アルゴリズム 17－3
(1990. 9.28)

トポロジカル ウォーク

浅野 哲夫[1] Leonidas J. Guibas[2] 徳山 豪[3]

[1] 大阪電気通信大学工学部 [2] MIT [3] IBM東京基礎研究所

直線の族が作る平面分割はアレンジメントと呼ばれ、計算幾何学では重要な対象である。本論文では、アレンジメント上のウォークを構成するオン・ライン算法(トポロジカル・ウォーク)を与える。ウォークとは、アレンジメントの双対グラフ上のグラフ・ウォークの事である。アレンジメント上の走査を行なう算法として、トポロジカル・ウォークは知られているどの算法よりも優れた理論的効率を持つ。更に、アレンジメント上の多くの最適化問題を解くパラダイムとして、トポロジカルウォークは利用される。

Walking on an arrangement topologically

Tetsuo Asano[1], Leonidas J. Guibas[2], and Takeshi Tokuyama[3]

[1]Department of Engineering, Osaka Electro-Communication University.
18-8 Hatsu-machi, Neyagawa, Osaka, 572 Japan.

[2]Laboratory for Computer Science, MIT.
545 Technology Square,Cambridge, MASS. 02139, USA.

[3]IBM Research, Tokyo Research Laboratory.
5-11 Sanban-cho, Chiyoda-ku, 102 Japan.

Abstract

We present an algorithm named *topological walk*, which is an on-line algorithm to give a walk of an arrangement. Here, a walk is a list of cells of the arrangement in which consecutive cells are adjacent to each other. The algorithm is input-sensitive; in precise, given an arrangement of n lines in a convex region which contains K cells, a walk is given in $O(K + n \log n)$ time and linear working space. Hence, as a sweep method on arrangements, topological walk assures the best theoretical performance. Further, we can efficiently solve several optimal-cell finding problems applying topological walk.

1. Introduction

Arrangements of lines are frequently used in algorithms of computational geometry [E]. A drawback of the arrangement is its high space complexity. We need $O(n^2)$ space to store an arrangement generated from n lines, that is too expensive in many applications. Therefore, instead of constructing an arrangement in advance, it is often advantageous to sweep an arrangement updating more compact data. The *topological sweep* developed by Edelsbrunner and Guibas [EG1, EG2] is an efficient method for this purpose. We consider an improved topological sweep algorithm named *topological walk*.

Let $H = \{l_1, l_2, ..., l_n\}$ be a set of lines in a plane. Associated arrangement is denoted by $A(H)$. A connected open region of the arrangement is called a *cell*. We say that two cells are *adjacent* if their closures share an edge. Motivations of the research are following:

Motivation 1: Let us consider a problem to find an optimal cell of $A(H)$ with respect to a given criterion (depending on information associated with cells). Suppose the information in a cell is given efficiently from that of its adjacent cell. If $A(H)$ has been constructed, we can find the optimal cell by searching on the dual graph of $A(H)$. This method needs $O(n^2U)$ time and $O(n^2 + I)$ space, where U is the time to update the information, and I is the space to store the information of single cell. If we apply topological sweep (or usual plane sweep), we should keep $O(n)$ cells in each step of the algorithm. Thus, space complexity is reduced to $O(nI)$. However, $O(nI)$ space is still expensive, since I is often considerably large. We would like to reduce the space complexity to $O(n + I)$.

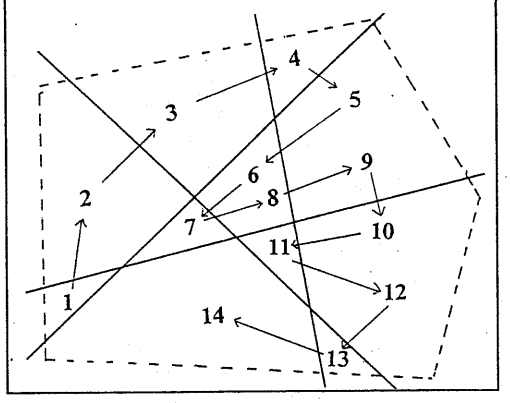
Motivation 2: Instead of searching over entire arrangement, it is often advantageous to search on a small part of an arrangement. Let us consider a convex subdivision (of a region) consisting of K cells in an arrangement. Then, usual topological sweep needs $O(n^2)$ time to search on it, while plane sweep method solves the problem in $O(K \log n)$ time (roughly speaking). We would like to design an algorithm which sweeps the subdivision in $O(K + n \log n)$ time. This motivation resembles to that of the optimal intersection reporting problem of set of line segments [CE].

Given a convex polygonal region R, the lines of H cut R into convex sub-regions (cells), and the corresponding subdivision is denoted by $A(H;R)$.

Definition 1. (Figure 1.) A sequence $W = (r_1, r_2, .., r_m)$ of cells in $A(H;R)$ is called a *walk* if r_{i+1} is adjacent to r_i for each $1 \leq i \leq m - 1$. A *complete walk* is a walk containing all cells in $A(H;R)$.

Our definition of the walk coincides with the graph theoretic *walk* [H] on the planar dual graph of $A(H;R)$. Topological

Figure 1. Walk of an arrangement



walk finds a complete walk of $A(H;R)$ in an on-line fashion. In each step, it keeps only a cell (with additional $O(n)$ information). Thus, it satisfies Motivation 1.

The algorithm is based on the topological sweep together with the depth first search of the *upper horizon tree,* and is sensitive to the size of the subdivision. More precisely, if K is the number of cells in $A(H;R)$ and s is the number of corners of R, the algorithm sweeps the arrangement in $O(K + n\log(n+s))$ time and $O(n+s)$ space. Thus, Motivation 2 is satisfied for the subdivision of a convex region R by an arrangement.

2. Cut and Upper Horizon Tree

The region R is bounded by a convex polygon $B(R)$ with s vertices. Let z be the leftmost vertex of $B(R)$. Cutting $B(R)$ at z, we obtain a clockwise oriented chain from z_0 to z_1, where z_0 and z_1 are copies of z. This chain is called the *stem.* We can assume each line of H intersects with the stem twice without loss of generality.

For a pair of edges e and f of an arrangement, we call e dominates f if there is a cell r incident to both e and f such that e is above r and f is below r.

Definition 2 (Figure 2). A *cut* of $A(H;R)$ is a list of (open) edges $C = \{e_1,..., e_m\}$ of $A(H; R)$ satisfying one of the following conditions for each $i = 1,2,..,m-1$:

(i): e_{i+1} dominates e_i.

(ii): The left end point of e_{i+1} is on the stem, and there exist an edge f_{i+1} of $A(H)$ to the left of e_{i+1} on the same line such that f_{i+1} dominates e_i in $A(H)$.

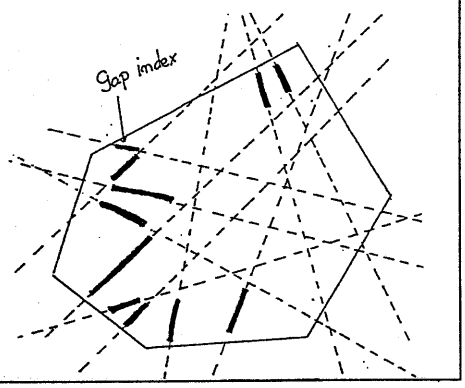
(iii): The left end point of e_i is on the stem, and there exist an edge f_i of $A(H)$ to the left of e_i on the same line such that e_{i+1} dominates f_i in $A(H)$.

We say the index i is a *gap index* if above condition (ii) holds.

We relabel the lines with respect to the ordinates of their left intersection with the stem in ascending order. Consider the leftmost edge $e_i(0)$ on l_i in $A(H,R)$. Then, the list $C_0 = (e_1(0), e_2(0), .., e_n(0))$ is a cut, which we call the *initial cut* of $A(H, R)$.

Definition 3 (Figure 3). We define the *upper horizon tree* $T(C)$ for a cut C, which is basically same as that of [EG1]. Let k be the first gap index of C. Then, we start with the cut edges (starter edges) $e_1, e_2,.., e_k$ and extend the edges to the right. When two

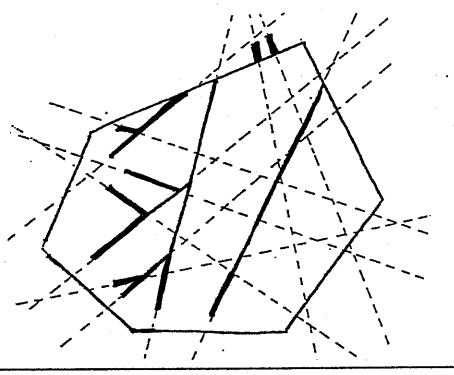
Figure 2. Cut and gap index



(extended) edges come together at an intersection point, only the one of higher slope continues on to the right; the other one stops at the point. If an edge intersects the stem, the edge is joined to the stem. How should we treat the rest of cut edges $e_{k+1}, e_{k+2},..., e_m$? We correspond e_j a *dummy edge* $\tilde{e}_j$ for each $j = k+1, k+2,.., m$, which branches from the stem at the left intersection of corresponding line outside the region R. Thus, we obtain a binary tree $T(C)$ rooted by z_0.

We call an edge of the arrangement lying on $T(C)$ an *edge piece* of $T(C)$, or an *edge* in short. The path on the stem between two consecutive intersections with the arrangement is also called an edge piece although it may contain corners of the boundary polygon. In order to distinguish from the edge piece, we call an *arc* for the segment between two nodes of $T(C)$, which coincides with a graph theoretical edge of the tree. Obviously, an arc of $T(C)$ is a

Figure 3. Upper horizon tree



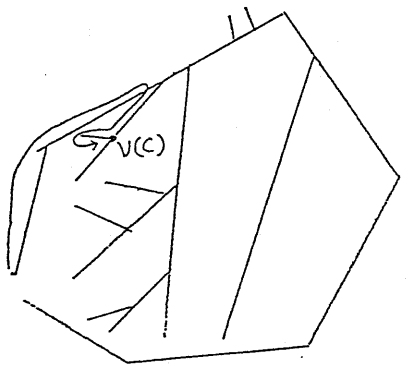
union of edge pieces (on a same line or on the stem).

An arc of $T(C)$ is called a leaf arc if it contains a cut edge of C or the root z_0; in the other words, one of its endpoints is a leaf node or the root of the tree. A branching node v of the tree is called a *leaf branch* if two leaf arcs meet there. A leaf branch is called a *twig* if two cut edges meet there, or a cut edge and the stem meet there. Notice that a leaf branch need not be a twig. It is obvious that there is at least a leaf branch if there is at least a cut edge.

We consider the depth first search on $T(C)$ starting from the root z_0, where we search the right branch first at each branching point. We denote $v(C)$ for the first encountered leaf branch during the depth first search (Figure 4).

The following is the key lemma (Overmars and Welzl [OW] essentially gave it in the dual version):

Figure 4. Twig $v(C)$



Lemma 1. *$v(C)$ is a twig.*

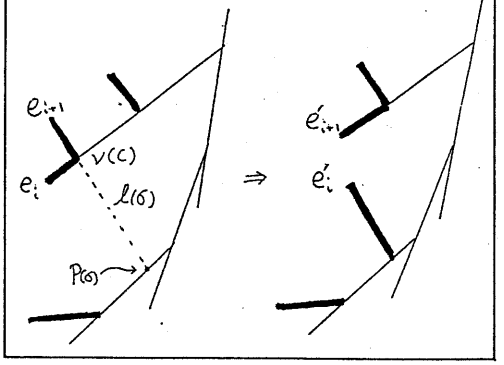
We update the cut by the following operation called *an elementary step* at a twig updating the upper horizon tree.

Definition 4 (Figure 5). If cut edge e_i and e_{i+1} meets at the twig $v(C)$, we denote e'_i (resp. e'_{i+1}) for the edge on the same line as e_{i+1} (resp. e_i) whose left end point is $v(C)$. The *elementary step* σ replaces e_i and e_{i+1} of C by e'_i and e'_{i+1} respectively to create another cut C'.

We show how to update the upper horizon tree after σ is done. We extend the line $l(\sigma)$ containing both e_{i+1} and e'_i to the right of $v(C)$ to find the intersection $p(\sigma)$ with $T(C)$. Then, connecting e'_i to the tree at $p(\sigma)$, we obtain $T(C')$.

We must consider the case $v(C)$ is the twig where a cut edge e meet the stem at its right endpoint. Here, the elementary step simply erase e from the cut. However, if e is erased, a gap index of the cut may be eliminated, and new starter edges may be added

Figure 5. Elementary step



to the upper horizon tree. Suppose the edges $e_{k+1},\ldots, e_h$ are added as new starter edges in C'. In such case, searching on $T(C)$ from $v(C)$, we find the branching point of the dummy edge $\tilde{e}_{k+1}$ before finding a twig. We execute an operation similar to an elementary step there, that is, we find the intersection of the line containing e_{k+1} with the upper horizon tree, and join e_{k+1} to the tree at that point. Joining edges $e_{k+1},\ldots, e_h$ by a series of the operations, we obtain $T(C')$.

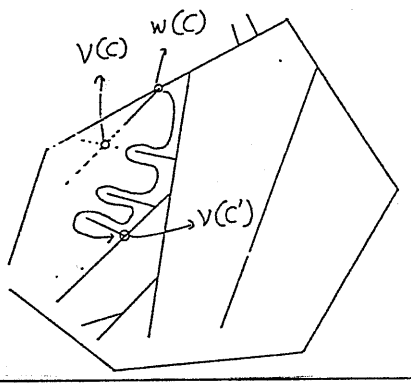
3. Algorithm to sweep $A(H;R)$

3.1. Procedure Sweep

1. $C = C_0$ is the initial cut;
2. Construct $T(C)$;
   while C is nonempty, ***do***;
3.1. find $v(C)$;
3.2. execute the elementary step at $v(C)$;
3.3. update C and $T(C)$;
   end_do;

This algorithm correctly sweeps the arrangement. The difference from the topological sweep algorithm of [EG1] is

Figure 6. Search for the next twig



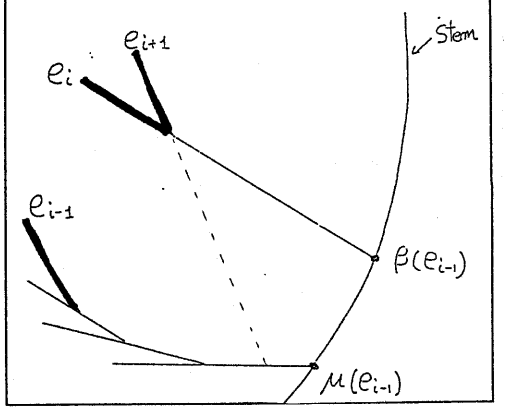
that we specify the place where next elementary step will be done. We remark that the data structure is much simpler in our version compared with that of [EG1], which uses both upper horizon tree and *lower horizon tree* in order to find a candidate of next elementary step.

Let us give an efficient implementation of the algorithm SWEEP. We consider an efficient way to find the next twig $v(C')$ after the cut is updated from C into C'. If $v(C)$ is not on the stem, we go back with an arc on the tree to the root to find a node $w(C)$. If $v(C)$ is on the stem, we define $w(C) = v(C)$. The following is a key lemma (Figure 6).

Lemma 2. *$w(C)$ is equal or precedent to $v(C')$ in the depth first search of $T(C')$.*

From Lemma 2, we can find $v(C')$ during the depth first search starting from $w(C)$.

Figure 7. $\beta(e_{i-1})$ and $\mu(e_{i-1})$



3.2. Analysis of the algorithm

The initialization needs $O(n \log(n+s))$ time including the sorting of the intersecting points of the lines of H with the stem.

We visit $O(K+n)$ nodes during the depth first search because of Lemma 2. Thus, the cost of the SWEEP is $O(K + n \log(n+s))$ except for the cost to update the upper horizon tree.

We prepare some notations to analyze the cost to update the upper horizon tree. For an edge e in $T(C)$, the path from e to the root is denoted by $Path(e)$. Let $C = (e_1,.., e_m)$ be the cut. For each e_i $(i = 1,2,..,m-1)$, the intersection of $Path(e_i)$ and $Path(e_{i+1})$ is denoted by $\beta(e_i)$ (see Figure 7). The following lemma is obvious:

Lemma 3. *Suppose an elementary transformation σ is done at a twig where cut edges e_i and e_{i+1} meet. Then the intersection point $p(\sigma)$ lies on the path from e_{i-1} to $\beta(e_{i-1})$.*

From Lemma 3, the following two different methods to find $p(\sigma)$ are applicable:

Forward search: We search for $p(\sigma)$ from e_{i-1} on the path $Path(e_{i-1})$.

Backward search: We search for $p(\sigma)$ from $\beta(e_{i-1})$ on the path $Path(e_{i-1})$ backward.

Unfortunately, there is an example where $K = O(n\sqrt{n})$, $s = O(\sqrt{n})$, and total updating cost is $\Omega(n^{7/4})$ time if we adopt the forward search, and there is another example which needs $\Omega(n\sqrt{K})$ time if we adopt the backward search.

In order to obtain an algorithm sensitive to K, we consider the following improved method. We define $\mu(e_i)$ by the branching point of $Path(e_i)$ from the stem if $\beta(e_i)$ is on the stem. Otherwise, we define $\mu(e_i)$ by $\beta(e_i)$ (Figure 7).

Obviously, $p(\sigma)$ is located either between e_{i-1} and $\mu(e_{i-1})$, or on the stem between $\mu(e_{i-1})$ and $\beta(e_{i-1})$. Note that we can maintain $\beta(e_i)$ and $\mu(e_i)$ for each e_i in $O(1)$ additional time and space.

Mixed search: We search for $p(\sigma)$ from $\mu(e_{i-1})$ on $Path(e_{i-1})$ backward. Moreover, if $\mu(e_{i-1})$ is on the stem, we search for $p(\sigma)$ on the stem from both $\mu(e_{i-1})$ and $\beta(e_{i-1})$ in parallel.

Theorem 1. *We can sweep the arrangement $A(H, R)$ in $O(K + n\log(n+s))$ time and $O(n+s)$ space, where s is the number of corners on R and K is the number of cells of the arrangement in the interior of R.*

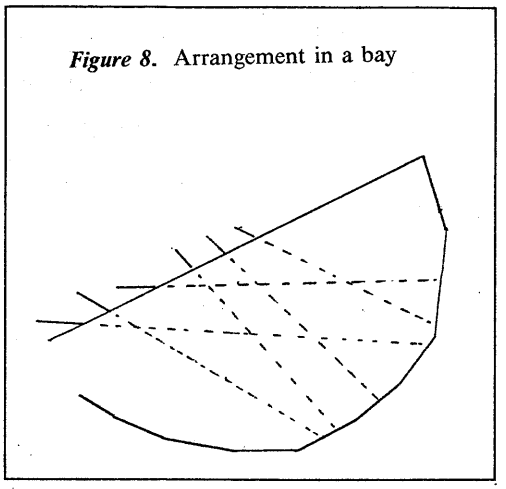


Figure 8. Arrangement in a bay

We devote the rest of this section to prove Theorem 1. First, we consider the case where R is a bay, that is, R is consisting of a segment L and a convex chain Γ below it. Moreover, each line of the arrangement has its left endpoint on L, and we assume these endpoints are sorted (Figure 8).

Suppose the length of the chain Γ is h. We consider L as a member of the arrangement, and start the sweep at the twig on L.

Lemma 4. *If R is a bay, we can sweep R in $O(K + n + h\log n)$ time.*

Proof of Lemma 4. We adopt the mixed search method. We begin with the following simple observation:

Claim 4.1 *Let l be a line containing an edge e on the path from $\mu(e_{i-1})$ to $p(\sigma)$. Then, l intersects $l(\sigma)$ in the interior of R.*

Similarly to the analysis in [EG2], we charge the cost to traverse e to the intersection of $l(\sigma)$ and l except for the rightmost edge on the path. Then, it is seen that each

intersection is charged at most once. Since these intersection are located in R because of Claim 4.1, the cost to search backward in the interior of the bay is $O(K+n)$. We must estimate the cost for searching on the boundary chain Γ (that is, between $\mu(e_{i-1})$ and $\beta(e_{i-1})$). We consider the following game:

Game: Let $I=[0,h]$ is an interval. We insert n dividing points into I in an on-line fashion and decompose it into $n+1$ subintervals. Suppose a value z is inserted into the subinterval $I_j=[x_j, x_{j+1}]$. Then, we divide it into $[x_j, z]$ and $[z, x_{j+1}]$. We pay $\min\{z-x_j, x_{j+1}-z\}$ units for this insertion.

Claim 4.2. *It costs $O(h\log n)$ units for the worst case to complete above game.*

Since we adopt the mixed search method, it follows from the claim that the cost to find the intersections on Γ is $O(h\log n)$ in total. Hence, we have Lemma 4.

Now, we consider the general case where R is a convex region. We ignore the dummy edges for a while. There are $N=O(n)$ bays in the initial upper horizon tree. These bays are called *gulfs*. We can observe that topological walk is processed as a sequence of gulf sweeps. A new gulf sweep starts if the topological walk visit the left end edge (that is a cut edge) of the lower chain of the current gulf. Let n_j , k_j, h_j be the number of inserted lines, intersections, and length of chain of the j-th gulf in the process.

$\sum_{j=1}^{N}(k_j+n_j+h_j\log n_j)$. is the total updating cost. Obviously, $\sum_{i=1}^{N}h_i=O(n+s)$, $\sum_{i=1}^{N}k_i=O(K)$, and $\sum_{i=1}^{N}n_i=O(K+n)$. Hence, $O(K+(n+s)\log n)$ time is needed in total.

However, if $s\geq n$, we can reform the region into one with at most $2n$ corners in $O(n\log(n+s))$ time preprocessing without deleting any cell. Hence, the total cost of the SWEEP is reduced to $O(K+n\log(n+s))$.

We must take care of dummy edges in a gulf. Suppose $\tilde{e}$ is a dummy edge corresponding to a cut edge e on a line l. Because of the property of the algorithm, each line which intersects with l at a point above e out of R has been eliminated when e joins the upper horizon tree. Thus, although the line l starts from the chain of the bay, claim 4.1 holds. Hence, the analysis shown in Lemma 4 is valid, and we obtain the Theorem 1.

4. Walk on $A(H; R)$

We consider the walk on the arrangement from the left region of $v(C)$ to that of $v(C')$. To make statements simple, we add all the intersections of the lines with the stem as the nodes of the upper horizon tree, although some of them are of degree 2. Then, the following two lemmas are easily seen:

Lemma 5. *Each arc of $T(C')$ encountered searching for $v(C')$ consists of single edge piece.*

Lemma 6. *During the depth first search, the region on the right (with respect to the direction from the root to leaves) of the current searched edge piece is changed if and only if we encounter a leaf node or a degree 2 node.*

We roughly sketch the algorithm to walk on the arrangement.

Procedure WALK ;
1. Execute the procedure SWEEP;
2. During the depth first search, we trace the right region of the current searched edge;

Note that we don't store the walk but only trace it. During the depth first search, a vertex is visited at most three times. More precisely, we can prove the following:

Theorem 2. *The above procedure generates a complete walk of length at most* $2K+n$ *in* $O(K+n\log(n+s))$ *time and* $O(n+s)$ *working space.*

5. Applications

Since the topological walk can sweep a part of an arrangement input sensitively, it is an efficient paradigm for almost all the problems solved by sweep methods.

A catalog of applications of topological sweep is given in [EG1]. Among them, the minimum area triangle search can not be solved by topological walk, since it needs the lower horizon tree in an essential way. However, the other applications -- computing a longest monotone path, computing a longest monotone concave path, computing a largest convex subset, computing a largest empty convex subset, finding a maximal stabbing line, and visibility problems -- can be solved by using topological walk.

5.1. Finding an optimal cell

We consider a cost function $f=f(r)$ associated with the arrangement, and search for the cell with the optimal cost. We consider a data structure $Data(r)$ from which we calculate $f(r)$. It needs T time to construct $Data(r)$, and I space to store it. Moreover, we can dynamically obtain $Data(r)$ and compute $f(r)$ in U time, if we know both $Data(r')$ and $f(r')$ for an adjacent cell r'. The updating time U is small compared with T. Typically, imagine the case $T=O(n)$, $I=O(n^{\alpha})$ $(0<\alpha\leq 1)$, and $U=O(1)$.

If we compute $f(r)$ along a walk of length M updating the dynamic data, we can obtain the cell with an optimal cost in $O(MU+T)$ time. Applying topological walk, we obtain the following theorem:

Theorem 3. *We can find the optimal value of the cost function and its associated cell in* $O(n\log(n+s)+KU+T)$ *time and* $O(n+s+I)$ *space.*

Let us compare the topological walk with other paradigms. If we construct the arrangement and walk on it in an usual fashion, it needs $O(n\log n+KU+T)$ time and $O(K+I)$ space. The plane sweep method needs $O(n\log n+K\{\log n+U\}+T)$ time and $O(nI)$ space. The usual topological

sweep needs $O(n^2 + KU + T)$ time and $O(nl)$ space, or $O(n^2 + KT)$ time and $O(n + l)$ space. Thus, the topological walk is always advantageous to each of these paradigms.

We will give a typical example of the optimal cell finding.

5.2. Linear approximation of multiple point sets

Let S_i be sets of n_i points for $i = 1,2,..,b$. $S = \bigcup_{i=1}^{b} S_i$ and $n = \sum_{i=1}^{b} n_i$. We would like to find a line l which approximates all S_i optimally. For a line l, $N_i(l)$ is the difference between the number of points located above l and that of points located below l in S_i. Consider $N(l) = \sum_{i=1}^{b} |N_i(l)|$, and find a line which minimizes $N(l)$. If $b = 2$, there is a line l called the *ham-sandwitch cut* which satisfies $N(l) = 0$, and an efficient $O(n \log n)$ time algorithm is known [E]. However, if $b \geq 3$, no such efficient algorithm is known, and we solve it by walking on the arrangement generated by the dual lines of points of S.

Theorem 4. *A line which minimize $N(l)$ is found in $O(n^2)$ time and $O(n)$ space*

Next, we consider the function $f_i(l) = \sum_{p \in S_i} d(l, p)$, where $d(l, p)$ is the vertical distance between l and p. We would like to minimize $f(l) = \max_{i=1,2,..,d} f_i(l)$. If $b = 1$, Yamamoto *et al* [YKII] gave a linear time algorithm by using the *prune and search method.* However, for larger b, the topological walk gives an efficient algorithm.

Theorem 5. *We can obtain the optimal approximation line in $O(bn^2)$ time and $O(n)$ space.*

6. Conclusion

We have presented an algorithm to walk on a part $A(H, R)$ of an arrangement. Topological walk is typically efficient if we search a cell with an optimal cost which can be calculated using certain dynamic data. Hence, the importance of topological walk will increase in proportion to the development of dynamic algorithms permitting both insertions and deletions.

References.

[CE] Chazelle, B., and Edelsbrunner, H., "An Optimal Algorithm for Intersecting Line Segments in the Plane," *Proc. 29-th IEEE FOCS pp. 590-600, (1988).*

[E] Edelsbrunner, H., *Algorithms in Combinatorial Geometry,* Springer-Verlag 1987.

[EG1] Edelsbrunner, H., and Guibas, L. J., "Topologically sweeping an arrangement," *J. Comput. Sys. Sci. 38, pp.165-194, (1989).*

[EG2] Edelsbrunner, H., and Guibas, L. J., "Topologically sweeping an arrangement - a correction, " *J. Comput. Sys. Sci., to appear.*

[H] Harary, F., *Graph Theory,* Addison-Wesley 1969.

[*OW*] *Overmars, M.H. and Welzl, E., "New Methods for Computing Visibility Graphs," Proc. 4-th ACM Computational Geometry, pp. 164-171 (1988).*

[YKII] Yamamoto P., Kato K., Imai K., and Imai H., "Algorithms for Vertical and Orthogonal L_1 Approximation of points," *Proc. 4-th ACM Computational Geometry, pp. 352-361 (1988).*