ELSEVIER

Computational Geometry 10 (1998) 89–104

Computational Geometry
Theory and Applications

Finding constrained and weighted Voronoi diagrams in the plane ☆

Cao An Wang [a,*], Yung H. Tsin [b]

[a] *Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7*
[b] *School of Computer Science, University of Windsor, Windsor, Ontario, Canada N9B 3P4*

Communicated by J. Urrutia; submitted 2 July 1996; accepted 14 July 1997

Abstract

The Voronoi diagram of a set of weighted points (sites) whose visibilities are constrained by a set of line segments (obstacles) on the plane is studied. The diagram is called *constrained* and *weighted* Voronoi diagram. When all the sites are of the same weight, it becomes the *constrained* Voronoi diagram in which the endpoints of the obstacles *need not* be sites. An $\Omega(m^2n^2)$ lower bound on the combinatorial complexity of both constrained Voronoi diagram and constrained and weighted Voronoi diagram is established, where n is the number of sites and m is the number of obstacles. For constrained Voronoi diagram, an $O(m^2n^2+n^4)$ time and space algorithm is presented. The algorithm is optimal when $m \geqslant cn$, for any positive constant c. For constrained and weighted Voronoi diagram, an $O(m^2n^2+n^4 2^{\alpha(n)})$ time and $O(m^2n^2+n^4)$ space algorithm (where $\alpha(n)$ is the functional inverse of the Ackermann's function) is presented. The algorithm is near-optimal when $m \geqslant cn$, for any positive constant c. © 1998 Elsevier Science B.V.

Keywords: Arrangement; Constrained Voronoi diagram; Visibility; Weighted Voronoi diagram

1. Introduction

The Voronoi diagram is an important geometric structure in computational geometry which has attracted a lot of attention [1]. Given a set of points (called sites) S in the plane, the *standard* Voronoi diagram of S consists of a set of Voronoi cells, $\{V(s_i) \mid s_i \in S\}$, such that for any point $x \in V(s_i)$, $d(x,s_i) \leqslant d(x,s_j)$ for all $s_j \in S$, where $d(x,y)$ denotes the Euclidean distance between x and y.

Many variations of the standard Voronoi diagram have been investigated. One of them is to consider the diagram in the presence of obstacles in the plane. When the distances in such a Voronoi diagram are measured by geodesic [8,10], the diagram is called *geodesic Voronoi diagram*; when the distances

☆ Research supported by the Natural Sciences and Engineering Research Council of Canada under Grants NSERC-OPG0041629 and NSERC-A7811.

* Corresponding author.

PII S0925-7721(97)00028-X

are measured by visible straight-line segments, the diagram is called *constrained* (or *bounded*) *Voronoi diagram* [3,4,7,9,11]. Another variation is to assign a weight to each site of the standard Voronoi diagram. This diagram is called *weighted Voronoi diagram* [2]. In this paper, we consider the *constrained* and *weighted* Voronoi diagram which is the Voronoi diagram of a set of weighted sites restricted by a set of line segments (obstacles) in the plane. In contrast with the diagrams studied in [4,7,9,11], our Voronoi diagram does not require that every obstacle endpoint be a site (the Peeper's Voronoi diagram [3] also does not make such requirement). This drastically increases the complexity of the diagram, resulting in superlinear combinatorial complexity in its structures.

Practical applications of weighted Voronoi diagram were reported in [2], which cited several other papers. The disciplines of applications include economy, geography, communications, and biology. In the previous works, the plane is assumed to be without restrictions. In reality, however, the plane usually contains obstacles. For instance, in modeling a set of microwave or laser transmitters with varying strength, it is desirable to determine the regions in which a certain transmitter (if any) received best among the others. If the area contains buildings and mountains, then the buildings and mountains can be regarded as obstacles because the waves of a transmitter can be blocked by them. Since our Voronoi diagram takes obstacles into consideration, it is therefore a better geometric model.

The existence of obstacles makes a simple divide-and-conquer method and a straight-forward sweep-line method for constructing the constrained and weighted Voronoi diagram inefficient. We observe that there exists a close relationship between the diagram and the arrangement of a set of lines, where the lines are completely determined by the sites and obstacles. By constructing a visible-site list and a closest-site list for each edge in the arrangement, we are able to construct the diagram efficiently.

This paper is organized as follows. For clarity, we first consider the diagram of equi-weighted sites (i.e., constrained Voronoi diagram) in Sections 3 and 4. Section 3 gives an $\Omega(m^2n^2)$ worst-case lower bound on the combinatorial complexity of constrained Voronoi diagrams of n sites and m obstacles. Section 4 describes an algorithm for constructing the constrained Voronoi diagram. The algorithm is worst-case optimal in both time and space when $m \geqslant cn$, where c is a constant. In Section 5, the diagram for sites with different weighs is considered.

2. Preliminaries

First, we give several definitions and outline some properties of constrained and weighted Voronoi diagram.

Definition 1. Let O be a set of disjoint line segments representing obstacles. Two arbitrary points x and y in the plane are *visible* to each other in the presence of O iff the open line segment joining them does not intersect any obstacle in its interior unless they are collinear.

Definition 2. Let s be a site in the plane and $w(s)$ be the weight of s. The *distance* of s to an arbitrary point x, denoted by $d_w(x,s)$, is defined as $d(x,s)/w(s)$, where $d(x,s)$ is the Euclidean distance between s and x. The bisector of two sites s and s' is the locus of all points x satisfying $d_w(x,s) = d_w(x,s')$.

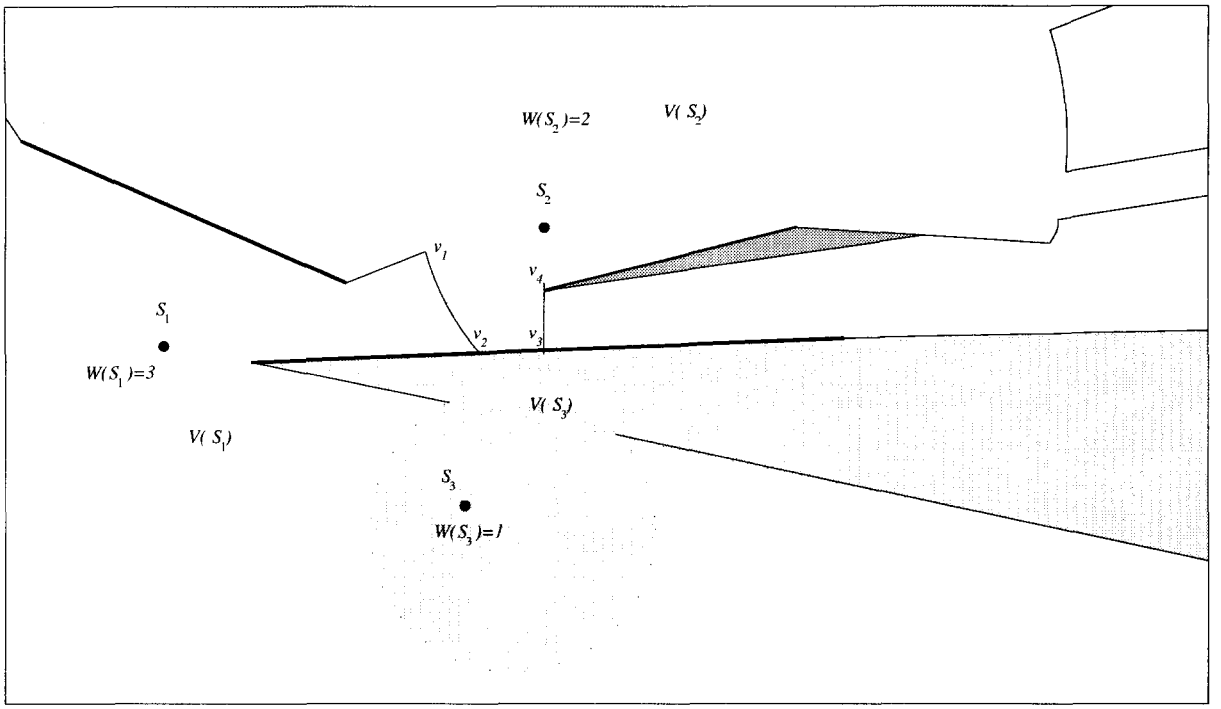

Fig. 1. A constrained and weighted Voronoi diagram.

Note that if s and s' are two sites of different weights, their bisector is a circle with center $(w^2(s)s - w^2(s')s')/(w^2(s) - w^2(s'))$, and radius $w(s)w(s')d(s,s')/(w^2(s) - w^2(s'))$ [2]. If they are of the same weight, then their bisector is the straight line bisecting the line segment joining them.

Definition 3. Let s be a site in the plane with weight $w(s)$ in the presence of obstacle O, the *visibility distance* between s and an arbitrary point x is defined as

$$d_{wo}(x,s) = \begin{cases} d_w(x,s), & \text{if } x \text{ and } s \text{ are visible to each other,} \\ \infty, & \text{otherwise.} \end{cases}$$

Definition 4. Let S be a set of weighted sites and O be a set of obstacles in the plane. The *constrained* and *weighted* Voronoi diagram, denoted by $\mathrm{CWV}(S,O)$, is a set of Voronoi cells $\{V(s_i) \mid s_i \in S\}$ such that $V(s_i) = \{x \in \mathbb{R}^2 \mid d_{wo}(x,s_i) \leqslant d_{wo}(x,s_j)$ and $d_{wo}(x,s_i) \neq \infty$, for all $s_j \in S$, $s_i \neq s_j\}$. A *Voronoi edge* is a maximal straight line segment or circular arc on the boundary of a Voronoi cell. The endpoints of the Voronoi edges are called *Voronoi vertices*.

An example of constrained and weighted Voronoi diagram of three sites and three obstacles is shown in Fig. 1. Clearly, if the weights of the sites in S are all the same, then $\mathrm{CWV}(S,O)$ becomes constrained Voronoi diagram $\mathrm{CV}(S,O)$, and if $O = \emptyset$, $\mathrm{CWV}(S,O)$ becomes weighted Voronoi diagram $\mathrm{WV}(S)$. Constrained and weighted Voronoi diagram have some special properties which do not normally exist in the other types of Voronoi diagrams. The most notable ones are the following.

(i) Every Voronoi edge is a section of the bisector of two sites ($\overline{v_1v_2}$ in Fig. 1), or a section of a line determined by a site and an endpoint of an obstacle ($\overline{v_3v_4}$ in Fig. 1), or a section of an obstacle ($\overline{v_2v_3}$ in Fig. 1). Note that the last two types of Voronoi edges do not exist in standard or weighted Voronoi diagrams and the last type of Voronoi edges does not exist in geodesic Voronoi diagrams and bounded Voronoi diagrams, although they do exist in the Peeper's Voronoi diagram [3].
(ii) The boundary of a Voronoi cell in CWV(S) consists of circular arcs and/or line segments.
(iii) A Voronoi cell may consist of several disjoint sub-cells. In Fig. 1, $V(s_1)$ consists of three disjoint sub-cells.
(iv) A Voronoi diagram may not cover the entire plane, i.e., some regions in the plane may not belong to any Voronoi cell due to the blockade of obstacles. These regions are called *blank* regions. In Fig. 1, the heavily shaded region is a blank region.

In the following two sections, we assume that the sites are equi-weighted, i.e., we consider constrained Voronoi diagram.

3. A lower bound for constrained Voronoi diagram

We shall show that the number of disjoint Voronoi sub-cells for constrained Voronoi diagrams is $\Omega((mn)^2)$. The time and space lower bounds are then immediate. The following notations are used in the discussion. X and Y represent respectively the x-axis and the y-axis of an xy-orthogonal coordinate system. $X^+ = \{(x,y) \in \mathbb{R}^2 \mid x \geqslant 0\}$ and $X^- = \{(x,y) \in \mathbb{R}^2 \mid x < 0\}$ (Y^+ and Y^- are defined similarly). $X^+(x_0,y_0) = \{(x,y) \in \mathbb{R}^2 \mid y = y_0 \wedge x \geqslant x_0\}$ (respectively $X^-(x_0,y_0) = \{(x,y) \in \mathbb{R}^2 \mid y = y_0 \wedge x < x_0\}$) represents a ray on the xy-plane starting at point (x_0,y_0), parallel to X and extends towards the positive (respectively negative) end of X. $Y^+(x_0,y_0)$ and $Y^-(x_0,y_0)$ are defined similarly.

Theorem 1. *Let S be a set of n sites and O be a set of m obstacles. The combinatorial complexity of* CV(S,O) *is $\Omega((mn)^2)$ in the worst case.*

Proof. (Refer to Fig. 2.) Let the m obstacles be of the same length, say d' and $d' = d - \varepsilon$ for some constants d and ε such that $\varepsilon \ll d$. Assume without loss of generality that m and n are even numbers. Divide both S and O into two groups of equal size. Place the first group of $m/2$ obstacles $\{o_1, o_2, \ldots, o_{m/2}\}$ on ray $X^+(0,0)$ such that o_i, $1 \leqslant i \leqslant m/2$, is placed between the two points $(i-1)d$ and $(i-1)d + d'$. Clearly, every two consecutive obstacles are at a distance ε apart. Place the second group of $m/2$ obstacles $\{o_{m/2+1}, o_{m/2+2}, \ldots, o_m\}$ on ray $Y^+(0,0)$ in a similar manner. Place the first group of $n/2$ sites $\{s_1, s_2, \ldots, s_{n/2}\}$ on ray $X^+(0,-m^2d)$ such that $s_i = ((i-1)2d/(n-2), -m^2d)$, $1 \leqslant i \leqslant n/2$. Place the second group of $n/2$ sites $\{s_{n/2+1}, s_{n/2+2}, \ldots, s_n\}$ on ray $Y^+(-2m^2d, 0)$ starting at $(-2m^2d, 0)$ in the same manner.

Let 2D-box be the quadrilateral determined by r_1, r_2, $X^+(0,0)$ and $Y^+(0,0)$, where r_1 is the line joining s_1 and the rightmost endpoint of $e_{m/2}$, and r_2 is the line joining $s_{n/2+1}$ and the upper endpoint of e_m. Based on the above construction, it is easily verified that the region inside the 2D-box visible from a site on ray $X^+(0,-m^2d)$ consists of $m/2 - 1$ distinct strips. Therefore, the $n/2$ sites on ray $X^+(0,-m^2d)$ determine $(m-2)n/4$ distinct strips inside the 2D-box. Similarly, the $n/2$ sites on ray $Y^+(-2m^2d,0)$ determine $(m-2)n/4$ distinct strips inside the 2D-box. When ε is sufficiently small, the first group of strips divides the second group into $\Theta((mn)^2)$ disjoint substrips such that any point

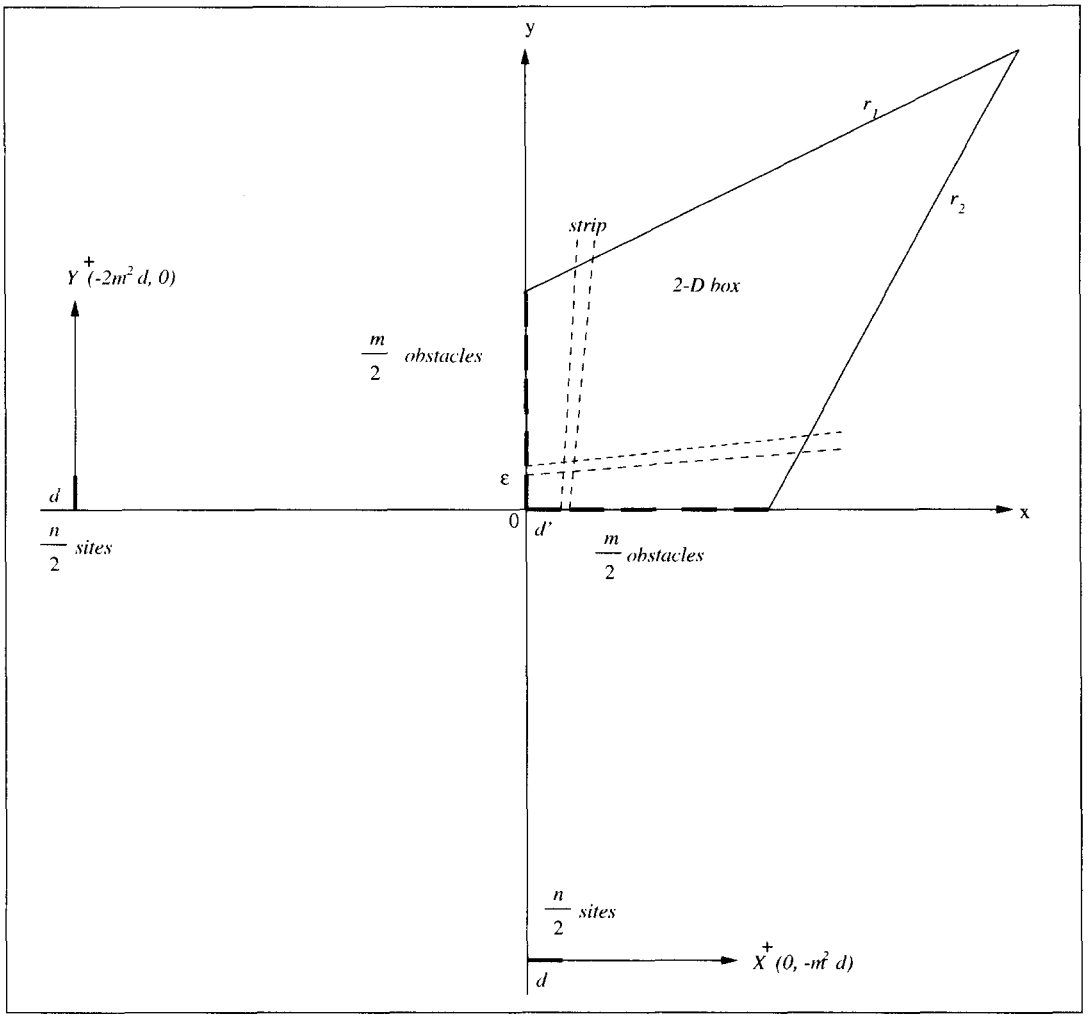

Fig. 2. A worst-case lower bound.

in these substrips is visible from some sites on ray $Y^+(-2m^2d, 0)$ and is *not* visible from any site on ray $X^+(0, -m^2d)$. Moreover, each of these substrips is bounded by four quadrilaterals, two of which are '*blank*' regions while the remaining two are intersections of the two groups of strips. By the positions of the sites and the obstacles, it is easily verified that any point in these intersections is closer to some sites on ray $X^+(0, -m^2d)$ than to any site on ray $Y^+(-2m^2d, 0)$. Hence, each of the $\Theta((mn)^2)$ substrips is a Voronoi sub-cell associated with a site on $Y^+(-2m^2d, 0)$. □

4. Constructing the constrained Voronoi diagram

The algorithm presented in this section uses the construction, called *arrangement*, of Edelsbrunner et al. [6]. We refer the reader to [6] for its definition.

Definition 5. A straight line is called the *base line* of a line segment (or a ray) if it contains the line segment (or the ray). An *obstacle line* is the base line of an obstacle. A *visibility line* is a straight line determined by a site and an endpoint of an obstacle.

We shall denote the set of all obstacle lines by O', the set of all bisectors by B and the set of all visibility lines by T. We further let $E = O' \cup B \cup T$ and $A(E)$ be the arrangement of E. Note that $|O'| = m$, $|B| = n(n-1)/2$, $|T| = 2mn$ and $|E| = \mathrm{O}(mn + n^2)$.

Lemma 1. *The number of Voronoi edges in* $\mathrm{CV}(S, O)$ *is* $\mathrm{O}((mn)^2 + n^4)$.

Proof. Every edge of $\mathrm{CV}(S, O)$ is either an edge of $A(E)$ or the union of some edges of $A(E)$. Since $|E| = \mathrm{O}(mn + n^2)$, by Lemma 2.5 of Edelsbrunner et al. [6, p. 344], the number of edges and vertices in $A(E)$, and hence in $\mathrm{CV}(S, O)$, is $\mathrm{O}((mn + n^2)^2) = \mathrm{O}((mn)^2 + n^4)$. □

Since $A(E)$ embeds $\mathrm{CV}(S, O)$, we may construct $\mathrm{CV}(S, O)$ by first constructing $A(E)$ and then deleting all those edges in $A(E)$ which are not in $\mathrm{CV}(S, O)$. The algorithm presented below is based on this idea. For the ease of description, we make the following assumptions.

Assumption 1. *Every obstacle line, bisector and visibility line is distinct (consequently, no two lines in* E *coincide).*

Assumption 2. *The obstacles are non-intersecting.*

Assumption 1 can be removed by including $\mathrm{O}((mn)^2 + n^4)$ testing steps as $|E| = \mathrm{O}(mn + n^2)$. Assumption 2 can also be removed without affecting the time and space bounds.

Definition 6. The *closest-site list* of an edge e in $A(E)$, denoted by CA_e, is a list of the n sites sorted by their Euclidean distances from a fixed point on e in ascending order, disregarding the obstacles.

From the fact that no bisector intersects the interior of e, it is easily verified the following.

Property 1. CA_e *is independent of the interior point used in calculating the* n *distances.*

From the fact that no visibility line intersects the interior of any edge in $A(E)$, we have Property 2.

Property 2. *Each edge of* $A(E)$ *is an 'atomic' element in the sense that it is* completely *visible or invisible from a site.*

Hence, it makes sense to talk about the visible sites of an edge.

Definition 7. A site s is a *closest-visible site* of an edge e if s is the first 'visible' site in CA_e.

Lemma 2. *Let* e *be any edge in* $A(E)$. *Then* e *does not lie on a Voronoi edge in* $\mathrm{CV}(S, O)$ *iff* e *has no closest-visible site or* e *is not on the boundaries of the Voronoi cells of its closest-visible sites.*

Proof. Trivial. □

Algorithm Find-CV(S, O)
Step 1. Construct E.
Step 2. Construct the arrangement $A(E)$.
Step 3. (Delete all those edges in $A(E)$ not belonging to CV(S, O).)
(a) For each edge e of $A(E)$, find the sites visible to e (visible-site list).
(b) For each edge e of $A(E)$, order the sites according to their Euclidean distances from e (closest-site list);
(c) For each edge e of $A(E)$, determine the closest-visible sites of e. If no such site exist, then delete e.
(d) For each undeleted edge e, determine if e appears on the boundaries of the Voronoi cells of its closest-visible sites. Delete e if it does not.
Step 4. Construct CV(S, O) from the edges remaining in $A(E)$.

The correctness of algorithm Find-CV(S, O) follows from Lemma 2.

Details of the steps

Step 1 is straight-forward and can be done in $O(mn + n^2)$ time and space.

Step 2 can be done in $O((mn)^2 + n^4)$ time and space [6]. The data structure used in representing the arrangement $A(E)$ is the conventional incidence graph. Additional information are associated with each node as described later.

The implementation of Step 3 is most complicated. For ease of description, we shall first present a non-optimal algorithm and then reduce its time and space to the desired optimal bounds.

In Step 3(a), the list of all visible sites for every edge in $A(E)$ is determined. A brute-force method requires testing the visibility of each edge against every site of S. This results in an $O(m^3n^3 + mn^5)$ time algorithm. The observations made in Lemma 3 below support a faster method.

Definition 8. Let e and e'' be two edges in arrangement $A(E)$ which share an endpoint v. e'' is said to be the (*clockwise*) *successor* of e if e'' is the first edge that e encounters when e rotates around v clockwise. If the base line l of e'' (e, respectively) is a visibility line determined by site s and obstacle o, then o is *up* (*down*, respectively) if o and e (e'', respectively) are on the opposite sides of l. o is *down* (*up*, respectively) otherwise. Let l be a visibility line determined by site s and endpoint p of an obstacle. An edge e on l is *unmasked* if e is visible from s and is on that half-line on l which is originated from p and does not contain s. Edge e is *masked*, otherwise.

Lemma 3. *Let* vis(e) *denote the set of all sites visible from edge* e. *Let* e *and* e'' *be two edges in* $A(E)$ *such that* e *and* e'' *share an endpoint* v *and* e'' *is the successor of* e. *Let* l (*respectively* l') *be the base line of* e (*respectively* e'') *and in cases where* l (*respectively* l') *is a visibility line,* l (*respectively* l') *is determined by obstacle* o *and site* s (o' *and* s', *respectively*).
(1) *Suppose that none of* l *and* l' *is a visibility line, then* vis$(e'') =$ vis(e).
(2) *Suppose that exactly one of* l *and* l' *is a visibility line. Then,*
(i) *if* l *is a visibility line and* o *is up, then* vis$(e'') =$ vis$(e) - \{s\}$ *if* e *is unmasked;* vis$(e'') =$ vis(e) *if* e *is masked*;
(ii) *if* l' *is a visibility line and* o' *is down, then* vis$(e'') =$ vis$(e) \cup \{s'\}$ *if* e'' *is unmasked*; vis$(e'') =$ vis(e) *if* e'' *is masked*;
(iii) *in the remaining cases,* vis$(e'') =$ vis(e).

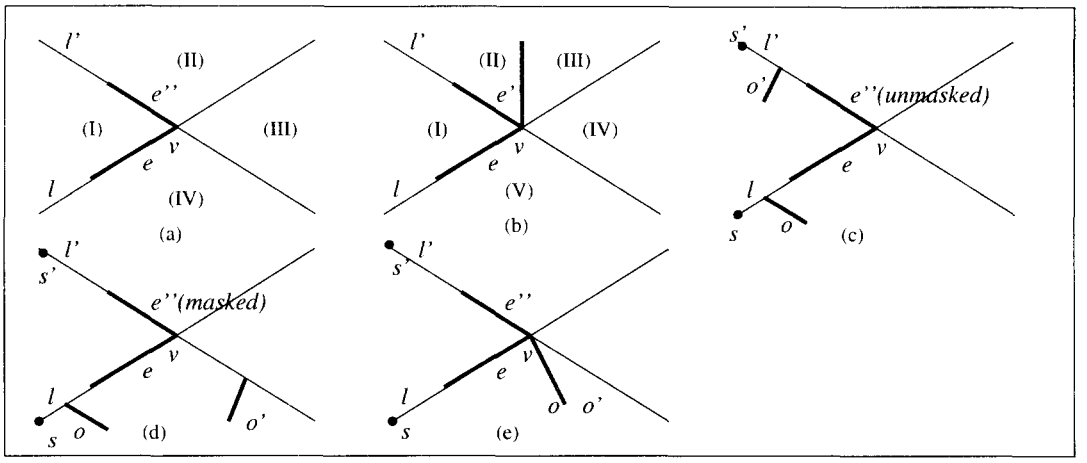

Fig. 3. Some illustrations of Lemma 3.

(3) *Suppose that both l and l' are visibility lines. Then,*
   (i) *if both o and o' are down, then* $\mathrm{vis}(e'') = \mathrm{vis}(e) \cup \{s'\}$ *if e'' is unmasked;* $\mathrm{vis}(e'') = \mathrm{vis}(e)$ *if e'' is masked;*
   (ii) *if o is up and o' is down, then* $\mathrm{vis}(e'') = \mathrm{vis}(e) - \{s\}$ *if e'' is masked and e is unmasked;* $\mathrm{vis}(e'') = \mathrm{vis}(e) \cup \{s'\}$ *if e'' is unmasked and e is masked;* $\mathrm{vis}(e'') = (\mathrm{vis}(e) - \{s\}) \cup \{s'\}$ *if e'' and e are unmasked;* $\mathrm{vis}(e'') = \mathrm{vis}(e)$ *if both e and e'' are masked;*
   (iii) *if o is down and o' is up, then* $\mathrm{vis}(e'') = \mathrm{vis}(e)$;
   (iv) *if both o and o' are up, then* $\mathrm{vis}(e'') = \mathrm{vis}(e) - \{s\}$ *if e is unmasked;* $\mathrm{vis}(e'') = \mathrm{vis}(e)$ *if e is masked.*

Proof. (1) Suppose that v is not on any obstacle, then l and l' divide the plane into four regions (Fig. 3(a)). Every site in region (II) or region (IV) is either visible or invisible to both e and e''. For each site s in region (I) or region (III), since e'' is the successor of e, no visibility line created by s can pass through v. Again, s is either visible or invisible to both e and e''. Thus, $\mathrm{vis}(e'') = \mathrm{vis}(e)$. A similar analysis can be applied to the case where v lies in the interior of an obstacle. For the case where v is an endpoint of an obstacle (Fig. 3(b)), the base lines and the obstacle divide the plane into five regions. Every site in region (II) or region (V) is either visible or invisible to both e and e''. Every site in region (III) is invisible to both e and e'' due to the blockade of the obstacle. There is no site in region (I) or (IV) as e'' is the successor of e. Thus, $\mathrm{vis}(e'') = \mathrm{vis}(e)$.

(3)(i) When v is not on any obstacle. Since e'' is the successor of e and o is down, regardless of whether e is masked or unmasked, s is either visible or invisible to both e and e''. Moreover, s' is visible to e'' and not to e if e'' is unmasked (Fig. 3(c)) and is visible or invisible to both e and e'' if e'' is masked (Fig. 3(d)). For every site other than s and s', the site is either visible or invisible to both e and e''. Hence, $\mathrm{vis}(e'') = \mathrm{vis}(e) \cup \{s'\}$ if e'' is unmasked and $\mathrm{vis}(e'') = \mathrm{vis}(e)$ if e'' is masked. When v lies in the interior of an obstacle, the above analysis can be applied similarly. When v is an endpoint of an obstacle, o and o' must coincide with that obstacle due to Assumption 1 (Fig. 3(e)). The rest of the analysis is similar to the above one.

Subcases (ii)–(iv) of case (3) and case (2) can be analyzed in a similar way. □

For each edge e in E, we represent vis(e) by an n-bit bucket, VB_e, such that the kth bit of VB_e is clear iff e is visible from site s_k. Moreover, if e is on an obstacle, then e is considered as having two sides each of which is associated with a distinct bucket. In Lemma 3, if e (e'', respectively) is on an obstacle, then vis(e) (vis(e''), respectively) refers to the VB bucket of that side of e (e'', respectively) which e'' (e, respectively) encounters when e rotates around v clockwise. Since the successor of e can be determined in $\mathrm{O}(1)$ time using the data structure of arrangement $A(E)$, Lemma 3 implies that given vis(e) for any edge e, vis(e''), where e'' is the successor of e, can be determined in $\mathrm{O}(1)$ time.

To construct the VB buckets, we first preprocess the visibility lines to *mark* or to *mask* out some of the edges. Let l be a visibility line determined by site s and endpoint p of an obstacle. Traverse l to locate the edges forming the line segment $\overline{sp}$ and check if any obstacle crosses $\overline{sp}$. If a crossing is detected, then the entire l is *masked*. Otherwise, the edge on $\overline{sp}$ incident upon p is *marked*. Moreover, let p' be the point on ray $\overrightarrow{sp}$ which is nearest to p and at which an obstacle crosses the ray. Then all those edges of l which are not on $\overline{pp'}$ are *masked*. Note that when p' does not exist, $\overline{pp'}$ is the ray on $\overrightarrow{sp}$ with end-point p.

Next, we construct the VB buckets for all the edges on or incident upon the obstacles. Let o be an obstacle with endpoints p and q. We shall call the halfplane to the left (right) of the ray $\overrightarrow{qp}$ the *left halfplane of* (*right halfplane*) o. For every edge e on o, the *leftside* (*rightside*) of e is that side of e facing the left halfplane (right halfplane) of o. Let e be the edge on o that has p as an endpoint. We begin with computing the VB bucket of the leftside of e, denoted by $\mathrm{VB}_e(L)$. Clearly, $\mathrm{VB}_e(L)$ consists of all those sites which lie in the left halfplane of o and are visible to e. Since each of those sites contributes a *marked* edge incident upon p, $\mathrm{VB}_e(L)$ can be easily determined. Once $\mathrm{VB}_e(L)$ is determined, we proceed around p clockwise to determine the VB buckets of all the edges incident upon p by using Lemma 3. When all these buckets are determined, the endpoint p is *marked*. Clearly, the VB bucket that is determined last is that of the rightside of e. Let r be the other endpoint of e. We proceed around r clockwise to determine the VB bucket of every edge incident upon r and lies in the right halfplane of o until an edge on o is encountered. Endpoint r is then marked. This process is repeated around o until we are back to p. Obviously, by that time, the VB buckets for all the edges on or incident upon o are determined.

Now, for each line $l \in E$, we traverse l, starting at an extreme edge f. First, if VB_f has not been determined, we determine it by testing, for each site, if an obstacle blocks the view of the site from f. In general, let p be the endpoint of the current edge which has not been examined. If p has been marked, then the VB buckets of all the edges incident upon p have been determined. So, we proceed immediately to the following edge on l. If p is unmarked, We compute the VB buckets for all the other edges incident upon p by using Lemma 3; mark p and then proceed to the following edge. In the case where p does not exist, we have reached the other end of l and we are done with l.

Lemma 4. *The VB buckets of the edges in A(E) can be determined in* $\mathrm{O}(((mn)^2 + n^4)n)$ *time and space.*

Proof. Preprocessing each visibility line l takes $\mathrm{O}(mn + n^2)$ time as there are $\mathrm{O}(mn + n^2)$ lines intersecting l creating at most that many edges on l and examining each such line and edge takes $\mathrm{O}(1)$ time. Preprocessing all the visibility lines thus takes $\mathrm{O}((mn + n^2)mn) = \mathrm{O}(m^2n^2 + n^4)$ time. The maximum number of edges incident upon endpoint p of obstacle o is $\mathrm{O}(m + n^2)$. Using the *marked* edges determined in the preprocessing step, and the structure of arrangement $A(E)$, $\mathrm{VB}_e(L)$

can be determined in $O(m + n^2)$ time. Therefore, determining the $VB_e(L)$ buckets for all the m obstacles takes $O(m^2 + mn^2)$. Determining the VB bucket for the extreme edge f of l can be done in $O(mn)$ time as there are mn site-obstacle pairs and testing each such pair could be done in $O(1)$ time. Since $|E| = O(mn + n^2)$, the total time required to compute the VB buckets for all those extreme edges is $O(m^2n^2 + n^4)$. For the remaining edges, as each of their VB buckets can be determined by copying the VB buckets of the (clockwise) predecessors and making a $O(1)$ time modification, it takes $O(n)$ time to determine the VB bucket for each of them. Let V be the set of all the vertices in $A(E)$. The total time spent on determining the VB buckets of all the edges in $A(E)$ is thus $O((m^2n^2 + n^4) + (m^2 + mn^2) + (\sum_{p \in V} \deg(p))n) = O((m^2n^2 + n^4)n)$, where $\deg(p)$ is the degree of p in $A(E)$. The space usage is obviously $O(m^2n^2 + n^4)$ n-bit buckets giving rise to the $O(((mn)^2 + n^4)n)$ space bound. □

Lemma 5. *The closest-site list,* CA_e*, for the edges in* $A(E)$ *can be determined in* $O(((mn)^2 + n^4)n)$ *time and space.*

Proof. Consider any line $l \in E$. l is divided by the other lines in E into a sequence of edges $(e_1, \ldots, e_k)$, where $k \leqslant O(mn + n^2)$. To compute CA_{e_i}, $1 \leqslant i \leqslant k$, we first determine CA_{e_1}. To do so, we compute the distances from an arbitrary point of e_1 to the sites (Property 1) and then sort the sites by these distances in ascending order. The resulting list is CA_{e_1}. From CA_{e_1}, we create an array of n pointers, $ptr_{e_1}[1..n]$, such that $ptr_{e_1}[i]$ points to the position of site s_i in CA_{e_1}. In general, $ptr_e[i] = j$ iff site s_i is the jth element in CA_e where e is the edge being examined. Clearly, this process takes $O(n \log n)$ time. To determine CA_{e_i} for $i > 1$, we traverse l, edge by edge, starting at e_1. Suppose $CA_{e_{i-1}}$ has just been determined. Let p be the common endpoint of e_{i-1} and e_i. We copy $CA_{e_{i-1}}$ into CA_{e_i} and $ptr_{e_{i-1}}$ into ptr_{e_i} and then examine all the lines passing through p. For each of the lines which is a bisector, we swap the pair of sites determining that bisector in CA_{e_i}. This could be done in $O(1)$ time with the help of ptr_{e_i}. We also swap the two corresponding pointers in ptr_{e_i}. Lines which are not bisectors are ignored. When all the lines passing through p are examined, CA_{e_i} is determined. Since copying each CA and ptr takes $O(n)$ time and swapping a pair of sites takes $O(1)$ time, it takes $O(n)$ time to process each line intersecting l. As there are $O(mn + n^2)$ lines in E intersecting l, the time required to determine the CA lists for all the edges on l is thus $O(n \log n + (mn + n^2)n)$. The total time required to determine the CA lists of all the edges in $A(E)$ is thus $O(((mn)^2 + n^4)n)$. The space bound is obviously $O(((mn)^2 + n^4)n)$. □

The VB buckets and CA lists of some of the edges on a line are shown in Fig. 4.

Lemma 6. *The closest-visible sites for the edges in* $A(E)$ *can be determined in* $O(((mn)^2 + n^4)n)$ *time and space.*

Proof. By scanning CA_e and examining the VB_e bucket for each entry encountered, the closest-visible sites of e can be determined in $O(n)$ time (note: when e is on an obstacle, the closest-visible sites of each of its two sides are determined separately). Hence, the closest-visible sites of the edges of $A(E)$ can be determined in $O((m^2n^2 + n^4)n)$ time and space. □

In determining whether an *undeleted* edge belongs to the boundary of the Voronoi cell of its closest-visible site in Step 3(d), we use the following lemma.

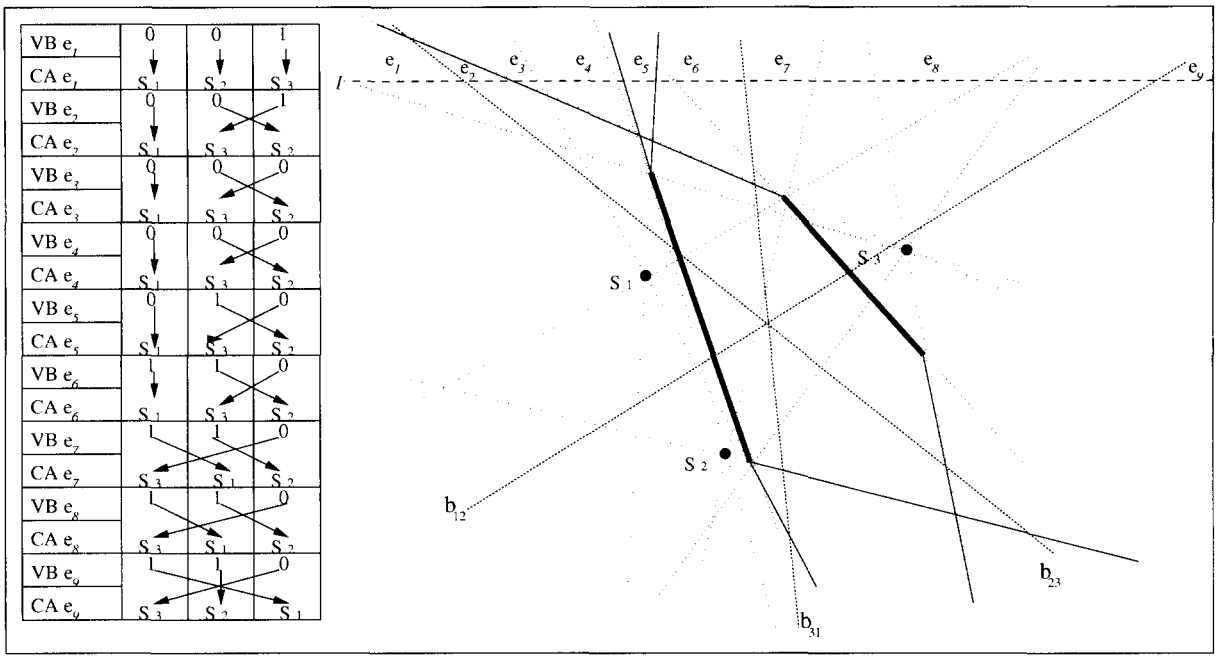

Fig. 4. The VB buckets and CA lists for some edges in $\mathrm{CV}(S, O)$.

Lemma 7. *Let e be an edge of $A(E)$ and s be its closest-visible site. Then e is on the boundary of the Voronoi cell $V(s)$ iff*
(1) *e is on an obstacle; or*
(2) *e is on a bisector and has two closest-visible sites; or*
(3) *e is an unmasked edge on a visibility line determined by s.*

Proof. Trivial. □

Lemma 8. *Determining the edges of $A(E)$ that lies on the Voronoi edges of $\mathrm{CV}(S, O)$ takes $\mathrm{O}((mn)^2 + n^4)$ time and space.*

Proof. Clearly, determining into which of the three cases an edge falls takes $\mathrm{O}(1)$ time. As there are $\mathrm{O}((mn)^2 + n^4)$ edges in $A(E)$, Step 3(d) can be done in $\mathrm{O}((mn)^2 + n^4)$ time and space. □

Step 4 is obvious and can be done in $\mathrm{O}((mn)^2 + n^4)$ time and space.

Theorem 2. *The constrained Voronoi diagram $\mathrm{CV}(S, O)$ can be constructed in $\mathrm{O}(((mn)^2 + n^4)n)$ time and space.*

Proof. Immediate. □

Reducing the time and space used in Find-$\mathrm{CV}(S, O)$. To reduce the time and space bounds by a factor of n, we must maintain and copy *no more than* $\mathrm{O}(m^2n + mn^2)$ VB buckets, CA lists

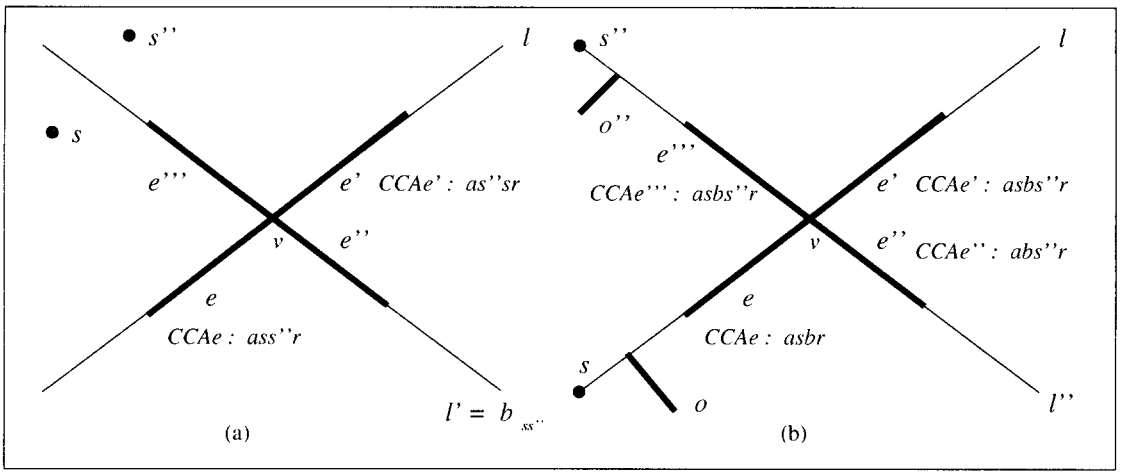

Fig. 5. Two illustrations of Lemma 9.

and ptr arrays. This can be achieved by combining Steps 3(a), (b) and (c), and having the edges to share buckets, lists and arrays. First, as edges on obstacles have two sides, we shall deal with them separately. Specifically, we determine the VB buckets of all the edges on or incident upon the obstacles using the same method described in Step 3(a). Following the analysis given there, this process takes $O((m^2n^2+n^4)+(m^2+mn^2)+(\sum_{p\in V'}\deg(p))n)$ time where V' is the set of vertices on the obstacles. Clearly, $\sum_{p\in V'}\deg(p)$ is the number of edges on or incident upon the obstacles which is bounded above by $O(m^2n+mn^2)$. Hence, setting the buckets of all these edges takes $O((m^2n+mn^2)n) = O(m^2n^2+n^4)$ time. Similarly, computing the CA arrays and the ptr pointers for these edges take $O(m^2n^2+n^4)$ time. Using VB_e and CA_e, for each e on or incident upon the obstacles, a doubly-linked list, called CCA_e, is created which chains all the sites in CA_e that are visible to e. The sites in CCA_e are ordered according to their order in CA_e. Computing the CCA lists can clearly be done in $O(m^2n^2+n^4)$ time. Since the closest-visible site of e is the first node in CCA_e, it can thus be determined in $O(1)$ time.

For the remaining edges in $A(E)$, our strategy is to share among the edges *no more* than $O(m^2n+mn^2)$ VB's, CA's, ptr's and CCA's. For clarity, we shall assume without loss of generality that $\deg(v)=4$ for every vertex v in $A(E)$.

Lemma 9. *Let v be a common endpoint of edges e, e', e'' and e''' in $A(E)$ where e and e' (respectively e'' and e''') are on the same base line l (respectively l''). Given $\mathrm{CCA}(e)$ and $\mathrm{CCA}(e'')$, $\mathrm{CCA}(e')$ and $\mathrm{CCA}(e''')$ can be determined in $O(1)$ time.*

Proof. Without loss of generality, we assume e''' is the successor of e. We consider three cases separately.

Case (i): both l and l'' are not visibility lines. Then by Lemma 3(1), $VB_{e'} = VB_{e'''} = VB_e$. If l'' is an obstacle line, then obviously, $CA_{e'} = CA_e$. Hence, $CCA_{e'} = CCA_e$. If l'' is a bisector line, then $CCA_{e'}$ can be obtained from CCA_e by swapping sites s and s'', where s and s'' determine l'' (Fig. 5(a)). Therefore, $CCA_{e'}$ can be computed in $O(1)$ time from CCA_e. Similarly, $CCA_{e'''}$ can be obtained from $CCA_{e''}$ in $O(1)$ time.

Case (ii): both l and l'' are visibility lines. Clearly, $CA_e = CA_{e'} = CA_{e''} = CA_{e'''}$. Let l (respectively l'') be determined by site s (respectively s'') and obstacle o (respectively o'').

(a) Suppose o, o'' are both down without restriction to e and e''' (Fig. 5(b)). Then o'' is down while o is up without restriction to e''' and e'. By Lemma 3(3)(i) and (iii), VB_e and $VB_{e'''}$ differ in *at most* the site s'' whereas $VB_{e'} = VB_{e'''}$. Hence, $CCA_{e'}$ can be obtained from CCA_e by inserting, at most, the site s'' into the list. The insertion can be done in $O(1)$ time as the neighboring sites of s'' in $CCA_{e'}$ can be determined by referring to $CCA_{e''}$ (Fig. 5(b)). Similarly, $CCA_{e'''}$ can be obtained from $CCA_{e''}$ in $O(1)$ time.

(b) In the remaining cases, it is easily verified that $CCA_{e'}$ and $CCA_{e'''}$ can be obtained from CCA_e and $CCA_{e''}$ in $O(1)$ time by deleting or inserting at most one site (s or s'') each.

Case (iii): exactly one of l, l'' is a visibility line. The argument is similar to those for the above two cases. □

Algorithm Closest-visible sites

(*Remark*: For clarity, we assume without loss of generality that no two lines in $A(E)$ are parallel.)

Step 0. Compute CCA_e and determine the closest-visible sites for every edge e on or incident upon an obstacle.

Step 1. Pick a line, say l, from E and compute CCA_e for every edge e on or incident upon l. Determine the closes-visible sites for each such e and mark the edge. For every edge incident upon l, mark its end-vertex on l and *partially-mark* its other end-vertex (if exists) once.

Step 2.

(a) Examine all the edges incident upon l to determine all those vertices which are *partially-marked* twice. Insert all those vertices into a queue Q.

(b) **repeat**
  while (the queue Q is non-empty) **do**
    remove a vertex v from Q and mark the vertex;
    let the four edges incidenting upon v be e, e', e'' and e''' and e, e'' be marked;
    for each of e' and e''' **do**
      if the edge has not been processed in Step 0
      then obtain its CCA list from one of e, e'' and modify it according to Lemma 9;
        determine the closest-visible sites of the edge;
    mark both e' and e'''
  endwhile;
  for each of the *newly marked* edges **do**
    if the edge is not an extreme edge (i.e., a ray)
    then let v' be the other endpoint; partially-mark v';
      if v' has thus been *partially-marked* twice then add v' into the queue Q.
  until (Q is empty).

To prove the correctness of the above algorithm, it suffices to prove the following assertion: "at the end of the dth execution of the **while** loop, all the edges whose longest paths from l has length at most d have been marked and all those edges whose longest paths from l has length exactly d are newly marked", where a path from l to an edge is defined as follows: convert every edge in $A(E)$

to a directed edge so that the edge is pointing away from the line l. A path from l to an edge e is a directed path from a vertex on l to the tail of e. The length of a path is the number of edges on it. The assertion can be easily proved by induction on d.

Let D be the maximum length of the paths from l to the edges. By the above assertion, after the Dth execution of the **while** loop, all edges are marked, indicating that their closest-visible sites are determined. Moreover, all the newly marked edges must be extreme edges (i.e., rays). As a result, no vertex is added to Q. Consequently, execution of the algorithm terminates at the **until** statement.

Lemma 10. *Algorithm* Closest-visible sites *computes the closest-visible sites for every edge of* $A(E)$ *in* $O(m^2n^2 + n^4)$ *time and space.*

Proof. As was mentioned earlier, Step 0 takes $O((mn)^2 + n^4)$ time and space. In Step 1, the methods described in Lemmas 4–6 can be used. Since l is the only line to be processed, this step can be done in $O(mn^2 + n^3)$ time and space. In Step 2, as no marked vertex is re-marked, no vertex is inserted into Q more than once. Consequently, no edge is examined more than twice. Since examining an edge and computing the CCA list for an edge in this step each takes $O(1)$ time (Lemma 9), Step 2 thus takes $O(mn^2 + n^4)$ time. The space used by the CCA lists in this step are those allocated in Steps 0 and 1 which is bounded by $O(mn^2 + n^4)$. □

Theorem 3. *The constrained Voronoi diagram,* $CV(S, O)$*, can be constructed in* $O(m^2n^2 + n^4)$ *time and space which is worst-case optimal when* $m \geqslant cn$ *for any constant* c.

Proof. Immediate. □

5. Constructing the constrained and weighted Voronoi diagrams

In this section, we first discuss the combinatorial complexity of constrained and weighted Voronoi diagram. We then show how to modify algorithm Find-CV(S, O) to construct such diagrams.

5.1. Preliminaries

Since the set of constrained and weighted Voronoi diagrams includes the set of constrained Voronoi diagrams as a proper subset, Theorem 1 immediately provides an $\Omega((mn)^2)$ worst-case lower bound on the combinatorial complexity of constrained and weighted Voronoi diagram.

Let $E_C = (O' \cup C \cup T)$ where O' is the set of obstacle lines; C is the set of circular bisectors and T is the set of visibility lines. Let $A(E_C)$ be the arrangement of E_C. From the fact that $A(E_C)$ embeds CWV(S, O), we immediately obtain the following lemma.

Lemma 11. *Every edge of* CWV(S, O) *is an edge of* $A(E_C)$ *or the union of some edges of* $A(E_C)$.

The above lemma indicates that constructing CWV(S, O) based on $A(E_C)$ is justifiable. Moreover, as $|E_C| = O(mn + n^2)$, by using a result of [5], we obtain the following counterpart of Lemma 1.

Lemma 12. *The number of edges and vertices in* CWV(S, O) *is* $O((mn)^2 + n^4)$.

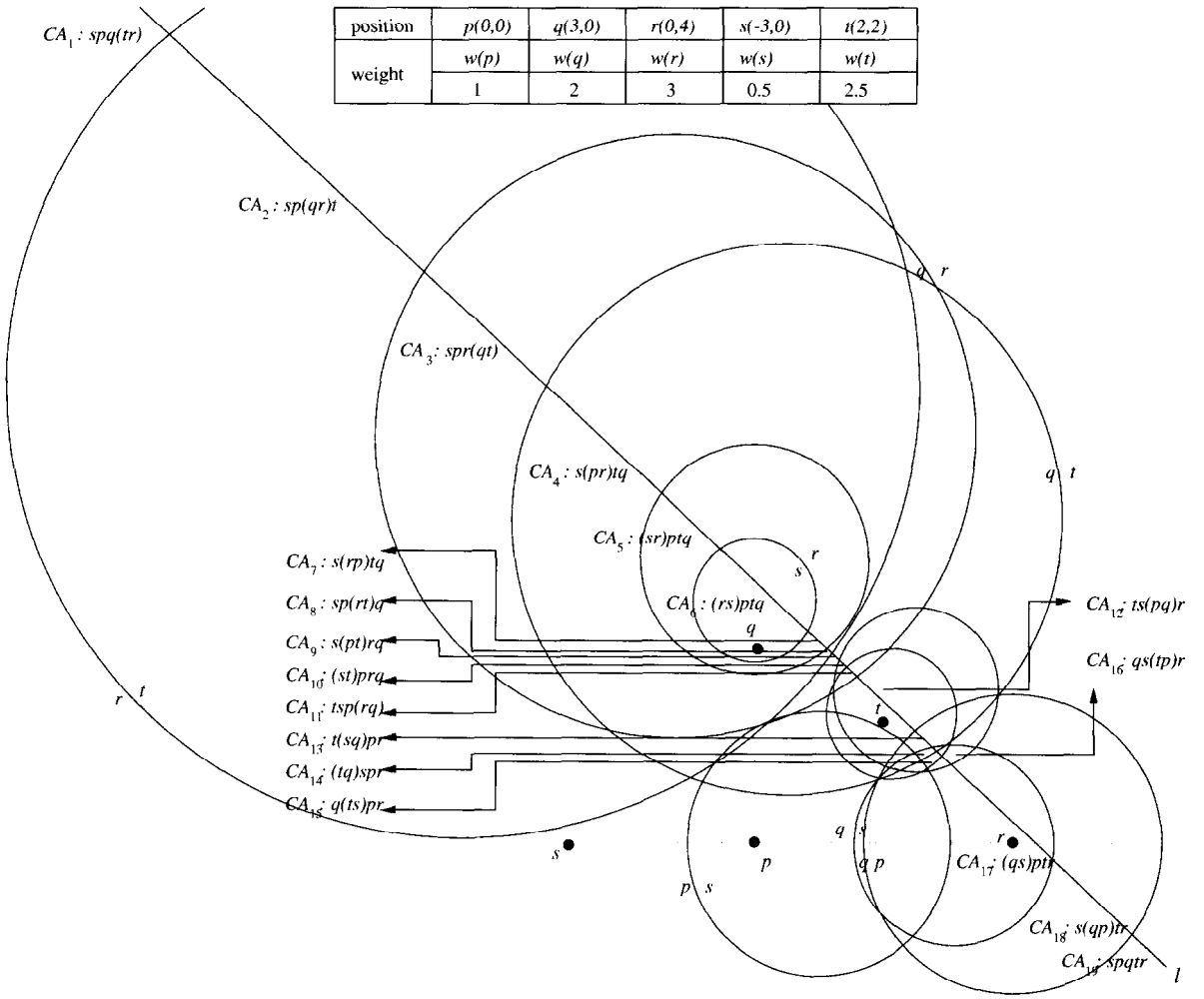

Fig. 6. The CA lists of the edges on a line in CWV(S, O). CA_i can be obtained from CA_{i-1} by swapping the pair of sites within the parentheses.

5.2. *An algorithm for constructing* CWV(S, O)

Owing to the fact that both CV(S, O) and CWV(S, O) are embedded in the arrangement of a set of lines determined by the sites and obstacles, and that they differ only in the shape of Voronoi edges originated from bisectors (specifically, those from the former can only be line segments whereas those from the latter can be either circular arcs or line segments), we shall attempt to modify algorithm Find-CV(S, O) to produce an algorithm for constructing CWV(S, O).

After a careful inspection of algorithm Find-CV(S, O), we identify the following three key issues which must be addressed in order to facilitate such modification: (1) how to insert C into arrangement $A(O' \cup T)$ to form $A(E_C)$; (2) how to find the visible-site list for each edge in $A(E_C)$; and (3) how to find the closest-site list for each edge in $A(E_C)$. We consider each issue separately below.

(1) Let $c_{i,j}$ denote the circular bisector determined by two weighted sites s_i and s_j. Suppose $w(s_i) > w(s_j)$, then site s_j lies inside $c_{i,j}$ and the visibility lines passing through s_j must cross $c_{i,j}$. Therefore, the insertion of $c_{i,j}$ can begin at the intersection of $c_{i,j}$ with one of those visibility lines. Starting from that intersection, we traverse $c_{i,j}$ and insert it, arc by arc, into $A(O' \cup T)$. As $c_{i,j}$ intersects each line and each circle at most twice, using the result of Edelsbrunner et al. [5], inserting $n(n-1)/2$ such $c_{i,j}$ into $A(O' \cup T)$ can be done in $O(m^2n^2 + n^4 2^{\alpha(n)})$ time and $O(m^2n^2 + n^4)$ space where $\alpha(n)$ is the functional inverse of the Ackermann's function.

(2) Note that the visible-site list for an edge in $A(E_C)$ is completely determined by the structure of $A(O' \cup T)$ and the position of the edge in the plane. Therefore, Step 3(a) of algorithm Find-CV(S, O) can be applied directly with slight modification.

(3) Again, Step 3(b) of algorithm Find-CV(S, O) can be applied directly with slight modification. (Refer to Fig. 6)

Hence, with slight modification, we can convert algorithm Find-CV(S, O) into an algorithm for constructing CWV(S, O). The resulting algorithm has $O(((m^2n^2 + n^4 2^{\alpha(n)})n))$ time complexity and $O(((m^2n^2 + n^4)n))$ space complexity.

Finally, it is easily verified that with slight modification, algorithm Closest-visible sites can also be used to reduce the time and space bounds by a factor of n, resulting in the next theorem.

Theorem 4. *The constrained and weighted Voronoi diagram* CWV(S, O) *can be constructed in* $O(m^2n^2 + n^4 2^{\alpha(n)})$ *time and* $O(m^2n^2 + n^4)$ *space.*

References

[1] F. Aurenhammer, Voronoi diagrams: A survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (3) (1991) 345–405.

[2] F. Aurenhammer, H. Edelsbrunner, An optimal algorithm for constructing the weighted Voronoi diagram in the plane, Pattern Recognition 17 (2) (1984) 251–257.

[3] F. Aurenhammer, G. Stockl, On the Peeper's Voronoi diagram, SIGACT News 22 (1991) 50–59.

[4] L. Chew, Constrained Delaunay triangulations, in: Proc. 3rd ACM Symposium on Computational Geometry, 1987, pp. 215–222.

[5] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, M. Sharir, Arrangements of curves in the plane: Combinatorial topology and algorithms, in: Proc. 15th International Colloquium on Automata, Languages and Programming, 1988, pp. 214–229.

[6] H. Edelsbrunner, J. O'Rourke, R. Seidel, Constructing arrangements of lines and hyperplanes with applications, SIAM J. Comput. 15 (2) (1986) 341–363.

[7] A. Lingas, Voronoi diagrams with barriers and their applications, Inform. Process. Lett. 32 (1989) 191–198.

[8] E. Papadopoulou, D.T. Lee, Efficient computation of the geodesic Voronoi diagram of points in a simple polygon, in: Proc. ESA '95, Lecture Notes in Computer Science, Springer, Berlin, 1995, pp. 238–251.

[9] R. Seidel, Constrained Delaunay triangulations and Voronoi diagrams with obstacles, Rep. 260, IIG-TU Graz, Austria, 1988, pp. 178–191.

[10] Y.H. Tsin, C. Wang, Geodesic Voronoi diagram in the presence of rectilinear barriers, Nordic J. Comput. 3 (1996) 1–26.

[11] C.Wang, L. Schubert, Optimal algorithm for constructing the Delaunay triangulation of a set of line segments, in: Proc. 3rd ACM Symposium on Computational Geometry, 1987, pp. 223–233.