SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1286–1302, December 1993

COMPUTING A FACE IN AN ARRANGEMENT OF LINE SEGMENTS AND RELATED PROBLEMS*

BERNARD CHAZELLE†, HERBERT EDELSBRUNNER‡, LEONIDAS GUIBAS§, MICHA SHARIR¶, AND JACK SNOEYINK¹

Abstract. This paper presents a randomized incremental algorithm for computing a single face in an arrangement of n line segments in the plane that is fairly simple to implement. The expected running time of the algorithm is $O(n\alpha(n)\log n)$. The analysis of the algorithm uses a novel approach that generalizes and extends the Clarkson–Shor analysis technique [in *Discrete Comput. Geom.*, 4 (1989), pp. 387–421]. A few extensions of the technique, obtaining efficient randomized incremental algorithms for constructing the entire arrangement of a collection of line segments and for computing a single face in an arrangement of Jordan arcs are also presented.

Key words. computational geometry, arrangements, randomized incremental algorithms, probabilistic backwards analysis, Davenport–Schinzel sequences

AMS subject classifications. 68P05, 68Q20, 68R99, 51M99

1. Introduction.

We consider the following problem. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a collection of n line segments in the plane, and let p be a point not lying on any of the segments. We wish to compute the face that contains p in the arrangement $\mathcal{A}$ of S. This problem arises in many applications, such as motion planning [9]. It has been shown in [9], [15] that the combinatorial complexity of such a single face is $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. This bound is shown in [19] to be tight in the worst case; as a matter of fact, the construction in [19] gives a set S of n line segments whose lower envelope has complexity $\Omega(n\alpha(n))$.

The problem of computing a single face has been studied by Edelsbrunner, Guibas, and Sharir [6]; they have given a deterministic algorithm that takes time $O(n\alpha(n)\log^2 n)$ in the worst case. This is less efficient than the best-known algorithm for computing the envelope of n segments, due to Hershberger [10], which runs in optimal $O(n \log n)$ time. This discrepancy between the two algorithms is intriguing because the maximum combinatorial complexity of a single face and of the lower envelope in an arrangement of n segments is asymptotically the same. We remark that in the special case where S is a collection of lines, computing a single face can be trivially done in time $O(n \log n)$. Another special case is when S is a collection of rays. A recent paper [1] shows that the complexity of a single face in this case is $O(n)$ and that the face can be constructed in time $O(n \log n)$. Both these algorithms are deterministic.

In this paper we (almost) close the gap by providing a simple randomized incremental algorithm for computing a single face in an arrangement of general segments, whose expected

*Received by the editors June 27, 1991; accepted for publication (in revised form) July 21, 1992. A preliminary version of this paper appeared in *Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms*, Association for Computing Machinery, New York, 1991, pp. 441–448.

†Department of Computer Science, Princeton University, Princeton, New Jersey 08540. The research of this author was supported by National Science Foundation grant CCR-87-00917.

‡Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. The research of this author was supported by National Science Foundation grant CCR-89-21421.

§DEC Systems Research Center, Digital Equipment Corporation, Palo Alto, California, 94301, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, and Department of Computer Science, Stanford University, Stanford, California 94305.

¶School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The research of this author was supported by Office of Naval Research grant N00014-90-J-1284, by National Science Foundation grants CCR-89-01484 and CCR-91-22103, and by grants from the U.S.–Israeli Binational Science Foundation, the Fund for Basic Research administered by the Israeli Academy of Sciences, and the German-Israeli Foundation for Scientific Research and Development.

¹Department of Computer Science, Stanford University, Stanford, California 94305.

running time is $O(n\alpha(n)\log n)$. (The expectation is taken over the randomizations used by the algorithm, and the bound holds for any input data.) We have learned that Clarkson has obtained a similar result in an unpublished work, using a different approach. The algorithm is similar in some features to the trapezoidal decomposition algorithm of [18], the intersection algorithms of [2] and [14], and the Delaunay triangulation algorithm of [8]. Like the latter, it is a purely on-line algorithm that needs no prior information about the as-yet uninserted segments. We also mention that recently Mitchell [13] has obtained a deterministic algorithm for constructing a single face, whose running time is $O(n\log^2 n)$.

A main novel feature of our algorithm is its analysis, which provides a useful extension of the probabilistic technique of Clarkson and Shor [5] to a domain where the interesting events that need to be counted are more difficult to specify. The reason is that the decision of what features of the arrangement of the segments in S appear on the desired face is global and cannot be determined from the local structure of the features. Such a locality is required in Clarkson and Shor's analysis and, for that matter, in all the randomized algorithms we have mentioned. Our analysis finesses this issue by applying a more general framework, which, as a consequence, also leads to simplified proofs. We expect that there will be additional applications of our technique to other contexts, thus extending the usefulness of the Clarkson–Shor method.

Our technique also can be generalized to other contexts, as discussed in §4. These problems include the construction of the entire arrangement of a given collection of line segments (§4.1) and computing a single face in an arrangement of curved segments (§4.2). In these extensions our technique yields algorithms with optimal or close-to-optimal expected time and storage complexities, matching or improving previously known algorithms. Section 2 presents the incremental algorithm, developing it to a level of detail that shows that it is indeed easy to implement. Section 3 gives the analysis of the algorithm. We conclude the paper in §5 with a discussion of our results and some open problems.

2. The algorithm. As mentioned in the introduction, the algorithm to be described in this section is incremental, that is, it computes the desired face by adding the segments one at a time. Section 3 will show that if the segments are inserted in a random order, then the expected behavior of the algorithm is very good. We describe the algorithm in detail, to convince the reader that the algorithm is easy to implement. A compact description of the algorithm in pseudocode is given at the end of this section. Let $s_1, s_2, \dots, s_n$ be the insertion sequence, so that at the ith step the algorithm adds s_i to the data structure built for s_1 through s_{i-1}. For convenience we start with a rectangular frame big enough to enclose all line segments in S, as well as the special point p defining the face f that we want to compute. We will be interested only (without loss of generality) in the portion of f within the frame. For $0 \le i \le n$ let f_i denote the face in the arrangement defined by $s_1, s_2, \dots, s_i$ that contains p, clipped to within the frame (f_0 is just the frame). We also assume that there are no degenerate cases, such as three segments meeting at a point, an endpoint of one segment lying on another segment, or two intersections with the same x coordinate; this assumption is justified by the algorithmic method of [7].

Although the face f_i is uniquely determined by the first i line segments, the data structure that we use to represent it is not — it also depends on the sequence in which the line segments are added. This is very much like in the case of a binary search tree constructed by repeated insertions, but without a balancing operation: the sorted sequence of the input is unique, but the tree that represents it depends on the sequence of insertions.

The main idea that leads to the data structure and algorithm of this paper is that while the central aim is to construct the face marked by p, we keep around everything ever built (typically portions of the earlier versions of the same face) as an aid in the search operations.

An important rule is that these older parts of the structure are not further refined during the insertion process — this helps keep the size of the extra structure within limits.

2.1. The conceptual level. The data structure that represents f_i, the face after adding the first i line segments, consists of three sorts of geometric information. These three parts should be considered as fundamentally different at a conceptual level, although we will represent them in a uniform way at a lower level. The three parts are the *city* (the face), the *suburbs* (the complement of the face), and the *history*.

2.1.1. The city. After i line segments are added, the face f_i is the city. It is a (not necessarily simply connected) polygonal region, as shown in Fig. 1. Its boundary consists of a finite number of contour cycles; one is the outer cycle (which, in case f_i is unbounded, coincides with the frame boundary), and all others define holes in the city. We represent the city by a collection of trapezoids generated by drawing a vertical line up and down from each vertex until it hits the boundary of the city again. These vertical edges, called *sides*, are drawn only inside the city—see Fig. 2. Two trapezoids are said to be adjacent if they (partially) share a vertical side.

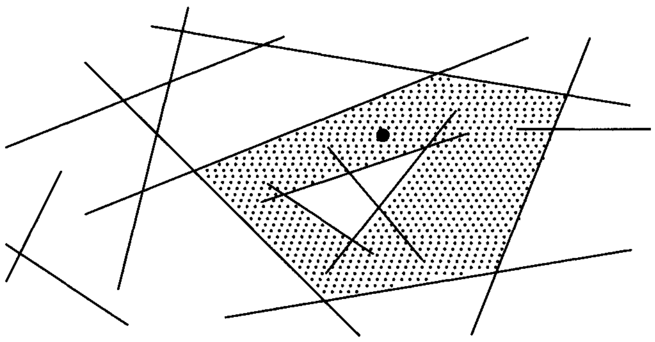

FIG. 1. *The input consists of a set of line segments and a point inside a frame. It defines a face, which we call the city.*

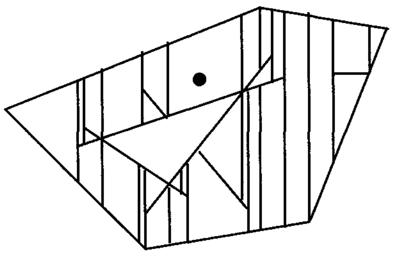

FIG. 2. *The city is decomposed into trapezoids by drawing vertical sides through endpoints and intersection points.*

There is a small number of different types of trapezoids, each defined by at most four line segments. A unique line segment contributes the *floor* (the bottom edge) of a trapezoid Δ, and, similarly, a unique line segment contributes the *ceiling* (the top edge). The left and right sides are each defined

(a) by an endpoint of another line segment,

(b) by another line segment intersecting the floor line segment,

(c) by another line segment intersecting the ceiling line segment, or

(d) as the intersection of the floor with the ceiling.

This makes 16 types of trapezoids altogether. One of the types is impossible, namely, where both the left and the right sides are case (d). Four of the 15 remaining types are shown in Fig. 3.

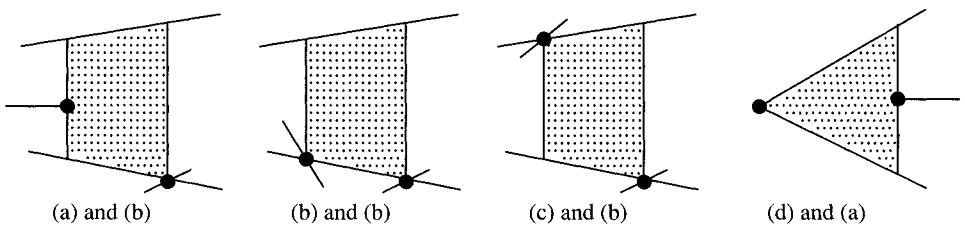

FIG. 3. *Four possible types of trapezoids.*

A trapezoid thus defined has one, two, three, or four adjacent trapezoids; four only if both the left and the right sides are case (a). Notice that the trapezoids that compose the city at stage i depend only on the set of segments $s_1, s_2, \dots s_i$ and not on the particular order in which these segments were inserted.

2.1.2. The suburbs. As new line segments are added the city gets smaller. Each new line segment may chop off parts of the city by separating them from the point p. When a portion of the city is thus disconnected from p, it is properly decomposed into trapezoids and these trapezoids are added to the representation of the complement of the city, the so-called suburbs. It is thus natural to represent the suburbs in the same way as the city, namely, as a collection of trapezoids with adjacency relations.

At any point in time the trapezoids of the city and the suburbs define a decomposition (a tiling) of the entire frame. It should be noted, however, that this decomposition is not edge-to-edge, in the sense that a vertex of some trapezoid may lie in the middle of an edge of another trapezoid. We view each edge of our diagram as two sided, so that the above vertex is not part of the description of the second trapezoid — each trapezoid is bounded by one-sided edges. The same distinction was necessary in the analysis of [14]. There is, however, an important difference between vertical and nonvertical edges. By our general position assumptions, at most one point of a left or right side can also be a corner of (two) other trapezoids (in case (a)), but arbitrarily many such points can lie on the floor or ceiling. For this reason we define and store adjacencies only across vertical sides.

An important difference between city and suburbs is that the former gets further refined as new line segments are added, while the latter only expands by the addition of new trapezoids chopped off from the city. A trapezoid of the suburbs, once created, remains part of the suburbs forever.

2.1.3. The history. There is a third type of trapezoid in our structure. This type consists of trapezoids that belonged to earlier versions of the city and had to be removed because they were cut by a new line segment. Such a trapezoid Δ is not deleted from the structure. Instead, it remains as part of the history. The new trapezoids, generated by the addition of the new line segment, that overlap Δ are added to the structure as children of Δ. Depending on how the new line segment cuts Δ, it can have two, three, or four children (see Fig. 4). In effect, Δ is removed from the representation of the city and is now part of a hierarchical structure of trapezoids built on top of the decomposition described as city plus suburbs. As will be detailed in the following, certain children trapezoids are merged with adjacent children trapezoids of neighbor trapezoids.

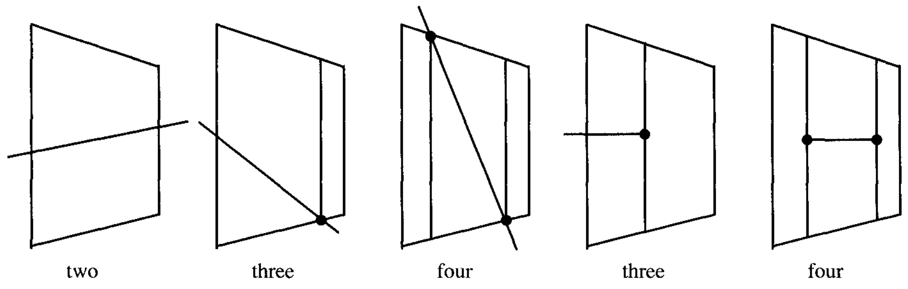

FIG. 4. *The different ways in which a trapezoid can be split by a new line segment.*

The collection (hierarchy) of such trapezoids that once belonged to the city but were later destroyed is called the *history*. The history trapezoids, together with those defining the city and suburbs, are all connected together by the children pointers in a directed acyclic graph. (Because of the merging of children trapezoids that was previously mentioned, the graph is not necessarily a tree.)

2.1.4. Adding a line segment. To understand exactly how our data structure looks at any stage of the incremental process, we need to understand how a line segment, say, s_{i+1}, is added. As mentioned earlier, already existing trapezoids of the suburbs and the history are unaffected by this insertion.

Here is how the city trapezoids are updated. First, we compute and draw $f_i \cap s_{i+1}$, which is a collection of portions (*edges*) of s_{i+1}. These new edges make it necessary to update the decomposition of f_i: the trapezoids of f_i that intersect s_{i+1} become part of the history, and the new trapezoids generated are included in the structure as their children. To understand this process, let us define the *transient* city g_i as f_i after s_{i+1} has been added and the decomposition of f_i into trapezoids has been updated. Of course, g_i may contain several trapezoids, some newly created and some pre-existing as part of the city at stage i, that are no longer accessible from p and thus not part of f_{i+1}. To obtain f_{i+1} we must thus remove all these trapezoids from g_i and place them in the suburbs. Figure 5 shows the development of the suburbs when the line segments are added in the indicated sequence.

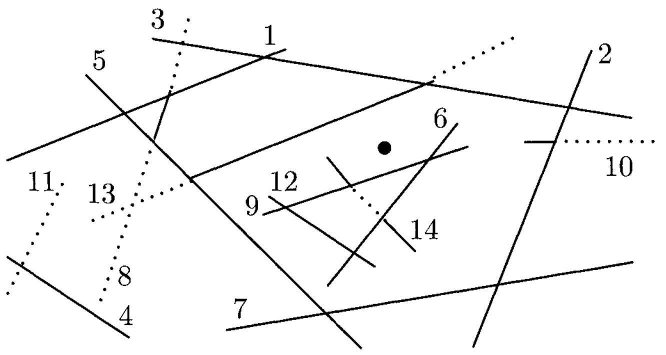

FIG. 5. *The subdivision is constructed by inserting the line segments in the indicated sequence. Line segment* 11 *is not drawn at all because it lies completely outside the city at the time it is added. Only portions of line segments* 8, 10, 13, *and* 14 *are drawn. The boundary of the final city consists of two contour cycles.*

To help us reflect on the process of adding line segments and updating the structure, let us look at what distinguishes suburb trapezoids from history trapezoids. For a trapezoid Δ

defined by line segments s_a, s_b, s_c, and s_d to be part of the suburbs or the history after the first $i + 1$ line segments are added, it must have occurred as part of at least one of the cities f_j or g_j for $1 \leq j \leq i$.

1. If there is a j, $j \leq i$, so that Δ is part of f_j but not part of g_j, then Δ is now part of the history because it was cut by s_{j+1} while being a city trapezoid.

2. If there is a j, $j \leq i$, so that Δ is part of g_j but not part of f_{j+1}, then it is now part of the suburbs because it was cut off from the city by s_{j+1}.

There is a subtlety in condition 2, which we take the time to discuss now. The trapezoid Δ can be of the type that belongs to f_j and to g_j but not to f_{j+1}, or it can be of the type that belongs to g_j but neither to f_j nor to f_{j+1}. In the latter case we call Δ a *transient* trapezoid because it lives a particularly short life. It will be important later to remember that transient trapezoids can only be part of the suburbs, not of the history.

2.2. The data structure. We will now be more specific about the data structure that is incrementally constructed by the algorithm. It consists of a directed acyclic graph (a dag) that stores the city, the suburbs, and the history, all at once, a linear array for the line segments; and a union-find structure for the line segments. We discuss the easy structures first.

2.2.1. The linear array. By keeping the line segments in a linear array we can use a single index rather than four real numbers wherever a line segment is to be stored. We assume that the segments are stored in the array in their insertion order.

2.2.2. The union-find structure. This structure allows us to keep track of topological changes that happen to the boundary of the city as line segments are added. Each set in the structure represents a connected component of the union of line segments and portions of line segments as drawn by the algorithm. Although a single line segment can have several disjoint portions drawn, they all belong to the same connected component. We can thus represent such a component by the set of line segments that contribute edges to it. Note that each contour cycle is part of a possibly bigger connected component. However, we will need the union-find structure only to the extent that it represents contour cycles. We will use a simple union-find structure, in which every element (segment) has a pointer to its current subset (contour cycle), so that each find operation takes $O(1)$ time. To form the union of two subsets we change the pointers of all the elements in the smaller set to be the same as those of the elements in the larger set. The overall cost of all unions is thus $O(n \log n)$.

2.2.3. The dag. Each node of the dag stores a unique trapezoid (city, suburbs, or history), represented by four indices (line segments) and a few bits to indicate the type. The dag has a unique root that stores the frame as a single trapezoid. Each interior node stores a history trapezoid and contains pointers to its (at most) four children. The city and suburb trapezoids are stored in the leaves of the dag, and each leaf has pointers to the at-most four leaves storing adjacent trapezoids. To distinguish the three types of nodes we mark history and suburb trapezoids as such and leave city trapezoids unmarked.

2.3. How it really works. Recall the basic steps that have to be performed when a line segment s_{i+1} is added.

1. We compute all portions of $s_{i+1} \cap f_i$.

2. Using these portions, we update the trapezoidal decomposition of f_i to get g_i. Destroyed trapezoids become history.

3. The new city f_{i+1} is the component of g_i that contains p. All other trapezoids in g_i need to be labeled as suburbs.

The portions of $s_{i+1} \cap f_i$ are computed by propagating s_{i+1} from the root of the dag down to the leaves. Each trapezoid of f_i intersected by s_{i+1} is updated, and the new city and suburbs are differentiated with the help of the union-find structure. Here are the details of how steps 1 through 3 are implemented.

2.3.1. Intersecting the new line segment with the city. Starting at the root of the dag, the line segment s_{i+1} is propagated downward to all leaves whose trapezoid meets s_{i+1}. When we are at an internal node ν, we know that s_{i+1} meets the history trapezoid of ν and we mark ν as already visited. Next we recursively visit the children of ν whose trapezoids meet s_{i+1} and that are not yet marked. The order in which we visit them is such that they meet s_{i+1} in sequence from left to right. Because of this ordering, the leaves are also visited in the sequence in which their trapezoids meet s_{i+1} from left to right.

2.3.2. Updating the trapezoidal decomposition. When a leaf storing a suburb trapezoid is reached, we do nothing. When a city trapezoid is reached, we do all the work. We distinguish six cases as illustrated in Fig. 6. We denote the leaf by λ.

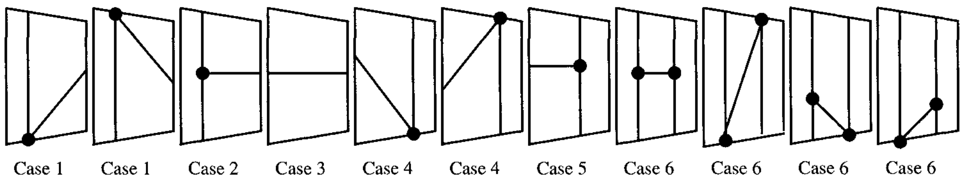

FIG. 6. *Updating the trapezoidal decomposition.*

In every case we construct the appropriate number of children, change λ from city to history, and use λ's former adjacency pointers to connect it to its children. In case 1, depending on whether the endpoint or intersection point that defines the right side of the old trapezoid is above or below s_{i+1}, one of the two children trapezoids that lie above and below s_{i+1} is not a properly defined trapezoid yet. This trapezoid will be merged with the adjacent child trapezoid of the next leaf. The same is true in case 2. The only difference between the two cases is that in case 1 we remember the line segment s_a that contains the left endpoint of the currently processed portion of $s_{i+1} \cap f_i$ (s_a contains the top or bottom edge of λ) and the two children of λ that lie above and below the current portion of $s_{i+1} \cap f_i$. In case 3 one of the child trapezoids is merged with a child trapezoid of the preceding leaf, and the same happens in cases 4 and 5. In case 4 we also take note of the line segment s_b that contains the right endpoint of the current portion of $s_{i+1} \cap f_i$. The pair (s_a, s_b) delimits this portion. The pair will be processed as described below. Finally, case 6 is in a way the easiest, because it requires only the construction of the four children for λ and no merging of trapezoids (nodes) is necessary. If, in case 6, s_{i+1} meets both top and bottom edges of λ, we immediately obtain the corresponding delimiting pair (s_a, s_b). In all other subcases this pair is undefined.

Let us say a few more words about the merging of children trapezoids. As we follow s_{i+1} from left to right, we maintain the current two children trapezoids that lie above and below s_{i+1}. One of these children may be open-ended on the right. When we reach a trapezoid λ and we are in case 3, 4, or 5, we extend the open-ended trapezoid (if any) and merge it with the appropriate child of λ. In case 3 we exit λ on the right with one of the children trapezoid closed and one open ended, as appropriate; in cases 4 and 5 both are closed. In cases 1 and 2 we create two new children accompanying s_{i+1} and leave one of them closed and one open ended, as above. In case 6, as mentioned above, no merging of children is necessary.

2.3.3. Maintaining the topology. After all portions of $s_{i+1} \cap f_i$ are added to the city decomposition as described, we have effectively obtained the trapezoidal decomposition of the transient city g_i. As a by-product, for each portion of $s_{i+1} \cap f_i$ we also get a pair (s_a, s_b)

of line segments that delimit the portion, and two trapezoids, one that lies immediately above it and one immediately below it. This extra information is not properly defined, but it is not needed anyway if the portion contains one of the endpoints of s_{i+1}.

For each such pair (s_a, s_b) we do the following. First we compute c_a and c_b, the names of the connected components containing s_a and s_b, respectively, by doing two find operations. If the two components are different, then we just have to union the two components to reflect the fact that the new segment s_{i+1} has merged the two contours into one; in this case the current portion of $s_{i+1} \cap f_i$ does not disconnect any portion of f_i from p. If $c_a = c_b$, i.e., the two contours are the same, then we have to work harder because the old city area on one side of the current portion of s_{i+1} now becomes suburb. It is not possible to decide locally which side this is. We thus perform two graph traversals in lock-step, starting at the two trapezoids (nodes) provided with s_a, which are trapezoids that lie on the two sides of the current portion of s_{i+1}. These traversals use the adjacency pointers and advance in a strictly alternating fashion, one trapezoid at a time. The traversals stop when one region is exhausted without finding the trapezoid that contains p (the exhausted region is now suburb and its trapezoids must therefore be relabeled) or when the trapezoid containing p is found (in this case the other region becomes suburb and its trapezoids must be relabeled). In either case the amount of time spent is at most proportional to the number of city trapezoids that became suburb.

Up to minor details, such as the fact that s_{i+1} should be added to the proper contour cycle or start a new one of its own, this concludes the description of the algorithm. For the convenience of the reader, we summarize the algorithm in pseudocode.

```
procedure $face(p, S)$;          % $p$ is a point and $S$ is a set of segments
    initialize $dag$ to a single node containing the enclosing frame;
    store a random permutation of $S$ in an array $[s_1, s_2, \ldots, s_n]$;
    initialize a union-find data structure on the segments, each
      stored as a singleton set;

    for all $i = 1, \ldots, n$ do
        perform a depth-first search of $dag$ to find all trapezoids crossed by $s_i$:
            construct a list $x\_trapezoids$;
            visit children of each node of $dag$ that $s_i$ crosses in
              left-to-right order (along $s_i$);
            mark each visited node (so as not to visit it again);
            add each city leaf crossed by $s_i$ to $x\_trapezoids$;

        update the trapezoidal decomposition:
            for each trapezoid $\lambda$ in $x\_trapezoids$ do
                depending on the type of $\lambda$ do
                case 1:
                    initialize $top\_trap$ and $bot\_trap$ to the subtrapezoids of $\lambda$
                      lying above and below $s_i$, respectively;
                    mark which of the two is open-ended on the right and which
                      is closed;
                    change $\lambda$ to a history node in $dag$;
                    add the 3 newly created subtrapezoids to $dag$;
                    store pointers from $\lambda$ to them;
                    the type of the new nodes (city/suburb) is not set as yet;
                    store adjacency pointers between the new nodes as appropriate
                      (the left subtrapezoid also inherits the left-adjacency
        pointer(s) from $\lambda$);
      set $s_a$ to the segment containing the top or bottom portion
        of $\lambda$, whichever $s_i$ intersects;
      set $\tau_1$ and $\tau_2$ to *top_trap* and *bot_trap*, respectively;
  case 2:
      proceed as in case 1 except for setting $s_a$, $\tau_1$, and $\tau_2$;
      both *top_trap* and *bot_trap* are now adjacent to the left
        subtrapezoid of $\lambda$;
  case 3:
      update *top_trap* and *bot_trap* by appending to the open-ended among
        them the corresponding top or bottom subtrapezoid of
        $\lambda$ and setting the remaining variable to the other
        subtrapezoid of $\lambda$;
      again mark which of the two is now open-ended and which is closed;
      add to *dag* the subtrapezoid that is not appended;
      change $\lambda$ to history;
      store pointers from $\lambda$ to these two subtrapezoids;
      store adjacency pointers between the new node of *dag* and the
        subtrapezoid preceding it on the left;
  case 4:
      update *top_trap* and *bot_trap* as in case 3;
      both resulting subtrapezoids are now closed;
      update *dag* as described;
      also, add the right subtrapezoid of $\lambda$ to *dag*;
      store a pointer to it from $\lambda$;
      store adjacency pointers between the subtrapezoids as appropriate
        (the right subtrapezoid also inherits the right-adjacency
        pointer(s) of $\lambda$);
      set $s_b$ to the segment containing the top or bottom portion
        of $\lambda$, whichever $s_i$ intersects;
  case 5:
      proceed as in case 4, except for setting $s_b$;
      the right subtrapezoid is now adjacent to both *top_trap*
        and *bot_trap*;
  case 6:
      construct the 4 subtrapezoids of $\lambda$, all closed, as new nodes of *dag*;
      change $\lambda$ to history;
      add pointers from $\lambda$ to these 4 nodes;
      store adjacency pointers between these nodes as appropriate
        (including the carrying over of adjacency pointers of $\lambda$);
      if $s_i$ crosses the top (respectively, bottom) edge of $\lambda$, set $s_a$
        (respectively, $s_b$) to the segment containing that edge;
  end case;

  if either $s_a$ or $s_b$ is undefined then
      mark all new nodes of *dag* as city;
  else
      find $s_a$ and $s_b$ in the union-find structure;
      if $s_a$ and $s_b$ are in different subsets then
```

```
                    union these subsets;
                    mark all new nodes of $dag$ as city;
                else update the topology of the face:
                    perform simultaneously, in lock-step, two searches of $dag$,
                      starting at $\tau_1$, $\tau_2$, respectively, and following the adjacency
                      pointers, until either one search is exhausted or the
                      trapezoid containing $p$ is encountered;
                    in either case, mark all nodes encountered in one search as city
                      nodes and in the other search as suburb nodes, as appropriate;
                end if;
            end if;
            if either $s_a$ or $s_b$ is defined then
                union $s_i$ with $s_a$ or with $s_b$, whichever is
                  defined;
            end if;
        end for;
    end for;

    return all city leaves of $dag$;

end procedure;
```

Remark. After completing the algorithm we notice that suburb trapezoids are fairly useless when we add line segments. We could prune the dag by removing all leaves that store suburb trapezoids and, recursively, all nodes storing history trapezoids that thus end up without children. However, the analysis in §3 will reveal that the savings possible by this optimization are not substantial (at least asymptotically).

3. The analysis.

The algorithm presented in §2 is a purely on-line algorithm for the single-face problem. In this section we show that if the segments are inserted according to a random permutation, then the expected behavior of our algorithm is very good in terms of both time and storage. We remark that without the randomization there can be situations where the space and time performance of our algorithm become quadratic in n. Such a situation is shown in Fig. 7.

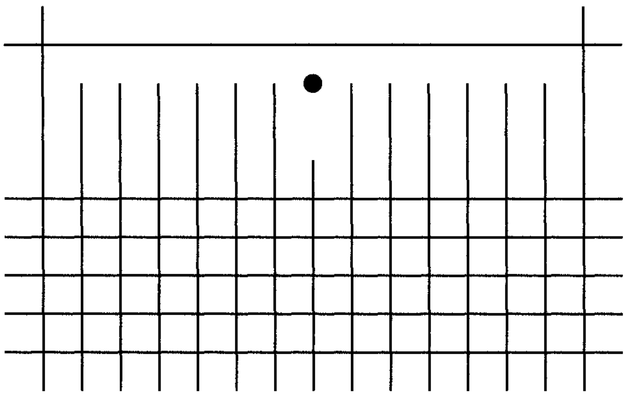

FIG. 7. *An example where our algorithm will require quadratic time if all the vertical line segments are inserted before the horizontal ones, which are then added from bottom to top.*

Recall that the main data structures used in our algorithm are a linear array, a union-find structure, and a dag. The sizes of the first two structures are proportional to n, the number of line segments. The size of the dag is proportional to the number of trapezoids constructed during the course of the algorithm. We will show in §3.1 that the expected number of trapezoids is $O(n\alpha(n))$.

The time spent by the algorithm is split among union-find operations, constructing trapezoids, searching for and labeling suburban trapezoids, and propagating the line segments down the dag. The cost of the union-find operations is at most $O(n \log n)$, even with a simple structure that supports $n - 1$ unions in amortized $O(\log n)$ time per operation and each find operation in constant time. By the results of §3.1 the expected number of find operations is $O(n\alpha(n))$, which thus takes expected time no more than $O(n\alpha(n))$. The same is true for constructing and labeling trapezoids because our lock-step search strategy ensures that the cost of these steps is proportional to the number of constructed trapezoids. Indeed, the cost of the lock-step search is easily seen to be proportional to the number of trapezoids in the smaller of the two face portions traversed and is thus proportional to the number of trapezoids that have been now disconnected from the city. Since a trapezoid can leave the city at most once, the claim follows. To understand the cost of propagating the line segments down the dag, let us define the *weight* of a trapezoid Δ, denoted $w(\Delta)$, as the number of line segments that intersect Δ. The cost is then proportional to $\sum w(\Delta)$, where the sum is taken over all trapezoids Δ constructed by the algorithm. We will show in §3.2 that the expectation of this sum is $O(n\alpha(n) \log n)$.

Hence, anticipating these results, we obtain the main result of the paper.

THEOREM 3.1. *Given a set of n line segments and a point in the plane, the algorithm of §2 constructs the face that contains the point in the arrangement of the line segments, in expected time $O(n\alpha(n) \log n)$ and expected space $O(n\alpha(n))$.*

3.1. The expected number of trapezoids. Before starting the probabilistic analysis, recall that the number of transient trapezoids (that is, trapezoids that are constructed but are never part of the city proper; see §2.1) cannot exceed the number of other trapezoids by more than a factor of 4. This is because all transient trapezoids belong to the suburbs and are therefore stored on the leaf level of the dag and because each inner node of the dag has at most four children. This observation allows us to consider only trapezoids that belonged to the city at the time they were created.

LEMMA 3.2. *The expected number of trapezoids constructed by the algorithm is $O(n\alpha(n))$.*

Proof. Fix a trapezoid Δ, and define the following two events: (i) $X_{r,\Delta}$: Δ is a trapezoid in f_r, which is the city as defined after adding the first r segments. (ii) $Z_{r,\Delta}$: Δ is a trapezoid in some f_i, for $0 \leq i \leq r$.

Clearly, $Z_{n,\Delta} = \bigcup_{r=0}^{n} X_{r,\Delta}$, and $\sum_\Delta \mathbf{P}[Z_{n,\Delta}]$ is the expected number of non-transient trapezoids constructed by the algorithm, where the sum is taken over all trapezoids Δ defined by at most four line segments each, as detailed in §2.1. By the remark before the lemma, $5 \sum_\Delta \mathbf{P}[Z_{n,\Delta}]$ is an upper bound on the expected number of constructed trapezoids, whether transient or not.

Notice that a trapezoid Δ can be constructed only once; thus $\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}$ is nonempty for at most one r, namely, if Δ is constructed at the time the rth segment is added. Therefore, $Z_{n,\Delta}$ is the disjoint union of the events $\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}$, for $1 \leq r \leq n$. This is true for all trapezoids Δ, except for the frame, which is the only trapezoid of f_0, by definition. It follows that

$$\sum_\Delta \mathbf{P}[Z_{n,\Delta}] = 1 + \sum_\Delta \sum_{r=1}^{n} \mathbf{P}[\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}].$$

By the definition of conditional probability we have

$$\mathbf{P}[\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}] = \mathbf{P}[\overline{X}_{r-1,\Delta} | X_{r,\Delta}] \cdot \mathbf{P}[X_{r,\Delta}].$$

To estimate the conditional probability, we note that Δ is defined by at most four line segments and, if we assume that Δ is in f_r, then it was also in f_{r-1} if and only if the rth segment to be added was not one of these at most four segments. This implies

$$\mathbf{P}[\overline{X}_{r-1,\Delta} | X_{r,\Delta}] \leq \frac{4}{r}.$$

The preceding equations thus imply

$$\sum_{\Delta} \mathbf{P}[Z_{n,\Delta}] \leq 1 + \sum_{\Delta} \sum_{r=1}^{n} \frac{4}{r} \mathbf{P}[X_{r,\Delta}] = 1 + \sum_{r=1}^{n} \frac{4}{r} \sum_{\Delta} \mathbf{P}[X_{r,\Delta}]. \tag{1}$$

However, $\sum_{\Delta} \mathbf{P}[X_{r,\Delta}]$ is the expected number of trapezoids in the city f_r, after r line segments have been added. By the results of [9], [15], [19], f_r can have at most $O(r\alpha(r))$ edges and, therefore, at most $O(r\alpha(r))$ trapezoids. This finally gives

$$\sum_{\Delta} \mathbf{P}[Z_{n,\Delta}] = \sum_{r=1}^{n} O(\alpha(r)) = O(n\alpha(n)). \qquad \square$$

Remark. The analysis just presented is fairly general, and so we would like to restate it in more abstract terms, which will be exploited in §4. In general, we have a set of n objects (line segments in our case) that we add incrementally in random order to form some structure (a single face in our case). This structure is represented as a collection of regions (trapezoids in our case), each defined by at most some constant number b of objects (4 in our case). Let $M(r)$ denote the expected number of regions composing the structure after r objects have been added. Then the expected number of regions ever constructed during the randomized incremental process is at most $\sum_{r=1}^{n} \frac{b}{r} M(r)$, provided that if a region is present in the structure after r steps and the rth object to be added is not one of the b objects defining the region, then the region was also present in the structure after the first $r-1$ objects had been added. (If each region is defined by exactly b objects, then the preceding sum is an exact expression for the expected number of regions.) As an example, we apply this observation to the case in which the objects are n points in the plane, the structure is their Delaunay triangulation, and the regions are Delaunay triangles. This fits well into the setup just discussed. Moreover, we know that $M(r)$ is $2r - h_r - 2$, where h_r is the expected number of vertices appearing on the convex hull of a random sample of r points of the given n. We thus conclude that the expected number of Delaunay triangles constructed during a randomized incremental algorithm is $\sum_{r=3}^{n} \frac{3}{r}(2r - h_r - 2)$. The same expression was recently derived in [20] by using a more involved analysis. This general framework has also been observed by Seidel [16]–[18] and by Mehlhorn [11] (see also [4]) and is referred to as "backwards analysis."

3.2. The expectation of the sum of weights. Recall that the weight of a trapezoid Δ, $w(\Delta)$, is defined as the number of line segments that intersect Δ. As in §3.1, we argue that for the purpose of proving an upper bound on the expectation of the sum of weights of all constructed trapezoids, it suffices to consider only nontransient trapezoids. To see this, let Δ be a transient trapezoid. Distribute its weight among all its parents in the dag so that the share of each parent does not exceed its original weight. This is possible because the union of the trapezoids of all parents of Δ contains Δ and therefore intersects at least as many segments as Δ does. Since any node in the dag has at most four children, its weight can thus go up by at most a factor of 5. Thus 5 times the expected sum of weights of all nontransient trapezoids is an upper bound on the expectation of the sum over all trapezoids.

LEMMA 3.3. *The expected sum of weights of all trapezoids constructed by the algorithm is $O(n\alpha(n)\log n)$.*

Proof. The expected sum of weights over all nontransient trapezoids constructed by the algorithm is equal to $\sum_\Delta w(\Delta)\mathbf{P}[Z_{n,\Delta}]$, where the sum is taken over all trapezoids Δ defined by at most four segments each, and $Z_{n,\Delta}$ is the event that, in the course of adding all line segments, Δ was constructed as a nontransient trapezoid, the same event as in Lemma 3.2. In addition to the events $Z_{r,\Delta}$ and $X_{r,\Delta}$ we define $Y_{r,\Delta}$: Δ is a trapezoid of f_r, and s_{r+1}, the line segment added next, is one of the $w(\Delta)$ segments that intersect Δ.

Note that if Δ is a trapezoid of f_r, then none of the segments intersecting Δ was chosen in the first r steps. So for $Y_{r,\Delta}$ to occur, given that $X_{r,\Delta}$ has occurred, we have to choose at the $(r+1)$th step one of these $w(\Delta)$ segments out of the remaining $n-r$ segments. Hence

$$\mathbf{P}[Y_{r,\Delta}] = \mathbf{P}[X_{r,\Delta}] \cdot \frac{w(\Delta)}{n-r}.$$

Observe also that

$$Y_{r,\Delta} \subseteq X_{r,\Delta} \cap \overline{X}_{r+1,\Delta},$$

and, in general, we may have proper inclusion, because Δ can be removed from the city also by a line segment that does not intersect Δ. Independent of whether proper or improper inclusion, this implies that

$$(2) \qquad \sum_\Delta \sum_{i=1}^{r} \mathbf{P}[Y_{i,\Delta}] \le \sum_\Delta \sum_{i=1}^{r} \mathbf{P}[X_{i,\Delta} \cap \overline{X}_{i+1,\Delta}] \le \sum_\Delta \mathbf{P}[Z_{r,\Delta}] = O(r\alpha(r)).$$

In other words, the expected number of trapezoids that become history during the first $r+1$ insertions is $O(r\alpha(r))$, which is clear because these trapezoids have to be constructed first, and the expected number of such trapezoids, over the course of the first r insertions, is $O(r\alpha(r))$, as shown in §3.1.

Now fix Δ, and recall from the proof of Lemma 3.2 that

$$\mathbf{P}[Z_{n,\Delta}] = \sum_{r=1}^{n} \mathbf{P}[\overline{X}_{r-1,\Delta} | X_{r,\Delta}] \cdot \mathbf{P}[X_{r,\Delta}] \le \sum_{r=1}^{n} \frac{4}{r}\mathbf{P}[X_{r,\Delta}].$$

This implies that

$$w(\Delta)\mathbf{P}[Z_{n,\Delta}] \le 4\sum_{r=1}^{n} \frac{n-r}{n-r} \cdot \frac{w(\Delta)}{r}\mathbf{P}[X_{r,\Delta}] = 4\sum_{r=1}^{n} \frac{n-r}{r}\mathbf{P}[Y_{r,\Delta}].$$

To simplify the notation we set $D_r = \sum_\Delta \mathbf{P}[Y_{r,\Delta}]$, and we can now write

$$(3) \qquad \sum_\Delta w(\Delta)\mathbf{P}[Z_{n,\Delta}] \le 4\sum_{r=1}^{n-1} \frac{n-r}{r} D_r = 4\sum_{r=1}^{n-1} \left(\frac{n-r}{r} - \frac{n-r-1}{r+1} \right) \sum_{i=1}^{r} D_i.$$

However, we have shown that

$$\sum_{i=1}^{r} D_i = \sum_\Delta \sum_{i=1}^{r} \mathbf{P}[Y_{i,\Delta}] = O(r\alpha(r)).$$

Hence we finally obtain

$$\sum_{\Delta} w(\Delta)\mathbf{P}[Z_{n,\Delta}] \leq 4\sum_{r=1}^{n-1} \frac{n}{r(r+1)} O(r\alpha(r)) = O(n\alpha(n)\log n), \tag{4}$$

as claimed. □

Remark. As in the remark at the end of §3.1, these calculations can also be extended to the more general setup discussed there. Specifically, if we denote by $S(r)$ the expected number of regions (trapezoids in our case) formed during the first r steps of the randomized process, then (1) implies that

$$S(r) \leq \sum_{j=1}^{r} \frac{b}{j} M(j)\,, \qquad r = 1, \ldots, n, \tag{5}$$

where b, M are as defined in the previous remark. The analysis leading to equations (2)–(4) can then be generalized to yield a bound on $T(n)$, which is defined to be the expectation of the sum of weights of the regions ever formed by the algorithm, where the weight of a region is the number of objects that intersect it. That is, we obtain

$$\begin{aligned} T(n) &\leq \sum_{r=1}^{n-1} \frac{bn}{r(r+1)} S(r) = \sum_{r=1}^{n-1} \frac{bn}{r(r+1)} \sum_{j=1}^{r} \frac{b}{j} M(j) \\ &= \sum_{j=1}^{n-1} \frac{b}{j} M(j) \sum_{r=j}^{n-1} \frac{bn}{r(r+1)} = \sum_{j=1}^{n-1} \frac{b^2 n M(j)}{j} \left(\frac{1}{j} - \frac{1}{n}\right), \end{aligned}$$

or

$$T(n) \leq \sum_{r=1}^{n} \frac{b^2(n-r)}{r^2} M(r). \tag{6}$$

4. Extensions. The technique presented in this paper is sufficiently general to be applicable to a variety of other related problems. In this section we present a few such applications. In §4.1 we extend the previous algorithm to compute the entire arrangement of n line segments, and in §4.2 we describe an algorithm for computing a single face in an arrangement of Jordan arcs. The overall strategy is similar to that described earlier, but there are certain additional technical details that are particular to the specific application. In each case we discuss in some detail these difficulties, the modifications to the algorithm that they require, and the analysis of the resulting modified algorithm.

4.1. Computing the entire arrangement of *n* line segments. We first consider a simple extension of our technique to the problem of calculating the entire arrangement of a collection of n line segments in the plane. This can be achieved by applying a simplified version of the technique of §2. In this case there is no need to distinguish between city and suburbs since every face of the arrangement needs to be constructed. Consequently, when a segment is added to the arrangement, all its portions are drawn and there is no need to maintain any face topology by means of a union-find structure. We leave it to the reader to work out the details of this modified and simplified algorithm. The analysis is also easy; it uses the general method described at the end of §3. In this case we have $b = 4$ (the maximum number of segments defining a trapezoid), and $M(r)$, the *expected* number of trapezoids forming the vertical decomposition of the arrangement of the first r segments that were inserted, is bounded by $O(r + K \cdot \frac{r^2}{n^2})$, where K is the total number of intersections between the given segments (see [5] for the simple proof of this bound). Hence the expected storage of the algorithm is

$$S(n) \le \sum_{r=1}^{n} \frac{4}{r} M(r) = O\left(\sum_{r=1}^{n} \left(1 + \frac{K}{n^2} r\right)\right) = O(n + K),$$

and the expected running time is

$$T(n) \le \sum_{r=1}^{n} \frac{16(n-r)M(r)}{r^2} = O\left(\sum_{r=1}^{n} \left(\frac{n-r}{r} + \frac{K(n-r)}{n^2}\right)\right) = O(n \log n + K).$$

We thus obtain an algorithm with optimal (expected) running time and storage. The same performance is achieved by the deterministic (but complicated) algorithm of [3] and by the alternative randomized algorithms of [5], [14] (a variant of the algorithm of [5] also achieves $O(n)$ working storage).

4.2. Computing a face in an arrangement of arcs. Let Γ be a collection of n Jordan arcs $\gamma_1, \dots, \gamma_n$. We assume that the arcs have a simple shape, which means that any pair of them intersect in at most some fixed number s of points, that each arc consists of a small fixed number of x-monotone pieces, and that it takes constant time to perform any of the following primitive operations — finding the intersection points between a pair of arcs, decomposing an arc into its x-monotone pieces, intersecting an arc with a vertical line, and testing whether a given point lies above or below a given (x-monotone piece of an) arc. To simplify the description of the algorithm, we assume that each arc is already x-monotone; otherwise, we first decompose the arcs into x-monotone pieces and then apply the algorithm. We also assume that the arcs are in general position, in the spirit of the similar assumption we have made for line segments.

As above, let p be a given point not lying on any arc. Our goal is to compute the face in the arrangement of Γ that contains p. To compute the desired face, we apply the same scheme of §2, except that there are several new technical difficulties that need to be addressed. The face (city) and its complement (suburbs) are represented by their vertical decomposition into *pseudotrapezoids*, obtained, as was done earlier, by drawing vertical segments up and down from every endpoint and intersection point until they hit another arc. If general position is assumed, each pseudotrapezoid is defined by at most four arcs, two containing its top and bottom edges and two defining its left and right sides.

The data structures that we use are the same as those in §2 — the dag, the linear array, and the union-find structure. Searching in the dag for the pseudotrapezoids that intersect a newly inserted arc γ is trickier in this case because the intersection of γ with a pseudotrapezoid Δ can consist of several connected components (at most $s + 1$ components, as is easily checked). In this case we expect the search through the dag to yield a partition of γ into a (sorted) list of subarcs, each of which is either contained in the suburbs (and is therefore not drawn at all) or intersects a single pseudotrapezoid of the current city. This list is initialized to consist only of γ itself and is refined during the search as follows. Any recursive step involves the processing of some history pseudotrapezoid Δ and some subarc γ' that is a connected component of $\gamma \cap \Delta$. We go over the (constant number of) children of Δ, and for each child Δ' we compute $\gamma' \cap \Delta'$. The constant number of resulting subarcs of γ' are sorted by x coordinate and replace γ' in the output list. The search now continues recursively at each of the new subarcs and at the pseudotrapezoid that contains it. Note that a node Δ of the dag may be visited several times during the search, each time with a different subarc γ'. However, it is easy to show that the cost of searching with γ in the dag is proportional to the number of pseudotrapezoids ever formed that are crossed by γ. At the end of the search we almost obtain the desired sorted partition of γ; since children pseudotrapezoids are merged, the final list of subarcs may contain pairs of adjacent subarcs that share an endpoint and are contained in the same pseudotrapezoid. An additional pass through the final list is needed to merge such pairs.

The remaining steps of the algorithm are the same (with certain trivial modifications) as those of the algorithm of §2. We leave it to the reader to fill in the details.

The analysis is also similar to that in §3. Using the general notations at the end of that section, we observe that in our case we have $b = 4$ (maximum number of arcs defining a pseudotrapezoid) and $M(r) = O(\lambda_{s+2}(r))$ (bound on the complexity of a single face in an arrangement of r arcs as above; see [9]). Then the expected storage of the algorithm is

$$S(n) \le \sum_{r=1}^{n} \frac{4}{r} M(r) = O(\lambda_{s+2}(n)),$$

and the expected running time, again dominated by the cost of the searches through the dag, is

$$T(n) = O\left(\sum_{r=1}^{n-1} \frac{n-r}{r^2} M(r)\right) = O(\lambda_{s+2}(n)\log n).$$

Hence we have the following theorem.

THEOREM 4.1. *Given a collection Γ of n arcs in the plane with the aforementioned properties and a point p not lying on any arc, the face of $\mathcal{A}(\Gamma)$ that contains p can be computed in randomized expected time $O(\lambda_{s+2}(n)\log n)$ and expected storage $O(\lambda_{s+2}(n))$.*

Remark. This result is an improvement of the previous (deterministic) algorithm of Guibas, Sharir, and Sifrony [9], whose running time is $O(\lambda_{s+2}(n)\log^2 n)$.

5. Discussion.

In this paper we have presented a randomized incremental technique for computing a single face in an arrangement of line segments and for several related problems. The technique is a variant of several related recent randomized algorithms. It improves the running time of the previously best algorithms for these problems, and it is fairly simple to implement. The main characteristic of the technique is maintaining the history of the random process as a dag of trapezoids, which facilitates efficient location of the new segment to be inserted relative to the current version of the computed face. The analysis of the algorithm is also novel, in the sense that it extends the previous analysis technique of Clarkson and Shor, resulting in a simpler and more general approach.

The problems studied in this paper are only a sample of problems that can be solved efficiently by using our technique. In addition to the earlier algorithm of Guibas, Knuth, and Sharir [8] for computing Delaunay triangulations in the plane, there appeared, after the original preparation of this paper, a few related works that also apply this or closely related techniques. Among these we mention work by Seidel [18] for constructing trapezoidal decompositions of arrangements of nonintersecting line segments and applying them for efficient point location and triangulation of simple polygons and work by Miller and Sharir [12] for computing the union of fat triangles or of pseudodiscs.

There are several other problems that are likely to be amenable to the technique presented here. Among these we mention the problems of computing many faces in an arrangement of lines or of line segments, computing a single cell in an arrangement of triangles in 3-space, computing the zone of a plane in an arrangement of planes in 3-space, and computing many cells in such an arrangement of planes. In all these cases it is straightforward to design the general structure of an appropriate algorithm, along the lines of the algorithms we have described. It is also fairly easy to extend the analysis to obtain sharp bounds on the expected number of regions constructed by the algorithm and on the expected sum of their weights, appropriately defined. The difficulty in completing the algorithm usually lies in the subproblem of maintaining the topology of the constructed structure. For example, in computing a single cell in an arrangement of triangles in space, when we add a new triangle t, we need to determine the way in which it modifies the current cell, which seems to be considerably more difficult than the similar problem in two dimensions.

To conclude, we mention one final open problem, namely, to close the still remaining gap between the expected running time of our main algorithm, i.e., $O(n\alpha(n)\log n)$, and the lower bound of $\Omega(n \log n)$.

Acknowledgment. The authors wish to express their gratitude for the generous support and hospitality of the DEC Palo Alto Systems Research Center.

REFERENCES

[1] P. Alevizos, J. D. Boissonnat, and F. P. Preparata, *An optimal algorithm for the boundary of a cell in a union of rays*, Algorithmica, 5 (1990), pp. 573–590.

[2] J. D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec, *On-line geometric algorithms with good expected behaviours*, in Proc. Journées Géometriques Algorithmiques, INRIA, Sophia-Antipolis, June 1990, pp. 7–13.

[3] B. Chazelle and H. Edelsbrunner, *An optimal algorithm for intersecting line segments in the plane*, J. Assoc. Comput. Mach., 39 (1992), pp. 1–54.

[4] L. P. Chew, *The Simplest Voronoi Diagram Algorithm Takes Linear Expected Time*, Manuscript, 1988.

[5] K. Clarkson and P. Shor, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[6] H. Edelsbrunner, L. Guibas, and M. Sharir, *The complexity and construction of many faces in arrangements of lines and of segments*, Discrete Comput. Geom., 5 (1990), pp. 161–196.

[7] H. Edelsbrunner and E. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graphics, 9 (1990), pp. 66–104.

[8] L. Guibas, D. E. Knuth, and M. Sharir, *Randomized incremental construction of Voronoi and Delaunay diagrams*, Algorithmica, 7 (1992), pp. 381–413.

[9] L. Guibas, M. Sharir, and S. Sifrony, *On the general motion planning problem with two degrees of freedom*, Discrete Comput. Geom., 4 (1989), pp. 491–521.

[10] J. Hershberger, *Finding the upper envelope of n line segments in $O(n \log n)$ time*, Inform. Process. Lett., 33 (1989), pp. 169–174.

[11] K. Mehlhorn, Unpublished manuscript, 1990.

[12] N. Miller and M. Sharir, *Efficient Randomized Algorithms for Constructing the Union of Fat Triangles and of Pseudodiscs*, Manuscript, 1991.

[13] J. S. B. Mitchell, *On Computing a Single Face in an Arrangement of Line Segments*, Manuscript, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, July 1990.

[14] K. Mulmuley, *A fast planar partition algorithm* I, J. Symbolic Comput., 10 (1990), pp. 253–280.

[15] R. Pollack, M. Sharir, and S. Sifrony, *Separating two simple polygons by a sequence of translations*, Discrete Comput. Geom., 3 (1988), pp. 123–136.

[16] R. Seidel, *Small dimensional linear programming and convex hulls made easy*, Discrete Comput. Geom., 6 (1991), pp. 423–434.

[17] ———, *Backwards analysis of randomized geometric algorithms*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., Springer-Verlag, Berlin, 1993, pp. 37–67.

[18] ———, *A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons*, Comput. Geom. Theory Appl., 1 (1991), pp. 51–64.

[19] A. Wiernik and M. Sharir, *Planar realization of nonlinear Davenport Schinzel sequences by segments*, Discrete Comput. Geom., 3 (1988), pp. 15–47.

[20] E. Yaniv. *Randomized Incremental Construction of Delaunay Triangulations: Theory and Practice*, M.Sc. thesis, Tel Aviv University, Tel Aviv, Israel, 1991.