal in its time complexity since an arrangement of n lines contains $O(n^2)$ intersections. A topologocal sweep algorithm(5) can evaluate all possible pairs of lines and intersections of lines also in $O(n^2)$ time but only using linear space.

A pair of duality transform and arrangement of lines is a powerful tool to solve a number of geometric problems, but one of its limits is that once we decide to use arrangement of lines the running time must be at least quadratic in the problem size. In some cases, however, only partial construction of the arrangement is sufficient instead of the whole construction. In such cases the complexity of the partial arrangement may be subquadratic. This paper presents an algorithm for constructing a partial arrangement in time roughly proportional to its complexity, that is, the number of intersections in the portion.

Then, we apply the algorithm to a bipartitioning problem of a bichromatic point set: For m red points and n blue points in the plane and a directed line l, the figure of demerit $f_d(l)$ associated with l is defined to be the sum of the number of blue points below l and that of red ones above l. Given a constant k, we want to find a line l which minimizes the figure of demerit $f_d(l)$ if the associated value of $f_d(l)$ is not greater than k. An algorithm which solves the above problem in $O(kn + \sqrt{k}n \log^2 n)$ time, which is subquadratic if k is sublinear.

2 Preliminaries

Let $\mathcal{L}$ be a set of n lines in the plane and $\mathcal{A}(\mathcal{L})$ be its arrangement. A k–level of $\mathcal{A}(\mathcal{L})$ consists of a sequence of line segments of $\mathcal{A}(\mathcal{L})$ such that at most $k - 1$ lines lie strictly below it and at most $n - k$ lines strictly above it. See Figure 1 for an example of a k–level in which the 2-level is shown by bold lines.

A k–belt of $\mathcal{A}(\mathcal{L})$ is defined to be the region bounded by the k–level and $(n - k)$–level of the arrangement $\mathcal{A}(\mathcal{L})$(7). It is known that the k–belt can be constructed in $O(n\sqrt{k} \log^2 n)$ time and $O(n\sqrt{k})$ space(7). A $[a, b]$–belt of $\mathcal{A}(\mathcal{L})$ is an extension of a k–belt. It is defined to be the portion of $\mathcal{A}(\mathcal{L})$ bounded by its a–level and b–level, $a < b$. Notice the difference between the two notions: The k–belt is just the region bounded by the two levels, and the $[a, b]$–belt is not a region but a portion of the arrangement $\mathcal{A}(\mathcal{L})$ bounded by its a–level and b–level. Several complexity results are known(4).

[Lemma 1] *For an arrangement of n lines in the plane, the complexity of its i–level is $O(\sqrt{i}n)$, and it is constructed in $O(\sqrt{i}n \log^2 n)$ time.*

[Lemma 2] *For an arrangement of n lines in the plane, the total complexity of i–levels for $i = 1, 2, \ldots, h$ is $O(hn)$.*

Now, the problem is whether the $[1, h]$–belt can be constructed in time roughly proportional to its

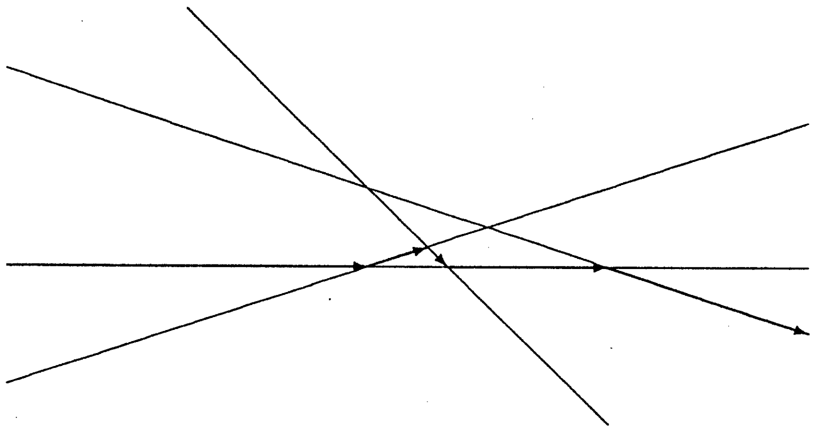

Figure 1: 2-level of an arrangement of 4 lines.

complexity, $O(hn)$. This paper presents such an algorithm based on Topological Walk Algorithm(2) proposed by the authors, which runs in $O(nh + \sqrt{hn}\log^2 n)$ time.

3 Algorithm for Constructing $[1, h]$−Belt

This section describes an algorithm for constructing a $[1, h]$−belet. Given a set of n lines in the plane, we first construct its h−level by using the ray-shooting technique(1). The resulting h−level is a sequence of line segments monotone in the x−direction. Let it be $(e_0, e_1, \cdots, e_{m-1})$. We direct those line segments from left to right (we assume as usual that there is no vertical line). Then, we add an additional vertical line at $x = -\infty$. Obviously the line intersects e_{m-1}. Removing the portion of the vertical line above the intersection with e_{m-1} and the portion of e_{m-1} to the right of the intersection results in another sequence of line segments which ends in the downward vertical half line. Let $(e_0, e_1, \cdots, e_{m-1}, e_m)$ be the resulting sequence.

Starting with e_m, we traverse the resulting sequence to the left in the reverse order $e_m, e_{m-1}, \cdots, e_1, e_0$, At each corner between e_i and e_{i+1} during the traverse, if $e_i \rightarrow e_{i+1}$ is a right turn, we do nothing. Otherwise, we emanate a ray $r(e_i)$ as an extension of e_i to the right starting at a point just beyond the intersection (e_i, e_{i+1}) until it encounters some line segment (it certainly encounters some line segment). Then, we add the ray $r(e_i)$ to the sequence. This operation results in a directed tree T rooted at negative infinity on the half line e_m (see Figure 2). In the above operation, if a ray (r_i) first hits another ray $r(e_j)$ which has been added to T so far, the starting point of e_i lies to the left that of e_j due to the monotonicity of the sequence $(e_0, e_1, \cdots)$. By the assumption the ray $r(e_i)$ does not hit any e_k, $k > i$ before hitting $r(e_j)$. Therefore, the slope of $r(e_i)$ is larger than that of $r(e_j)$. This means that the resulting tree is equivalent to a lower horizon tree(5) used in the topological sweep.

It is easily seen that the lower horizon tree T partitions the portion of the arrangement below the h−level into convex regions. Furthermore, as is verified in (2), the depth-first search of the lower horizon tree gives a sequence of regions to visit, which has the following properties.

(1) Every region is visited.

(2) The length of the sequence is at most twice larger than the number of regions.

(3) For each edge (directed from left to right) of the lower horizon tree, the region lying to its right is visited before the one lying to its left (see Figure 3).

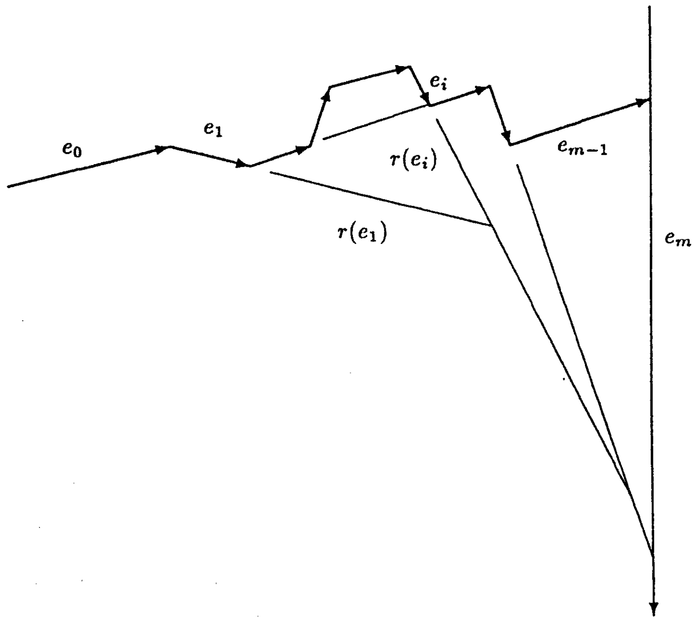

Figure 2: Definition of rays and lower horizon tree T.

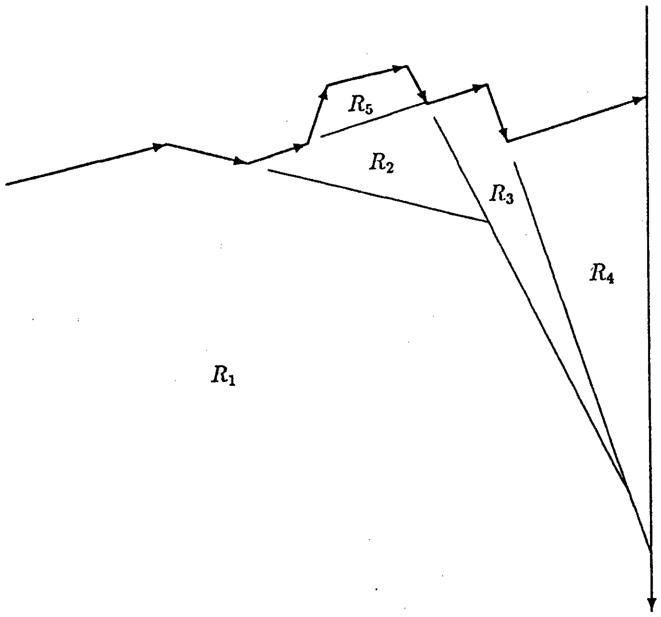

Figure 3: Partition of the arrangement by the lower horizon tree and the order of visiting regions.

This implies an ordering of the regions to be processed. Actually we construct an arrangement of lines according to the ordering, using Topological Walk.

Topological Walk works as follows: Let P be a convex polygon and $\mathcal{L}$ be a set of lines in the plane. For each line l of $\mathcal{L}$, we compute its left and right intersections $p_L(l)$ and $p_R(l)$, respectively, with P. The boundary of the convex polygon P can be decomposed into upper and lower chains. Starting with the rightmost vertex of the lower chain, we traverse the boundary of P in the clockwise manner. Each time we encounter a left endpoint $p_L(l)$, we emanate the ray from $p_L(l)$ along the line l toward the interior of P until it hits its boundary or the rays added so far. Once we encounter a right endpoint of some line, we let the lines whose left endpoints appear thereafter wait at the boundary. The above operations result in a tree, which coincides with the upper horizon tree used in Topological Sweep. Then, we implement the depth-first search on the tree while updating the tree at twigs. When the portion of the boundary at which some line is waiting is traversed in the depth-first search, the waiting edge is incorporated in the upper horizon tree.

In our case we visit convex regions defined by the lower horizon tree associated with the h-level in the order induced by depth-first search on the lower horizon tree. Since each such region is convex, we can apply Topological Walk to enumerate all intersections in the region.

As a preprocessing, after constructing the h-level in $O(\sqrt{h}n \log^2 n)$ time, we sort all the endpoints on the h-level in $O(\sqrt{h}n \log n)$ additional time.

To implement Topological Walk in a region R, we need to sort the left intersections of lines with the bounded boundary of R. The key observation here is that the left endpoint of a line with R is located on the lower convex chain of R if R is a region in the decomposition induced by the lower horizon tree. If there is a left endpoint of a line on the upper chain of R, the slope of the line is smaller than the slope of the intersecting edge of R, thus, R should be cut by the line in the lower horizon tree decomposition. In fact, there is no "waiting edge" mentioned above.

Thus, we only need to enumerate and sort the endpoints on the lower convex chain. The endpoints on the lower convex chain is given from the topological walk in the previous stages (we only need linear time merge operation to obtain the sorted list).

Summarizing the above discussion, we have the following algorithm.

[Constructing $[1, h]$-Belt]

(Step 1) Construct the h-level using a ray-shooting algorithm.

(Step 2) Build the lower horizon tree T associated with the h-level.

(Step 3) Decompose the region below the h-level into small convex regions by the lower horizon tree T.

(Step 4) Sort all the endpoints on the h-level.

(Step 5) Perform a depth-first search on T to determine the order to visit those convex regions.

(Step 6) According to the order we construct an arrangement of each convex region based on Topological Walk Algorithm.

[Theorem] *Given a set of n lines in the plane and an integer $h \leq n$, the $[1, h]$-belt of the arrangement of the lines can be constructed in $O(\sqrt{h}n \log^2 n)$ time.*

Finally it should be noted that although the above algorithm describes only how to enumerate all the intersections it is not so hard to modify it so as to build graph structure to represent the arrangement. Once the graph structure is obtained, we can label each edge by its associated level in linear time. This leads to the decomposition of the arrangement into levels.

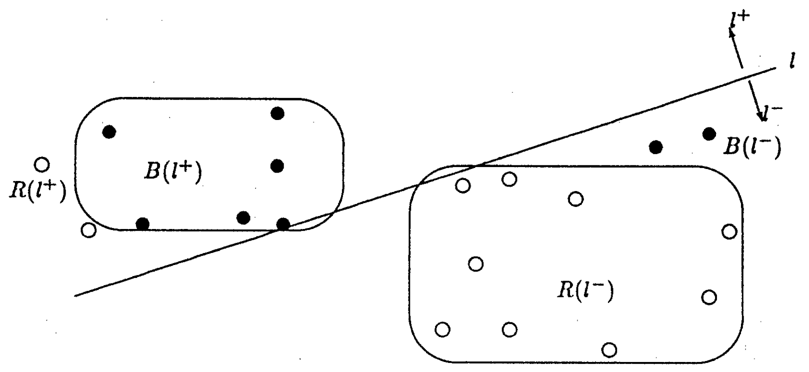

Figure 4: Partitioning bichromatic point sets. Filled and empty circles represent blue and red points, respectively. $r(l^+) = |R(l^+)| = 2, b(l^-) = |B(l^-)| = 2$, and we have $f_d(l) = r(l^+) + b(l^-) = 4$.

4 Finding Optimal Partitioning of Bichromatic Point Set

Let R and B be point set of in a plane. We call a point in R a *red point* and that of B a *blue point*. The number of red points is r, and that of blue points is b. Let $n = r + b$.

For a directed line l in the plane, l^+ (resp. l^-) denotes the half plane above (resp. below) of l. Let $R(l^+)$ be the set of red points in l^+, and $r(l^+)$ be the cardinarity of it. Similarly, we define $B(l^-)$ and $b(l^-)$. An example is shown in Figure 4.

The problems we focus on is the following:

P(1): Optimal separating line finding Find a line l which minimizes $f_d(l) = r(l^+) + b(l^-)$.

It is well known that we can determine in $O(n \log n)$ time whether there is a line competely separating point sets. All we need to do is to compute convex hulls of the point sets and to check whether they intersect each other. In that case $f_d(l)$, the figure of demerit of the separating line l, is zero. Here we are interested in the case in which there is no such separating line, that is, the two convex hulls have non-empty intersection.

In many practical cases the figure of demerit is expected to be small compared with the total number of points. So, the following problem is also meaningful.

P(2): Optimal separating line finding for constraint case Given a constant k, find a line l which minimizes $f_d(l)$ if the associated value of $f_d(l)$ is not greater than k.

We show below an algorithm for solving P(2) in $O(kn + \sqrt{k}n \log^2 n)$ time, which is subquadratic if k is sublinear.

We dualize the problem, and consider the arrangement of lines. We assume we have computed the red i-level for $i = 1, 2, ..., k$ and blue j-level for $j = b - k, b - k + 1, ..., b$. We denote $red(i)$ for the red i-level, and $blue(j)$ for the blue j-level.

The algorithm consists of two steps. First one is the initial setting, and the second one is the main routine.

At the initial step, we find the highest (i.e. the smallest indexed) red level intersecting the $blue(b-k)$. If $red(1)$ is below $blue(b-k)$, we set the level to $red(1)$. If $red(k)$ is above $blue(b-k)$, then we return *false* and exit the algorithm. This level is found in $O(nk)$ time. Without loss of generality, we can assume that the red level is $red(1)$.

Now we start the main routine.

Algorithm:

```
1. $p = k + 1$.
2. For $i = 1$ to $k$;
3.     Find the lowest blue level $blue(b - j)$ intersecting $red(i)$ such that $i + j < p$
4.     If there is such $j$ as above, set $p = i + j$
5. endfor
```

The number p above is called the target number. For finding the lowest blue level $blue(b - j)$, we apply the plane sweep method. Notice that, as the output of topological walk, the vertices on each level are sorted with respect to the x coordinate value.

We first locate the left endpoint (at $x = -\infty$) of the red level $red(i)$ in the blue arrangement. Suppose it is just between $blue(b - s)$ and $blue(b - s - 1)$. If $s < p - i - 1$, we set $t = s$, otherwise we set $t = p - i - 1$. We sweep on $red(i)$ and $blue(b - t)$ from the left until we find an intersection between them. Once we find it, we update t to $t - 1$. Then, while keeping the current position in the red level, we sweep the new $blue(b - t)$ from its left end until it goes beyond the current position. Note that the new $blue(b - t)$ never intersects $red(i)$ before the current position since it lies above the old $blue(b - t)$ to the left of the current position. Then, we perform the plane sweep again for $red(i)$ and $blue(b - t)$. We report t as the sweep end.

It is clear that the algoirthm correctly finds an optimal partitioning line. The sweep operation concerning $red(i)$ needs the time proportional to the complexity of $red(i)$ and those of $blue(b - t)$ for $t = p - i - 1, p - i - 2 ..., t' = p' - i$, where t' is the output of the subroutine, and p' is the target number for the next step (where i is replaced by $i + 1$.). Notice that a blue level is used only once in the main routine.

Thus, the total complexity of the algorithm is $O(kn)$.

5 Conclusions

In this paper we have presented an algorithm for constructiing the $[1, h]-$belt of an arrangement of lines in the plane in time roughly proportional to the complexity of the belt. Then, we have applied the algorithm to solve a bipartitioning problem of a bichromatic point set. Given n points in total, the algorithm obtained can answer the question whether there is a line separating the bichromatic point set into two so that the number of points lying in the wrong sides is at most k in time $O(nk + n\sqrt{k}\log^2 n)$ time and if the answer is "yes" then it outputs an optimal separating line minimizing the number of points lying in teh wrong sides in the same time complexity. If k is sublinear then this algorithm finds an optimal separating line in subquadratic time.

References

(1) P. K. Agarwal, "Ray Shooting and Other Applications of Spanning Trees with Low Stabbing Number", Proc. 5th ACM Symp. on COmputational Geometry (1989) 315-325.

(2) Te. Asano, L. J. Guibas, and T. Tokuyama, "Walking on an Arrangement Topologically", Proc. 7th ACM Symp. on Computational Geometry (1991) 297-306.

(3) B. M. Chazelle, L. J. Guibas, and D. T. Lee, "The Power of Geometric Duality", *BIT* 25 (1985) 76-90.

(4) H. Edelsbrunner, "Algorithms in Combinatorial Geometry", *Springer-Verlag,* (1986).

(5) H. Edelsbrunner and L.J. Guibas, "Topologically Sweeping an Arrangement", *Proc. 18th Annual ACM Symp. on Theory of Computing,* (1986) 389-403.

(6) H. Edelsbrunner, J. O'Rourke, and R. Seidel, "Constructing Arrangements of Lines and Hyperplanes with Applications", *SIAM J. on Comput.,* 15 (1986) 341-363.

(7) H. Edelsbrunner and E. Welzl, "Constructing Belts in Two-Dimensional Arrangements with Applications", *SIAM J. on Comput.,* 15 (1986) 271-284.

(8) E. Welzl, "More on k-sets of finite sets in the plane", *Discrete and Comput. Geometry* 1 (1986) 95-100.