Nordic Journal of Computing 1(1994), 231–245.

WIDEST-CORRIDOR PROBLEMS

RAVI JANARDAN*
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455, U.S.A.
janardan@cs.umn.edu

FRANCO P. PREPARATA†
Department of Computer Science
Brown University
Providence, RI 02912, U.S.A.
franco@cs.brown.edu

Abstract. A k-dense corridor through a finite set, S, of n points in the plane is the open region of the plane that is bounded by two parallel lines that intersect the convex hull of S and such that the region contains k points of S. The problem of finding a widest k-dense corridor arises in robot motion-planning. In this paper, efficient solutions are presented for several versions of this problem. Results include: two algorithms for finding widest k-dense corridors for any k, an algorithm to dynamically maintain a widest empty corridor under online insertions and deletions in S, an algorithm to find a widest $(n-1)$-dense closed corridor, and a widest empty corridor algorithm for polygonal obstacles. The techniques used are based on geometric duality and on efficient searching in the convex layers of a point-set.

ACM CCS Categories and Subject Descriptors: E.1, F.2.2, I.2.9

Key words: arrangement, computational geometry, convex layers, data structures, geometric duality

1. Introduction

Let S be a set of n points in the Euclidean plane. A *corridor* through S is the open region of the plane that is bounded by two parallel straight lines that intersect the convex hull, $\mathrm{CH}(S)$, of S. The *width* of a corridor is the distance between the bounding lines. A corridor is called *k-dense* if it contains k points of S, where $0 \leq k \leq n-2$. (As we will see later, each of the bounding lines must contain at least one point of S, and hence $k \leq n-2$.) A *widest k-dense corridor* through S is a k-dense corridor of maximum width.

The problem of constructing a widest k-dense corridor arises in robot motion-planning. Houle and Maciel [6] gave an $O(n^2)$-time and $O(n)$-space algorithm to compute a widest empty (i.e., 0-dense) corridor. Subsequently, Chattopadhyay and Das [1] showed how to compute a widest k-dense corridor, $0 < k \leq n-2$, in $O(n^2 \log n)$ time and $O(n^2)$ space. In this paper, we present efficient algorithms for several versions of the widest k-dense corridor problem. Our results include:

*Research supported in part by NSF grant CCR–92–00270.
†Research supported in part by NSF grant CCR–91–96176.

Received August 1993. Accepted February 1994.

(1) A simple sweepline algorithm to compute a widest k-dense corridor through S in $O(n^2 \log n)$ time and $O(n)$ space, where $0 \leq k \leq n-2$. This improves upon the space bound in [1]. Moreover, using this approach we can **(i)** compute a widest k-dense corridor for each k, where $k = 0, 1, \ldots, n-2$, in $O(n^3)$ time and $O(n)$ space and **(ii)** compute in $O(n^2 \log n)$ time and $O(n)$ space a minimum-density corridor of width at least w for any given real number w (provided such a corridor exists for the given w).

(2) An algorithm to dynamically maintain a widest empty corridor in $O(n \log n)$ time per update and $O(n^2)$ space as points are inserted and deleted online in S. This is considerably more efficient than recomputing a widest empty corridor from scratch after each update.

(3) An algorithm to compute a widest empty corridor through a set of polygonal obstacles in the plane in $O(n^2)$ time and $O(n)$ space, where n now is the total number of edges of the polygons.

(4) An $O(kn^2)$-time and $O(kn^2)$-space algorithm for computing a widest k-dense corridor through S, $0 < k \leq n-2$. This algorithm is faster than our algorithm in 1 above for small k, i.e., when $k = o(\log n)$; however, it uses more space.

(5) Let a *k-dense closed corridor* through S be the closed region of the plane that is bounded by two parallel lines that intersect $\mathrm{CH}(S)$ and which contains k points of S, where $2 \leq k \leq n$. For $2 \leq k \leq n-2$, our algorithms in 1 and 4 above (as well as the algorithm of [1]) can be modified easily to compute widest k-dense closed corridors, without affecting their time and space bounds. As observed in [1], a widest n-dense closed corridor is determined by a diametral pair of S and hence can be computed in $O(n \log n)$ time and $O(n)$ space [7]. Here we show how to also compute a widest $(n-1)$-dense closed corridor within the same bounds.

Our approach in results 1–4 is based on geometric duality, which transforms the corridor problem to an equivalent problem on a set of lines. Result 5 is based primarily on a technique for searching efficiently in the convex layers of a planar point-set.

2. Geometric preliminaries

Throughout the paper we assume that no three points of S are collinear and no four points form the vertices of a trapezoid. These assumptions simplify the exposition and can be removed easily. Moreover, we will assume that no widest k-dense corridor is vertical since such a corridor can be computed easily in $O(n \log n)$ time by sorting the points by x-coordinate.

The following theorem states a key property of widest k-dense corridors. (As noted in [1], this theorem follows immediately from [6].)

THEOREM 2.1. ([1, 6]) *Let $C^\star$ be a widest k-dense corridor through S, with bounding lines ℓ' and ℓ''. Then one of the following conditions must hold:*

(a) *One of the lines, say ℓ', passes through two points p_i and p_j of S and ℓ'' passes through a point p_h of S, or*

(b) *there are points p_i and p_j of S such that ℓ' passes through p_i, ℓ'' passes through p_j, and ℓ' and ℓ'' are perpendicular to the line passing through p_i and p_j.* □

Thus, we can restrict our search for a widest k-dense corridor to those k-dense corridors, C, that satisfy one of these conditions. We will call a k-dense corridor C which satisfies condition (a) (resp. condition (b)) a *type-(a)* (resp. *type-(b)*) corridor.

Let us reinterpret conditions (a) and (b) in the dual plane, under the duality transform $\mathcal{F}$, which maps a point $p = (a, b)$ to the line $\mathcal{F}(p) : y = ax - b$ and the non-vertical line $\ell : y = mx - c$ to the point $\mathcal{F}(\ell) = (m, c)$. Note that $\mathcal{F}(\mathcal{F}(p)) = p$ and $\mathcal{F}(\mathcal{F}(\ell)) = \ell$, that parallel lines are mapped to points with the same abscissa, and p lies below (resp., on, above) ℓ if and only if $\mathcal{F}(p)$ lies above (resp. on, below) $\mathcal{F}(\ell)$.

Let H be the set of lines $\{\ell_i = \mathcal{F}(p_i) \mid p_i \in S\}$. Let $\mathcal{A}$ be their *arrangement*, i.e., the subdivision of the plane induced by H. Let $v_{ij} = \ell_i \cap \ell_j$ be any vertex of $\mathcal{A}$ and $x(v_{ij})$ its abscissa.[1] The following is easily shown:

If C is a type-(a) corridor, then $\mathcal{F}(\ell') = v_{ij}$ and $\mathcal{F}(\ell'')$ is the point where the vertical line through v_{ij} intersects ℓ_h. If C is a type-(b) corridor, then $\mathcal{F}(\ell')$ and $\mathcal{F}(\ell'')$ are points on ℓ_i and ℓ_j, respectively, where both points have abscissa $-1/x(v_{ij})$ (because ℓ' and ℓ'' are perpendicular to the line passing through p_i and p_j and the slope of this line is $x(v_{ij})$). Moreover, if C is of type-(a) (resp. type-(b)) and ℓ is any line that is parallel to ℓ' and ℓ'' and contained in C, then $x(\mathcal{F}(\ell)) = x(v_{ij})$ (resp. $x(\mathcal{F}(\ell)) = -1/x(v_{ij})$) and $y(\mathcal{F}(\ell')) < y(\mathcal{F}(\ell)) < y(\mathcal{F}(\ell''))$.

Thus, in either case, the dual of C, which we denote by $\mathcal{F}(C)$, is the open vertical line segment bounded by $\mathcal{F}(\ell')$ and $\mathcal{F}(\ell'')$.[2] Moreover, C is a k-dense corridor through S if and only if $\mathcal{F}(C)$ intersects k lines of H. Also, the width of C is $|y(\mathcal{F}(\ell')) - y(\mathcal{F}(\ell''))|/\sqrt{1 + x(\mathcal{F}(C))^2}$.

We can now compute $C^\star$ as follows: We consider each vertex v_{ij} of $\mathcal{A}$ in turn, find the $(k+1)$st line vertically above it (if it exists), and compute the corresponding type-(a) k-dense corridor. Similarly for the $(k+1)$st line vertically below v_{ij}. Also, if at abscissa $-1/x(v_{ij})$, there are exactly k lines between ℓ_i and ℓ_j (excluding ℓ_i and ℓ_j) then we compute the corresponding type-(b) k-dense corridor. At the end, we output the overall widest corridor found.

[1] In general, we will use $x(p)$ and $y(p)$ to denote, respectively, the abscissa and the ordinate of a point p. Also, we use $x(s)$ to denote the abscissa of a vertical line segment s.

[2] Using $\mathcal{F}(C)$ to denote the dual of C is a minor, but convenient, abuse of notation. For convenience, we call $\mathcal{F}(\ell')$ and $\mathcal{F}(\ell'')$ the *endpoints* of $\mathcal{F}(C)$ even though they do not belong to $\mathcal{F}(C)$.

3. A general algorithm for widest k-dense corridors

We give a simple sweepline algorithm to compute a widest k-dense corridor $C^\star$ in $O(n^2 \log n)$ time and $O(n)$ space. Our algorithm sweeps over the dual lines but does not maintain the entire arrangement in storage. A similar approach was used by Edelsbrunner and Welzl [5] for a different problem.

We use two sweeplines: a *master* sweepline, V, to find type-(a) corridors and a *slave* sweepline, V', to find type-(b) corridors. As V sweeps from $x = -\infty$ through $x = 0$ to $x = \infty$, V' moves from $x = 0$ to $x = \infty$ and then from $x = -\infty$ to $x = 0$.

Consider the master sweep. Associated with V are an array, D, and a priority queue, Q, organized as a min-heap.[3] These structures satisfy the following invariants: D stores the dual lines in the order in which they intersect V, from bottom to top. Q stores v_{fg}, with key $x(v_{fg})$, if ℓ_f and ℓ_g are adjacent in D and intersect to the right of V. We store with ℓ_f and ℓ_g in D a pointer to v_{fg} in Q and also store with v_{fg} in Q a pointer to ℓ_f and to ℓ_g in D.

The master sweep is initialized by inserting the lines into D in increasing order of the ordinates of their intersections with the line $x = -\infty$.

In a general step, the next event point v_{ij} is obtained by deleting the item with minimum key in Q. To the immediate left of v_{ij}, let ℓ_i be above ℓ_j, let ℓ_a be the line above ℓ_i, and let ℓ_b be the line below ℓ_j. The following actions are performed at v_{ij}: **(1)** ℓ_i and ℓ_j are interchanged in D. **(2)** If v_{ia} is to the right of V, then it is deleted from Q. Similarly for v_{jb}. **(3)** If v_{ib} is to the right of V, then it is inserted into Q. Similarly, for v_{ja}. **(4)** If ℓ_j is now in $D[t]$, then the $(k+1)$st line above v_{ij} is in $D[t+k+1]$ (provided $t+k+1 \leq n$) and the $(k+1)$st line below v_{ij} is in $D[t-k-2]$ (provided $t-k-2 \geq 1$). The corresponding type-(a) k-dense corridors are computed.

Now consider the slave sweep. The data structures, D' and Q', associated with V' are analogous to D and Q. With each line in D we store a pointer to the line in D' and vice versa. The slave sweep is initialized at $x = 0$, by starting V' at $x = -\infty$ and sweeping up to $x = 0$, performing steps analogous to 1–3 above on D' and Q'.

Subsequently, suppose that V is at event point v_{ij}. Then V' advances to the rightmost vertex u such that $x(u) \leq -1/x(v_{ij})$, performing steps 1–3 above on D' and Q' at each vertex (including u) that is encountered. (The desired event point for V' is actually $x = -1/x(v_{ij})$. However, the status of D' and Q' just after u is processed is the same as their status at $x = -1/x(v_{ij})$.) The final step (step 4) is as follows: Suppose that ℓ_i is in $D'[b]$ and ℓ_j is in $D'[c]$. If $|b-c|-1 = k$, then there are k lines between ℓ_i and ℓ_j at abscissa $-1/x(v_{ij})$ and so the corresponding type-(b) k-dense corridor is computed.

Once V reaches $x = 0$, V' is re-initialized at $x = -\infty$ and the two sweeps

[3] We call Q a priority queue with the understanding of course that elements of lower priority can also be deleted in logarithmic time (given their position).

are continued until Q becomes empty, i.e., V reaches $x = \infty$.

It is clear that the algorithm is correct and uses $O(n)$ space. The running time is dominated by the $O(\log n)$ time for the priority queue operations at $O(n^2)$ arrangement vertices.

THEOREM 3.1. *A widest k-dense corridor ($0 \leq k \leq n-2$) through n points in the plane can be computed in $O(n^2 \log n)$ time using $O(n)$ space.* □

3.1 Extensions of the general algorithm

Suppose that we want to compute a widest k-dense corridor for each k, where $k = 0, 1, \ldots, n-2$. At each master event point v_{ij}, we have a type-(a) k-dense corridor for each k for which the $(k+1)$st line above (and, similarly, below) v_{ij} exists. At the corresponding slave event point, there will be k lines between ℓ_i and ℓ_j, for some k, thus yielding a type-(b) k-dense corridor. For each k, we maintain the widest k-dense corridor found so far. The time per event point now becomes $O(n)$, which gives:

COROLLARY 3.1. *For $k = 0, 1, \ldots, n-2$, the set of widest k-dense corridors through n points in the plane can be computed in $O(n^3)$ total time using $O(n)$ space.* □

For any real number w, call a corridor through S a $(\geq w)$-*wide corridor* if its width is at least w. Suppose that we want to find a $(\geq w)$-wide corridor through S of minimum density. W.l.o.g., we may assume that w is no larger than the diameter of S since, otherwise, a $(\geq w)$-wide corridor cannot exist.

Clearly, if there is a $(\geq w)$-wide minimum-density corridor C, then there is a $(\geq w)$-wide minimum-density corridor which satisfies one of the conditions (a) or (b) of Theorem 2.1. Thus, the only change to the sweepline algorithm is in step 4 of the master and slave sweeps. For any abscissa x, let $\Delta_y(x) = w\sqrt{1+x^2}$. Let v_{ij} be the current event point in the master sweep.

In step 4 of the master sweep, we do a binary search in D for the first line, ℓ, above (and, similarly, below) v_{ij} such that the vertical line segment bounded by ℓ and v_{ij} has length at least $\Delta_y(x(v_{ij}))$. If ℓ exists, then we have a $(\geq w)$-wide corridor of type-(a). In step 4 of the slave sweep, we check whether the vertical line segment with abscissa $-1/x(v_{ij})$ and bounded by ℓ_i and ℓ_j has length at least $\Delta_y(-1/x(v_{ij}))$. If so, then we have a $(\geq w)$-wide corridor of type-(b). We conclude:

COROLLARY 3.2. *Let S be a set of n points in the plane. Given a real number w (assumed to be no larger than the diameter of S), a corridor through S of width at least w and of minimum density can be computed in $O(n^2 \log n)$ time and $O(n)$ space.* □

4. Widest empty corridors

4.1 Widest empty corridor through a planar point-set

We give an $O(n^2)$-time and $O(n)$-space algorithm to compute a widest empty corridor $C^\star$. Except for minor differences, the algorithm is analogous to the one given by Houle and Maciel [6]. We present it here for completeness since our results in later sections use some of the ideas.

Let C be an empty corridor. Since $k = 0$, it follows that both endpoints of $\mathcal{F}(C)$ lie on the boundary of the same face, f, of $\mathcal{A}$. Thus the search for $C^\star$ can be restricted to the faces of $\mathcal{A}$. We first give an $O(n^2)$-time and -space algorithm to compute $C^\star$.

We first construct the arrangement $\mathcal{A}$. This takes $O(n^2)$ time and space using, for instance, the algorithm of [2], and yields a standard planar graph representation of $\mathcal{A}$ in which the edges incident to each vertex are available in sorted order around the vertex and the boundary of each face f can be traversed in time $O(|f|)$.

We determine the empty type-(a) corridors, C, corresponding to f, as follows: f is convex and its boundary can be decomposed into a lower chain, L, and an upper chain, U, whose endpoints are the leftmost and rightmost vertices of f. For each internal (i.e., non-endpoint) vertex on U, we need to find the first line vertically below it. This line must contain an edge of L. We can find this edge for each vertex of U as follows:

We consider the interior vertices of U in turn, from left to right. For the leftmost internal vertex of U, we scan L from left to right and stop as soon as we find an edge vertically below the vertex. For each subsequent interior vertex of U, the scan of L can be resumed from where it had stopped for the previous vertex (since f is convex). Thus the total time is $O(|L| + |U|) = O(|f|)$. We can symmetrically compute for each interior vertex of L the first line vertically above it.

We determine the empty type-(b) corridors corresponding to f as follows: Consider the sorted list of abscissae of the vertices of f and let I be some open interval defined by consecutive abscissas in the list. With I we can associate two lines, ℓ_i and ℓ_j, where ℓ_i contains an edge of U and ℓ_j contains an edge of L, such that for any abscissa in I, ℓ_i is the first line vertically above ℓ_j and ℓ_j is the first line vertically below ℓ_i. (See Figure 1.) Thus we only need to check whether $-1/x(v_{ij}) \in I$, which can be done in constant time given ℓ_i and ℓ_j.

The pairs of lines associated with each interval can be determined in time $O(|f|)$ as follows: We scan f from left to right and merge the vertices of L and U into a list $v_1, v_2, \ldots, v_{|f|}$, sorted by x-coordinate. If v_1 is the intersection of lines ℓ_p and ℓ_q, where ℓ_p is above ℓ_q to the immediate right of v_1, then the lines associated with the interval $(x(v_1), x(v_2))$ are ℓ_p and ℓ_q. Now v_2 must be the intersection of one of ℓ_p and ℓ_q, say ℓ_p, with a new line ℓ_r. Thus the lines associated with the interval $(x(v_2), x(v_3))$ are ℓ_q and ℓ_r. And so on.

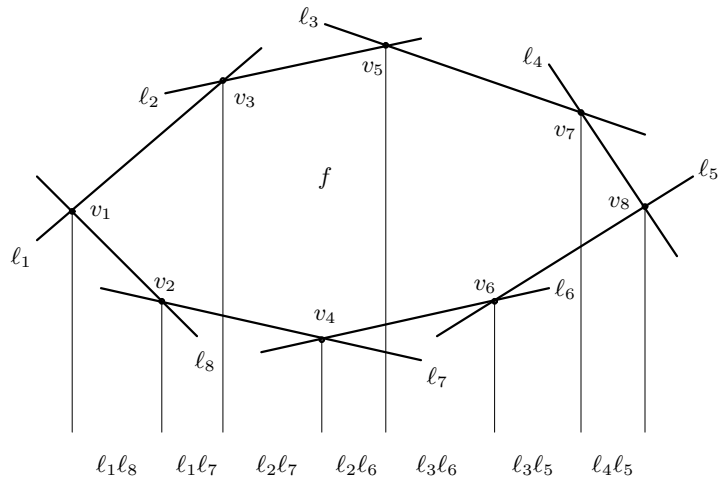

Fig. 1: *Partitioning the x-span of face f into open intervals. With each interval, I, can be associated two bounding lines, $\ell_i\ell_j$, of f such that anywhere inside I, ℓ_i (resp. ℓ_j) is the line that is immediately above (resp. below) ℓ_j (resp. ℓ_i).*

Clearly, when all faces have been thus processed, the widest empty corridor through S will have been found. The time to process all faces of $\mathcal{A}$ is $O(\sum_{f\in\mathcal{A}} |f|) = O(n^2)$, since $\mathcal{A}$ is a planar graph with $O(n^2)$ vertices. Thus, inclusive of the time to compute $\mathcal{A}$, the algorithm takes $O(n^2)$ time. The space used is $O(n^2)$ since the entire arrangement is stored.

To reduce the space to $O(n)$ we compute only the portions of $\mathcal{A}$ that are relevant at any given time. Towards this end, we use the topological sweep paradigm of Edelsbrunner and Guibas [3]. They showed that by using a sweepline that is tailored to the topology of the arrangement being swept, it is possible to visit the vertices, edges, and faces of an arrangement in a systematic way in $O(n^2)$ time and $O(n)$ space. Moreover, they showed that as soon as a face f is reached in the sweep, its vertices and edges can be extracted in $O(1)$ time apiece from the supporting data structures. Thus, we can extract and process f in $O(|f|) = O(n)$ time and space and then reuse the space for the next face. We conclude:

THEOREM 4.1. ([6]) *A widest empty corridor through a set of n points in the plane can be computed in $O(n^2)$ time and $O(n)$ space.* □

4.2 Dynamic maintenance of a widest empty corridor

We now describe how the widest empty corridor through S can be efficiently updated online as points are inserted into or deleted from S. Let n be the current size of S. Our approach is based on the following easily-verified properties:

(1) When a point p is inserted into S, an empty corridor C is destroyed only if the endpoints of $\mathcal{F}(C)$ lie on the boundary of a face of $\mathcal{A}$ that

is intersected by $\mathcal{F}(p)$. Moreover, if $\mathcal{A}'$ is the arrangement resulting from the insertion of $\mathcal{F}(p)$, then the search for new empty corridors can be restricted to those faces of $\mathcal{A}'$ to which $\mathcal{F}(p)$ contributes an edge. Symmetrically for deletions.

(2) In an arrangement of n lines, the sum of the sizes of the faces intersected by any line is $O(n)$ (see [2] for a proof). This implies that the number of empty corridors affected by an update is $O(n)$.

In addition to $\mathcal{A}$, we also maintain a priority queue, Q. For each empty corridor of S, with bounding lines ℓ' and ℓ'' and width w, there is an entry in Q consisting of ℓ', ℓ'', and w. Q is organized as a max-heap on the w's; thus the widest empty corridor can be retrieved in $O(1)$ time following an update.

With each vertex v_{ij} of $\mathcal{A}$, up to three empty corridors can be associated, namely, two type-(a) corridors and one type-(b) corridor. We store with v_{ij} a pointer to each of the three associated corridors in Q. With this setup, if one of the corridors associated with v_{ij} is to be deleted, its position in Q can be determined in $O(1)$ time. Clearly, Q has size $O(n^2)$. Thus it supports insertions and arbitrary deletions (given the position of the entry to be deleted) in $O(\log n)$ time.

Consider the insertion of a point p. Starting with the leftmost intersection of $\mathcal{F}(p)$ with $\mathcal{A}$, we successively traverse the boundaries of the faces intersected by $\mathcal{F}(p)$ and update $\mathcal{A}$ by adding the edges and vertices induced by the intersection of $\mathcal{F}(p)$ with $\mathcal{A}$. During the traversal, we also do the following: Suppose that $\mathcal{F}(p)$ splits a face f into faces f_1 and f_2. We find (as in Section 4.1) the corridors whose duals have both their endpoints on the boundary of f, and delete from Q those corridors for which one endpoint of the dual is on the boundary of f_1 and the other endpoint is on the boundary of f_2. We then find the corridors whose duals have both their endpoints on the boundary of f_1 (and similarly for f_2) and insert these into Q. All this takes $O(|f| \log n)$ time, which when summed up over all faces f intersected by $\mathcal{F}(p)$ yields a time bound of $O(n \log n)$, by Property 2 above.

Deletion of a point p is essentially the reverse.

THEOREM 4.2. *A widest empty corridor through a planar point-set S can be maintained dynamically, under online insertions and deletions in S, in $O(n \log n)$ time and $O(n^2)$ space, where n is the current size of S.* □

4.3 Widest empty corridor through polygonal obstacles

Let P be a set of polygonal obstacles in the plane with a total of n edges. An empty corridor, C, through P is the open region of the plane that is enclosed by two parallel straight lines that intersect the convex hull of the vertices of the polygons in P and such that the region does not intersect any polygon in P. (Note that for a given P no empty corridor may exist.) It is easy to prove that if $C^\star$ is a widest empty corridor through P, with

bounding lines ℓ and ℓ', then one of the conditions (a) or (b) of Theorem 2.1 must hold, where p_i, p_j, and p_h are now vertices of polygons in P.

Let T be the set of edges of the polygons. Under $\mathcal{F}$, a segment $t \in T$, with endpoints p and p', is mapped to the doublewedge, $W(t)$, formed by $\mathcal{F}(p)$ and $\mathcal{F}(p')$ and not containing the vertical line through the point $\mathcal{F}(p) \cap \mathcal{F}(p')$. Furthermore, a line ℓ intersects t if and only if $\mathcal{F}(\ell)$ lies in $W(t)$.

Let $\mathcal{A}$ be the arrangement of the lines bounding the doublewedges $W(t)$ for $t \in T$. It is well-known that lines ℓ_1 and ℓ_2 intersect the same segments of T (hence the same number of segments) if and only if $\mathcal{F}(\ell_1)$ and $\mathcal{F}(\ell_2)$ lie in the same face, f, of $\mathcal{A}$. Let $count(f)$ be the number of segments of T intersected by any line whose dual point falls in f. Clearly, an open vertical line segment whose endpoints lie on the boundary of a face f such that $count(f) = 0$ is the dual of an empty corridor through P.

Thus, to find $C^\star$, we use the topological-sweep-based algorithm described in Section 4.1, but perform corridor computations only for the faces f of $\mathcal{A}$ for which $count(f) = 0$. To identify these faces during the sweep, we use a technique of Edelsbrunner and Guibas [3, page 182] to compute $count(f)$ in $O(1)$ time for each face f of $\mathcal{A}$ when it is first encountered in the sweep. We conclude:

THEOREM 4.3. *A widest empty corridor through a set of polygonal obstacles in the plane can be computed in $O(n^2)$ time and $O(n)$ space, where n is the total number of edges of the polygons.* □

5. A faster widest k-dense corridor algorithm for small k

We show how to compute a widest k-dense corridor, in $O(kn^2)$ time and $O(kn^2)$ space, where $k > 0$. As in Section 4.1, we process $\mathcal{A}$ face-by-face, determining for each vertex on the upper chain of a face the $(k+1)$st line, if any, that is vertically below it. Similarly, in a second face-by-face pass, we determine the $(k+1)$st line vertically above each vertex. In a third pass, we determine type-(b) corridors.

However, unlike Section 4.1, the $(k+1)$st lines below and above a vertex do not lie on faces containing the vertex but are instead several faces away, and so the processing involves cutting across face boundaries. To do this efficiently and systematically, we process the faces according to the following total order [4].

For distinct faces f and g of $\mathcal{A}$, say that $f \gg g$ if there is a vertical line that intersects both f and g, f above g. It is well-known [4] that the relation $\gg$ is acyclic. Consider the subset $\succ$ of $\gg$ consisting of those pairs (f, g) that share an edge. We can compute $\succ$ in $O(n^2)$ time by traversing the boundary of each face of $\mathcal{A}$. The relation $\succ$ has cardinality $O(n^2)$ and, moreover, its transitive closure coincides with that of $\gg$. The desired total order is obtained by performing a topological sort on $\succ$, which takes $O(n^2)$ time.

Let $f_1, f_2, \ldots, f_s$ be the faces of $\mathcal{A}$ under this total order, where f_1 (resp. f_s) is the topmost (resp. bottommost) face of $\mathcal{A}$. With the exception of these two faces, each f_i can be decomposed into a lower chain L_i and an upper chain U_i; faces f_1 and f_s consist of only a lower chain L_1 and an upper chain U_s, respectively.

We begin with face f_2 and, as in Section 4.1, proceed to map each arrangement vertex on U_2 to a point vertically below it on L_2. In general, let f_i be the face to be processed next. The set, P_i, of points on U_i that must be resolved (i.e., mapped to points vertically below on L_i) consists of (i) arrangement vertices lying on U_i and (ii) downward projections of arrangement vertices lying on upper chains U_j, where $j < i$. With each such point $p \in P_i$, we associate a pointer, $vert(p)$, to the arrangement vertex that projects to p and a level number, $level(p)$, where $level(p) = h$ implies that $vert(p)$ has been projected vertically downwards h times, where $0 \leq h \leq k+1$. In particular, if p is an arrangement vertex lying on U_i, then $vert(p) = p$ and $level(p) = 0$.

To resolve P_i, we scan the boundary of U_i and the boundary of L_i simultaneously and for each point $p \in P_i$ we do the following: If $level(p) = k+1$ then the open vertical line segment with endpoints p and $vert(p)$ intersects k lines and thus is the dual of a type-(a) corridor. We compute this corridor and then discard p. If, however, $level(p) < k+1$, then we reset p to the point vertically below it on L_i and increment $level(p)$. Figure 2 illustrates the flow of the algorithm.

The first pass terminates once f_{s-1} has been processed. To compute the $(k+1)$st line vertically above each vertex of $\mathcal{A}$, we perform a symmetric second pass, processing faces in the reverse order. In addition, we determine for each vertex of $\mathcal{A}$ the kth line vertically above it. The reason for this will become clear in the sequel.

The idea behind finding type-(b) corridors is similar to that of Section 4.1 but somewhat more involved. We perform a third face-by-face pass, from f_1 to f_s, which simulates the first pass but, in addition, also does the following: Let f_i be the current face and let $Q_i = p_1, p_2, \ldots, p_q$ be the left–to–right sequence of points consisting of the arrangement vertices on L_i and of the points that have been projected down from U_i, where p_1 is the leftmost arrangement vertex on L_i. Let p_1 be the intersection of lines ℓ_x and ℓ_y, where ℓ_x supports an edge of U_i and ℓ_y supports an edge of L_i. Let ℓ_z be the kth line above p_1 (recall that ℓ_z was computed in the second pass). Now, for any abscissa in a sufficiently small neighborhood to the right of p_1, there are k lines between ℓ_y and ℓ_z. This is because at $x(p_1)$, there are $k-1$ lines between p_1 and ℓ_z and just to the right of p_1, ℓ_x contributes one to this count.

Let $j > 1$ be the smallest index such that $vert(p_j)$ is the intersection of one of ℓ_y and ℓ_z, say ℓ_y, with a new line ℓ_w. (Note that j exists and is, at worst, equal to q.) Then, for any abscissa in the interval $(x(p_1), x(p_j))$, there are k lines between ℓ_y and ℓ_z. Let v_{yz} be the arrangement vertex defined by the intersection of ℓ_y and ℓ_z. If $-1/x(v_{yz}) \in (x(p_1), x(p_j))$, then a k-dense

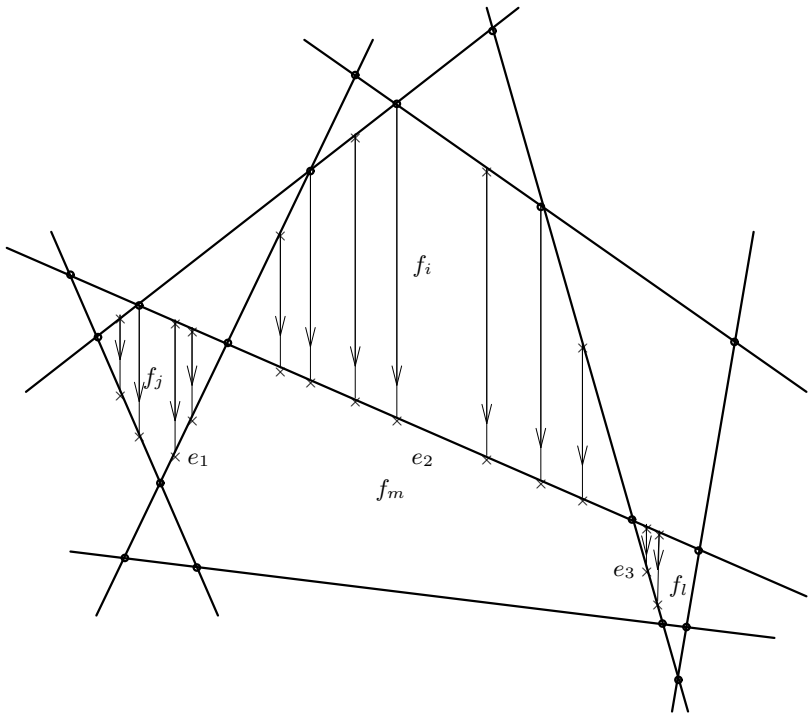

Fig. 2: *Flow of control in the first pass of the $O(kn^2)$ algorithm. (Solid circles denote arrangement vertices and crosses denote downward projections of arrangement vertices belonging to already-processed faces.) Among the faces f_i, f_j, f_l, f_m, face f_m is processed last. The points on the upper chain, $U_m = e_1e_2e_3$, of f_m that must be projected downward can be retrieved in sorted order along U_m by concatenating the lists of such points on the edges of U_m.*

type-(b) corridor has been found.

Again, for any abscissa in a sufficiently small neighborhood to the right of p_j, there are k lines between ℓ_z and ℓ_w and so we find the interval $(x(p_j), x(p_{j'}))$ for which this property holds. The processing of f_i in the third pass ends once p_q is reached.

Figure 3 illustrates the third pass for $k = 2$. Here $Q_i = p_1, p_2, \ldots, p_{10}$. The second line above p_1 is $\ell_z = D$, and we have $\ell_x = A$ and $\ell_y = B$. Point p_j is found to be p_2, which is the intersection of lines B and $\ell_w = C$. Thus, within $(x(p_1), x(p_2))$, there are two lines between lines B and D. Proceeding with D and C, we find that $p_{j'} = p_4$, the intersection of lines D and A (we skip p_3 since it involves neither D nor C). Thus, within $(x(p_2), x(p_4))$, there are two lines between C and D. And so on until $p_q = p_{10}$ is reached.

The running time of the algorithm is dominated by the time to project vertices. Since each vertex is projected at most k times, a simple charging argument shows that the total time is $O(kn^2)$. The space is also $O(kn^2)$ since this many points (projections and vertices) can be active at the same time.

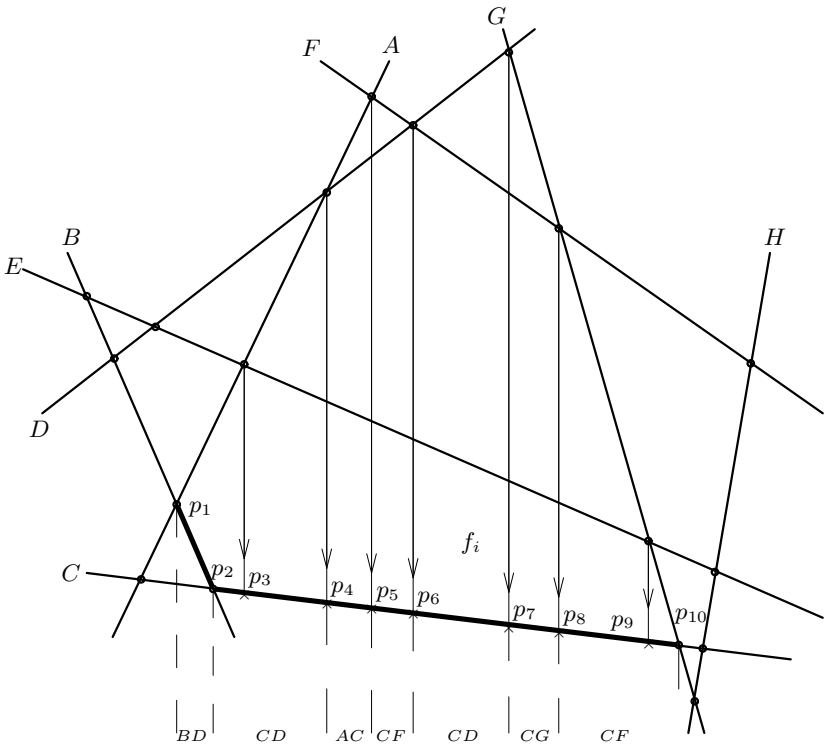

Fig. 3: *Third pass of the $O(kn^2)$ algorithm. The lower chain, L_i, of face f_i is shown in bold. The figure illustrates, for $k = 2$, how the x-span of f_i is partitioned into open intervals such that with each interval two lines can be associated that have k other lines between them at any abscissa within the interval.*

THEOREM 5.1. *A widest k-dense corridor through a set of n points in the plane can be computed in $O(kn^2)$ time and $O(kn^2)$ space, where $0 < k \leq n - 2$.* □

6. Computing a widest $(n-1)$-dense closed corridor efficiently

Since any $(n-1)$-dense closed corridor through S excludes exactly one point of S, the excluded point must be a vertex of the convex hull, $\mathrm{CH}(S)$, of S. Let $p_0, p_1, \ldots, p_{h-1}$ be the vertices of $\mathrm{CH}(S)$, taken clockwise. One strategy is to delete each hull vertex p_i in turn and compute a widest $(n-1)$-dense closed corridor through $S_i = S - \{p_i\}$. This corridor is determined by a diametral pair and hence is computable in linear time, given $\mathrm{CH}(S_i)$.

However, this approach will take at least quadratic time. Fortunately, there is enough coherence in the successively considered collections of points that a substantially more efficient algorithm can be developed.

Our algorithm uses the two outermost convex layers of S [7], namely, $L_0 = \mathrm{CH}(S)$ and $L_1 = \mathrm{CH}(S - L_0)$. When p_i is deleted, $\mathrm{CH}(S_i)$ can be

obtained by simply attaching between p_{i-1} and p_{i+1} a suitable subchain, $H(p_i)$, of L_1. (Throughout, indices are taken modulo h.) $H(p_i)$ is defined as follows: If the line segment $\overline{p_{i-1}p_{i+1}}$ does not intersect L_1, then $H(p_i)$ is simply this line segment. Otherwise, the first (resp. last) edge of $H(p_i)$, going clockwise, is the line segment $\overline{p_{i-1}p'}$ (resp. $\overline{p_{i+1}p''}$) which supports L_1 at the vertex p' (resp. p'') of L_1. Of the two possible such supporting segments from p_{i-1} (resp. p_{i+1}) to L_1, we take the one that makes the smaller interior angle with edge $\overline{p_{i-1}p_i}$ (resp. $\overline{p_{i+1}p_i}$). See Figure 4.

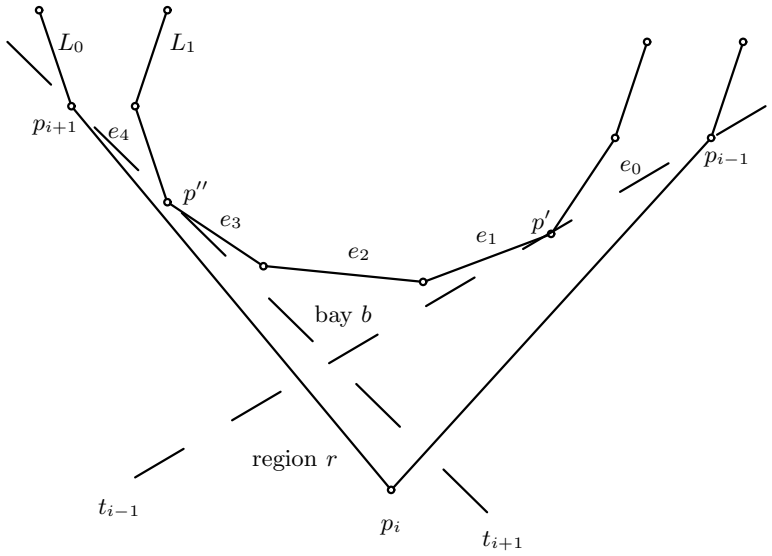

Fig. 4: *Chain $H(p_i)$ consists of the edges e_0 through e_4. Here t_{i-1} and t_{i+1} are, respectively, the supporting segments from p_{i-1} and p_{i+1} to L_1. Bay b is the region bounded by the edges e_1, e_2, e_3 and the upward-facing wedge formed by t_{i-1} and t_{i+1}. Region r is the region bounded by the downward-facing wedge formed by t_{i-1} and t_{i+1}.*

For future use, we now establish that, for $i \neq j$, $H(p_i)$ and $H(p_j)$ are edge-disjoint. Clearly, it suffices to show that $H(p_i)$ and $H(p_{i+1})$ have this property. From Figure 4, note that p_i must lie in the region r facing bay b for otherwise L_0 intersects L_1. Thus the supporting line from p_i to L_1 (the one which contains the first clockwise edge of $H(p_{i+1})$) cannot meet L_1 at any interior vertex of $H(p_i) \cap L_1$. This establishes the claim.

For any vertex $p_i \in \mathrm{CH}(S)$, a diametral pair of $\mathrm{CH}(S_i)$ is one of the *antipodal pairs* of vertices of $\mathrm{CH}(S_i)$, i.e., a pair of vertices through which parallel lines supporting $\mathrm{CH}(S_i)$ can be drawn [7]. Clearly, the $(n-1)$-dense closed corridors through S that exclude p_i correspond to those antipodal pairs of $\mathrm{CH}(S_i)$ that are not also antipodal pairs of $\mathrm{CH}(S)$. Thus, each $(n-1)$-dense closed corridor through S can be associated with an antipodal pair of some $\mathrm{CH}(S_i)$, $0 \leq i \leq h-1$.

We note the following about antipodal pairs. Given any planar point-set S', for each edge $e \in \mathrm{CH}(S')$ there is a (generally unique) vertex $\alpha(e)$ of

$\mathrm{CH}(S')$, not belonging to e, such that the line through $\alpha(e)$ and parallel to e does not intersect the interior of $\mathrm{CH}(S')$. It is then immediate that if e' and e'' are clockwise consecutive edges on $\mathrm{CH}(S')$, then the vertices clockwise from $\alpha(e')$ to $\alpha(e'')$ are exactly those forming antipodal pairs with the vertex shared by e' and e''.

It is therefore clear that each $(n-1)$-dense closed corridor through S can be determined by generating the edge-vertex pairs $(e, \alpha(e))$ of all $\mathrm{CH}(S_i)$, $0 \leq i \leq h-1$. Moreover, for each $\mathrm{CH}(S_i)$, we need only consider edges e that belong to $H(p_i)$. For each such e, $\alpha(e)$ is a vertex of $\mathrm{CH}(S)$.

The preceding discussion suggests a *rotating caliper* algorithm [7, 8]. Let $e^\star$ denote the edge for which $\alpha(e^\star)$ is currently being sought. Initializing $e^\star$ as the first clockwise edge of $H(p_0)$, we determine $\alpha(e^\star)$. Next we let $e^\star$ march clockwise along $H(p_0)$ while $\alpha(e^\star)$ marches clockwise along L_0, thus finding all antipodal pairs of $\mathrm{CH}(S_0)$ and determining the widest $(n-1)$-dense closed corridors through S that exclude p_0. Upon reaching p_1, we repeat the process for $H(p_1)$, starting with the first clockwise edge of $H(p_1)$. And so on until we process $H(p_{h-1})$.

L_0 and L_1 can be computed easily in $O(n \log n)$ time. Computing $H(p_i)$ essentially involves finding the two supporting lines to L_1, which can be done in $O(\log n)$ time. Thus the time to construct all the $H(p_i)$ is $O(n \log n)$. In the rotating caliper stage, the caliper moves monotonically clockwise because, as shown earlier, for $i \neq j$, $H(p_i)$ and $H(p_j)$ are edge-disjoint, so that the caliper is never forced to back up. For each edge e, the cost of determining $\alpha(e)$ is proportional to the number of vertices of $\mathrm{CH}(S)$ scanned and so can be charged as $O(1)$ per such vertex. Each such vertex is charged only $O(1)$ times because, as can be seen from Figure 4, for any i the supporting line from p_i to L_1 (which contains the first clockwise edge of $H(p_{i+1})$) forms a clockwise positive angle with t_{i+1} (which contains the last clockwise edge of $H(p_i)$). It follows that the rotating caliper stage runs in $O(n)$ time. We conclude:

THEOREM 6.1. *A widest $(n-1)$-dense closed corridor through a set S of n points in the plane can be computed in $O(n \log n)$ time and $O(n)$ space.* □

7. Conclusions and open problems

We have given efficient algorithms for several widest k-dense corridor problems, based on geometric duality and on efficient searching in convex layers. Open problems include: (i) computing a widest empty corridor in $o(n^2)$ time, (ii) computing a widest k-dense corridor in $O(n^2)$ time for any $k > 0$, and (iii) devising efficient algorithms to dynamically maintain widest k-dense corridors and to compute widest k-dense corridors through polygonal obstacles for any $k > 0$.

Acknowledgement

The authors thank Prof. Godfried Toussaint for making available the reference [6] and the two referees for helpful comments.

References

[1] S. Chattopadhyay and P. Das. The k-dense corridor problems. *Pattern Recognition Letters*, 11:463–469, 1990.

[2] B.M. Chazelle, L.J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[3] H. Edelsbrunner and L. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38:165–194, 1989.

[4] H. Edelsbrunner, L.J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

[5] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM Journal on Computing*, 15:271–284, 1986.

[6] M. Houle and A. Maciel. Finding the widest empty corridor through a set of points. In G.T. Toussaint, editor, *Snapshots of computational and discrete geometry*, pages 201–213. TR SOCS–88.11, Dept. of Computer Science, McGill University, Montreal, Canada, 1988.

[7] F.P. Preparata and M.I. Shamos. *Computational Geometry – An Introduction*. Springer–Verlag, 1988.

[8] G.T. Toussaint. Solving geometric problems with the 'rotating calipers'. In *Proceedings of IEEE MELECON '83*, 1983.