Topological Walk Revisited

Tetsuo Asano[1] Takeshi Tokuyama[2]

Abstract

Topological Walk is an algorithm that can sweep an arrangement of n lines in $O(n^2)$ time and $O(n)$ space. This paper revisits Topological Walk to give its new interpretation in contrast with Topological Sweep. We also present several new applications of Topological Walk to make the distinction clearer.

1 Introduction

Sweeping an arrangement of lines in the plane occurs in various situations in computational geometry. The most well-known algorithm for this purpose is Topological Sweep[6]. It can sweep an arrangement of n lines in $O(n^2)$ time and $O(n)$ space. Topological Walk[5] is another algorithm that can sweep an arrangement in optimal time and space. A catalog of problems which can be solved by Topological Sweep is given in [6]. It was shown in [5] that all of them can be solved by Topological Walk. The most noteworthy feature of Topological Walk is that it can sweep a part of an arrangement optimally. That is, given a convex $s-$gon R and n lines in the plane, it can sweep the arrangement of the lines within the $s-$gon R in $O(K + (n+s)\log(n+s))$ time and $O(n+s)$ space, where K is the total number of intersections of lines within R. Another important feature is that it can find a walk of an arrangement. Here, a walk is a visit of the cells of an arrangement in such a manner that consecutive cells are adjacent to each other. Therefore, it is suitable to finding an optimal cell in an arrangement.

In this paper we give a new interpretation of Topological Walk in contrast with Topological Sweep. We also present several new applications of Topological Walk to make the distinction clearer.

2 A New Interpretation of Topological Walk

2.1 Topological Sweep

Topological Sweep and Topological Walk are both algorithms for sweeping an arrangement of lines in the plane. As is well known, Topological Sweep uses a monotone curve for the sweep line although a vertical or horizontal straight line is used as a sweep line in the plane sweep method. The most important operation in Topological Sweep is an elementary step which exchanges order of two consecutive lines at their intersection. Thus, how to find a next intersection at which an elementary step is carried out is crucial for efficient implementation. For this purpose two horizon trees are defined: Upper and lower horizon trees. When two lines intersect in an upper horizon tree, one with larger slope is extended to the right while one with smaller slope is extended to the right in a lower horizon tree. The next intersection at which an elementary step is done is the place at which two leaf branches meet in both of the trees. Therefore, it is not so easy to characterize the order of such intersections.

2.2 Topological Walk in an entire arrangement

Topological Walk sweeps an arrangement in a totally different manner. To have conceptual understanding we shall imagine a graph representation of an arrangement of lines in the plane. For the time being we assume that there is no degeneracy, that is, no three lines meet at one point. We further assume without loss of generality that there is no vertical line. In addition to given lines we add an almost vertical line L_∞ with slope larger than that of any given line at positive infinity. Note that every line intersects the line L_∞. We cut off each line at the intersection and remove the portion to the right of the intersection. The vertices of the graph are intersections of lines and the leftmost endpoints of lines at negative infinity plus the right endpoint of L_∞ at positive infinity.

[1]Osaka Electro-Communication University, 18-8 Hatsu-cho, Neyagawa, 572 Japan.
[2]Tokyo Research Laboratory, IBM Japan, 623-14 Shimotsuruma, Yamato, Kanagawa, 242 Japan

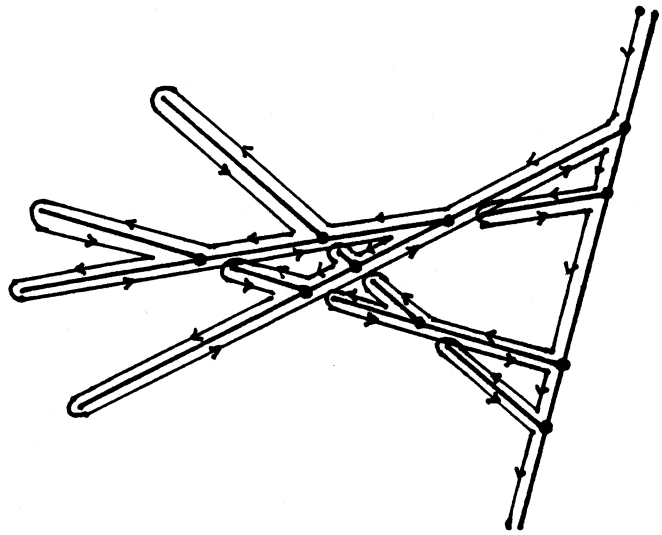

Figure 1: Depth-first search simulating the behavior of Topological Walk.

Then, each intersection is characterized by a pair of lines. We split each intersection into two vertices, one incident only to the edge going down to the right, and the other connecting to the remaining three edges. Notice that the above splitting operation is done at each intersection except for the intersections with the rightmost line L_∞.

It is not so hard to see that the resulting graph is a tree rooted at the topmost vertex of the rightmost line L_∞. Since it is a tree, we can implement depth-first search on the tree with right branch preferred at each vertex (intersection). This completely specifies the order of vertices to be visited. Topological Walk visits the vertices in this order. However, if we construct the graph structure in an explicit manner, it takes $O(n^2)$ space, which is too expensive in many cases. Thus, we implement the depth-first search while updating the tree.

Figure 1 illustrates how the depth-first search is executed on the tree.

We start with the initial upper horizon tree which takes only $O(n)$ space. It is constructed as follows. First of all we sort the lines in the decreasing order of their slopes. Let it be $(L_\infty, L_1, L_2, ..., L_n)$. For simplicity we assume that no two lines have the same slope although this assumption can be easily removed. This assumption implies that this order on slopes coincides with the vertical order of the lines at negative infinity. Then, we draw the line L_∞ at positive infinity. After that we add lines in the above order one by one. That is, we draw each line from the left infinity to the right until the first intersection. When we add a line L_i, we trace the current upper horizon tree toward the root starting from the left endpoint of the line L_{i-1} just added until we encounter the edge intersecting the line L_i.

After constructing the initial upper horizon tree, we start the depth-first search on the tree with right branch preferred. When we reach a vertex incident to two leaf vertices, we execute the elementary step and update the cut (the vertical order of lines intersecting the vertically monotone sweep line). This is just the same as Topological Sweep.

Another important difference of Topological Walk from Topological Sweep is that we can maintain information associated with cells in the arrangement. Since Topological Sweep is based on cuts, it can maintain only $O(1)$ information for each cell to keep the total space complexity $O(n)$. On the other hand, Topological Walk visits cells one by one in consecutive manner (i.e., always from one cell to its adjacent cell). Thus, it can use up to $O(n)$ space at a cell currently visited while keeping the total space linear.

2.3 Topological Walk in a part of an arrangement

As far as an entire arrangement is swept, the difference between Topological Sweep and Topological Walk is not so essential. However, it does not seem that Topological Sweep can be adapted so as to sweep a part of an arrangement in optimal time and space. As was shown in [5], given a convex polygon R and an arrangement, Topological Walk can sweep the arrangement within the polygon R in $O(K + (n+s)\log(n+s))$ time and $O(n+s)$ space, where K and s are the number of intersections within R and that of vertices of R. It is not so straightforward to achieve the bound. When we execute

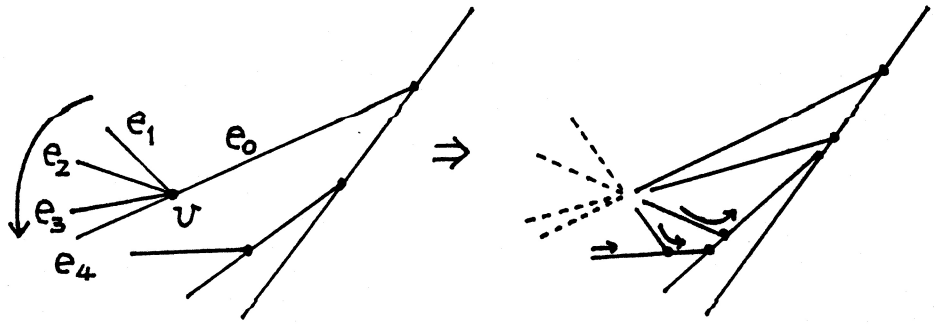

Figure 2: Extended elementary step – degenerate case.

an elementary step at an intersection v, we delete two edges incident to v from the left and extend the *upper left edge e to the right until it encounters an edge of the tree.* To find such an edge, Topological Sweep performs one-directional search from the edge located in the next place of e in the current cut. However, as was shown in [5], one-directional search is not sufficient to achieve the above-described bound. This is why bi-directional search is incorporated in Topological Walk.

3 Degeneracy

Degeneracy causes a big trouble for Topological Sweep. Recall that in Topological Sweep to find the next intersection at which an elementary step is executed we had to examine the two horizon trees to find a pair of lines that define twigs (vertices incident to two leaf branches) in both of the trees. If we maintain all the twigs as ordered pairs of lines (ordered by, say, slopes) in a table, we can determine in $O(1)$ time whether a twig in one horizon tree is also a twig in the other horizon tree. However, if we have degeneracy, such a check cannot be done in constant time. This is the difficulty.

On the other hand, Topological Walk can easily admit degeneracy without complicated trick. Our approach is not to use perturbation but directly to deal with degenerate inputs, like one in [10]. A data structure required for this purpose is very simple. We just maintain a set of edges incident to a vertex in an upper horizon tree by a list of edges in counter-clock wise manner starting from the edge incident to v from the right (note that there is only one such edge). Depth-first search is carried out with right edges preferred as before. When we reach such a vertex v that all the edges incident to v from the left are leaf edges, we execute an extended elementary step. Let the list of edges be $(e_0, e_1, ..., e_k)$ where e_0 is the edge incident to v from the right and the remaining edges are ordered in counter-clock wise manner. Then, we delete the edges $e_1, ..., e_k$ and extend them except for e_k in this order. When we add e_1 we start from the edge located in the next position of e_k in the current cut to find the intersecting edge in the current horizon tree. For $e_i, i = 2, 3, ..., k-1$ we can start from the extension of e_{i-1} which has been just added. See Figure 2.

4 Applications

Since the topological walk can sweep a part of an arrangement in an input sensitive manner, it is an efficient paradigm for almost all the problems solved by sweep methods. A catalog of applications of topological sweep is given in [6]. All of those applications – computing a longest monotone path, computing a longest monotone concave path, computing largest convex subsets, computing a largest empty convex subsets, finding a maximal stabbing line, computing minimum area triangle, and visibility problems – can be solved by using topological walk.

The following is a typical application of sweeping a part of arrangement:

4.1 Finding a max-weight cut line of a straight line graph

We consider a graph G embedded in a plane as a straight-line graph. The vertex set is a set V of points in the plane, and each edge $e = e(u, v)$ is a straight line segment between u and v. We assign a weight $w(e)$ for each edge e. The cardinality of vertices and edges is denoted by n and m respectively. We also consider two subsets S_1 and S_2 of V. We assume that S_1 and S_2 are separable from each other by a straight line. We would like to find a line, a *maximum cut line* l, which separates S_1 and S_2 and maximizes the summation of weights of edges of G by l .

The dual set of the set of separating lines of S_1 and S_2 becomes a convex polygon R, which has less than or equal to $s = |S_1| + |S_2|$ corners. The dual lines of V makes an arrangement in R, which has K cells. Thus, the problem can be solved by using the topological walk.

Theorem 4.1 *A Maximum cut line is found in $O(K + m + n \log n + s \log s)$ time and $O(n + m + s)$ space.*

Proof The proof is given in [4]. □

A similar problem occurs in the design of divide-and-conquer algorithms for the VLSI routing problem, in which S_1, S_2 and V are sets of modules, and G is the net graph of the modules in V. The weights of edges represent the connectivity between modules. There, minimum cut line is needed under the condition that it subdivides V into almost evenly sized sets. See [4] for detail.

4.2 Space efficient optimal cell searching

We consider a cost function $f = f(r)$ associated with the arrangement, and search for the cell with the optimal cost. We assume a dynamic algorithm to compute $f(r)$ exists. The following is the situation: We consider a dynamic data structure $Data(r)$ from which we calculate $f(r)$. We need 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 $T = O(n), I = O(n^\alpha)$ $(0 < \alpha < 1)$, 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. Thus, applying topological walk, we obtain the following theorem:

Theorem 4.2 *W 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 the usual fashion, we need $O(n \log n + KU + T)$ time and $O(K + I)$ space (we omit the terms containing s for simplicity.) 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 $O(n + I)$ space. Thus, the topological walk is always advantageous compared to each of these paradigms. One typical example of the application of topological walk to such a problem is shown in [5]. We show another one in the following.

4.3 Maximal set of intersecting polygons

Consider the following problem: Suppose we are given n convex (possibly open) polygons $P_1, P_2, ..., P_n$ in the plane. We assume that the total number N of edges of the polygons is $O(n)$. We would like to find all maximal subsets of polygons with non-empty intersections. This problem is the key subroutine for extracting curve images in digital images [1]; Typically, each polygon is a slab defined by a pair of parallel lines if we consider the problem of extracting line images.

The defining lines of the edges of the polygons partition the plane into cells. Each cell c is associated with a set $H(c)$ of polygons containing it. Topological Walk can compute all cells associated with maximal subsets of polygons in $O(n^2)$ time and $O(n)$ working space.

The key point is that, we only need to maintain the cuts and one cell for the Topological walk. Therefore, although we store $O(n)$ information in the cell maintained in the algorithm, the total space complexity remains $O(n)$. Such a space-efficient implementation seems to be difficult for Topological Sweep, since one must keep $O(n)$ cells for each of which $O(n)$ information should be stored.

4.4 Linear approximation of multiple point sets

Let S_i be sets of n_i points for $i = 1, 2, ..., b$. $S = \cup_{i=1}^{b} S_i$ and $n = \sum_{i=1}^{b} n_i$. We would like to find a line l, which approximates all S_i simultaneously.

First, we consider a discrete cost function. For a line l, $N_i(l)$ is the absolute value for the difference between the number of points located above l and that of points 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-sandwich cut* which satisfies $N(l) = 0$, and an efficient $O(n \log n)$ time algorithm is known [9]. However, if $b \geq 3$, $N(l)$ may be always positive. No subquadratic algorithm is known for finding an optimal l, and we solve it by searching over the arrangement $\mathcal{A}(\mathcal{D}(S))$ generated by the dual lines of points of S. Naturally, $N_i(l) = N_i(l')$ if $\mathcal{D}(l)$ and $\mathcal{D}(l')$ are in the same cell r of $\mathcal{A}(\mathcal{D}(S))$ for $i = 1, 2, ..., b$. Thus, choosing an arbitrary dual point $\mathcal{D}(l)$ in r, we can define $Data(r)$ consisting of the list $(N_1(l), N_2(l), ..., N_b(l))$ and $N(l)$.

Thus, the situation is:

1. Each cell r of the arrangement is associated with the data $Data(r)$ of $O(b)$ space.
2. $N(r)$ is computed in $O(n)$ time statistically.
3. $Data(r)$ is given in $O(1)$ time from $Data(r')$ of a cell r' adjacent to r.

Thus, usual sweep line method needs $O(nb)$ space and $O(n^2 \log n)$ time. Naive topological sweep needs $O(nb)$ space and $O(n^2)$ time. By applying topological walk, we obtain the following theorem.

Theorem 4.3 *An optimal line l is found in $O(n^2)$ time and $O(n)$ space.*

An optimal line with respect to the above discrete function need not be a good approximation of the point sets. Thus, let us consider a continuous function. For a line $l : y = l(x) = \alpha x + \beta$ and a point $p_0 = (x_0, y_0)$, the vertical distance between l and p_0 is $d(l, p_0) = |\alpha x_0 + \beta - y_0|$. We consider the function $f_i(l) = \sum_{p \in S_i} d(l, p)$ and $f(l) = \max_{i=1,2,...,d} f_i(l)$.

We would like to find an l which minimizes $f(l)$. Yamamoto et al[11] dealt with the problem when $b = 1$, where, since $f(l)$ becomes a convex function, a *prune and search* [8] is applicable and a linear time algorithm is given. However, if b is large, it seems that the topological walk gives an efficient algorithm. We define functions $F_i(x) = f_i(\mathcal{D}(x))$ on the dual plane. Let r be a cell in $\mathcal{A}(\mathcal{H}(S))$. Then, $F_i(x)$ is a linear function of the closure of r. We denote $F_i^{(r)}(x)$ for this linear function extended to the entire plane. $F^{(r)}(x) = \max_{i=1,2,...,b} F_i^{(r)}(x)$ is the convex function, which coincides with $F(x) = f(\mathcal{F}(x))$ on r.

The minimal value $m(r)$ of $F^{(r)}(x)$ is taken at a point $p(r)$, which is not necessarily located in r. For an edge e of r, the maximal value $m^r(e)$ of $F^{(r)}(x)$ on the line containing the edge e is taken at a point $p^r(e)$. If r' and r re cells incident to e, it is easily seen that $m'(e) = m^{r'}(e)$ and $p^r(e) = p^{r'}(e)$. Therefore, we write $m(e)$ and $p(e)$ for $m^r(e)$ and $p^r(e)$ respectively. The following lemma comes from the convexity:

Lemma 4.4 *In r, the maximal value of $F(x)$ is taken at $p(r)$ if $p(r)$ is located on r, else it is taken on an edge of r.*

It needs $O(n)$ time to construct the set of functions $F_i^{(r)}(x), (i = 1, 2, ..., b)$ for a cell r, where $n = \sum_{i=1}^{b} n_i$. However, we can construct them from the set of functions $F_i^{(r')}(x)$ for $i = 1, 2, ..., b$ for an adjacent cell r' to r in $O(1)$ time from $F_i^{(r)}$. If we know the functions $F_i^{(r)}(x)$ for $i = 1, 2, ..., b$, we can compute $(p(r), m(r))$ and $(p(e), m(e))$ for each r and e in $O(b)$ time by using the Meggido's method [8].

Now, we apply the walk on $\mathcal{A}(\mathcal{H}(S))$ to solve the problem. When we visit a new region r, we compute $(p(r), m(r))$. We remark that we, at this stage, do not decide whether $p(r)$ is located in r or not. However, when we walk on an edge e bounding a region r, we judge whether $p(r)$ is on the same side as r with respect to e; If not, we reset $m(r)$ to ∞.

When we apply an elementary step at a vertex v, we compute $(p(e), m(e))$ for each outgoing edge from v. We store it until we apply an elementary step at the right endpoint of e. Suppose we come to a vertex v, whose left region is r and incoming edges are e and e' to apply an elementary step.

We have $(p(e), m(e))$, $(p(e'), m(e'))$, $(p(r), m(r))$, and the *current minimum* M of $F(x)$. If $p(e)$ (resp. $p(e')$) is not located on e (resp. e'), we set it to ∞. Then, we replace M by the minimum of $M, m(e), m(e')$ and $m(r)$, and erase other information. In this way, we can obtain the optimal point (the dual point to the optimal line) in $O(bn^2)$ time and $O(n)$ space.

Theorem 4.5 *We can obtain the optimal approximation line for b sets of points in $O(bn^2)$ time and $O(n)$ space, where n is the sum of the number of points in the point sets.*

Finally, we show that the search on clipped arrangement is also useful in this problem. Assume we can add a condition that the approximating line $y = l(x)$ must satisfy that $\lambda_j \leq l(\xi_j) \leq \mu_j$ for given real numbers ξ_j, λ_j, μ_j for $j = 1, 2, ..., s$. Let us consider the dual line L_j of the point (ξ_j, λ_j) and M_j of (ξ_j, μ_j). The intersection $\mathcal{R}$ of all the upper half planes L_j^+ and lower hyperplanes M_j^- for $j = 1, 2, ..., s$ is a convex region. Then, under above condition, we can find the optimal approximating line by searching on $\mathcal{A}(\mathcal{H}(S); \mathcal{R})$. The algorithm needs only $O((n+s)\log n + bK)$ time, where K is the number of vertices in $\mathcal{A}(\mathcal{H}(S); \mathcal{R})$.

4.5 Partial construction of an arrangement

It is shown in [2] that Topological Walk can construct *at-most-k* levels of an arrangement of n lines in the plane in time $O(nk + n\log n)$ which is optimal since $\Omega(nk)$ line segments are included there. This slightly improves the result by Everett et al. [7] in the space complexity.

5 Concluding Remarks

In this paper we have presented a new interpretation of Topological Walk Algorithm[5] in contrast with Topological Sweep Algorithm[6] together with a new catalog of applications. We have also described a positive treatment of degeneracy. Implementation of the algorithm is easy since no complicated data structure is needed.

References

[1] T. Asano, N. Katoh and T. Tokuyama: "A Unified Scheme for Detecting Fundamental Curves in Binary Edge Images," *SIGAL Technical Report of Inform. Process. Society of Japan*, AL38-7, pp.49-56 (1994).

[2] T. Asano and T. Tokuyama: "Partial Construction of an Arrangement of Lines and Its Application to Optimal Partitioning of Bichromatic Point Set," to appear in *Trans. (E) of IEICE of Japan.*

[3] T. Asano and T. Tokuyama: "Algorithms for Projecting Points to Give the Most Uniform Distribution with Applications to Hashing," *Algorithmica*, pp.572-590 (1993).

[4] T. Asano and T. Tokuyama: "Circuit Partitioning Algorithm Based on Geometry Model," *Algorithmic Aspects of VLS Layout, Lecture Notes Series on Computing – Vol. 2, ed. by M. Sarrafzadeh and D. T. Lee, World Scientific*, pp.199-212 (1993).

[5] T. Asano, L. Guibas, and T. Tokuyama: "Walking in an Arrangement Topologically," to appear in *Int. J. of Comput. Geom. and Appl.*

[6] H. Edelsbrunner and L.J. Guibas: "Topologically Sweeping an Arrangement," *J. Comp. and Sys. Sci.*, 38 pp.165-194 (1989).

[7] H. Everett, J. -M. Robert, and M. van Kreveld: "An Optimal Algorithm for Computing ($\leq k$)−Levels with Applications to Separation and Transversal Problems," *Proc. 9th ACM Symposium on Computational Geometry*, pp.38-46 (1993).

[8] N. Meggido: "Linear Time Algorithm for Linear Programming in R^3 and Related Problems," *SIAM J. Comput.*, 12, pp.759-776 (1983).

[9] H. Edelsbrunner: "Algorithms in Combinatorial Geometry," EATCS Monographs on Theoretical Computer Science 10, Springer-Verlag (1987).

[10] C. Burnikel, K. Mehlhorn and S. Schirra: "On Degeneracy in Geometric Computations," *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.16-23 (1993).

[11] P. K. Yamamoto, K. Kato, K. Imai and H. Imai: "Algorithms of Vertical and Orthogonal L_1 Approximation of Points," *Proceedings of 4th ACM Symposium on Computational Geometry*, pp.352-361 (1988).