Note

Dynamically maintaining the widest k-dense corridor [☆]

Subhas C. Nandy[a, *], Tomohiro Harayama[b], Tetsuo Asano[b]

[a] *Computer and Statistical Service Center, 203, B.T. Road, Indian Statistical Institute, Calcutta 700 035, India*

[b] *School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan*

Received 15 September 1999; revised 29 May 2000; accepted 7 August 2000
Communicated by K.-Y. Chwa

Abstract

In this paper, we propose an improved algorithm for dynamically maintaining the *widest k-dense corridor* as proposed in Shin et al. (Inform. Process. Lett. 68 (1998) 25–31). Our algorithm maintains a data structure of size $O(n^2)$, where n is the number of points present on the floor at the current instant of time. For each insertion/deletion of points, the data structure can be updated in $O(n \log n)$ time, and the widest k-dense corridor in the updated environment can be reported in $O(kn + n \log n)$ time. Another interesting variation of this problem, called *query problem*, is also considered in this paper, where the objective is to report the widest k-dense corridor containing a given query point q. We propose two schemes for answering this query. In the first scheme, an $O(n^2)$ space data structure can answer this query in $O(nk)$ time. In the second scheme, we construct an $O(nk)$ space data structure afresh for each query, and then answer the query in $O(nk \log(\lfloor n/k \rfloor))$ time.

Keywords: Geometric duality; Levels of arrangement; Corridors

1. Introduction

Given a set S of n points in the Euclidean plane a corridor C is defined as an open region bounded by parallel straight lines ℓ' and ℓ'' such that it intersects the convex hull of S [4]. The width of the corridor C is the perpendicular distance between the bounding lines ℓ' and ℓ''. The corridor is said to be k-dense if C contains k points in

☆ This work was done when the author was visiting School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan.

* Corresponding author.

E-mail address: nandysc@isical.ac.in (S.C. Nandy).

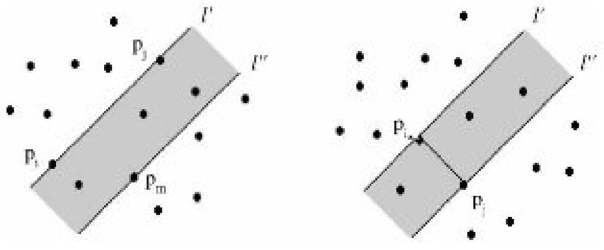

Fig. 1. Two types of corridors.

its interior. The widest k-dense corridor through S is a k-dense corridor of maximum width [1]. See Fig. 1 for illustration.

The widest empty corridor problem was first proposed in the context of robot motion planning where the objective was to find the widest straight route avoiding obstacles [4]. The widest k-dense corridor problem was introduced in [1] along with an algorithm of time and space complexities $O(n^2 \log n)$ and $O(n^2)$, respectively. Here the underlying assumption is that the robot can pass through (or in other words, can tolerate collision with) a specified number (k) of obstacles. In [5], the space complexity of the widest k-dense corridor problem was improved to $O(n)$. In the same paper, they have suggested an algorithm for maintaining the widest empty corridor, where the set of obstacles is dynamically changing. The space complexity of the algorithm is $O(n^2)$, and it takes $O(n \log n)$ time for each insertion/deletion of a point. However, the dynamic problem for general $k(\geqslant 0)$, was posed as an open problem. In [7], both the static and dynamic versions for the widest k-dense corridor problem were studied. The time and space complexities of their algorithm for the static version of the problem are $O(n^2)$ and $O(n^2)$, respectively. For the dynamic version, their algorithm is the pioneering work. Maintaining an $O(n^2)$ size data structure they proposed an algorithm which reports the widest k-dense corridor after the insertion and deletion of a point. The time complexity of their algorithm is $O(\mathscr{K} \log n)$, where $O(\mathscr{K})$ is the combinatorial complexity of $(\leqslant k)$-*level* of an arrangement of n half-lines, each of them belonging to and touching the same side of a given line. They proved that the value of $\mathscr{K}$ is $O(kn)$ in the worst case.

In this paper, we improve the time complexity of the dynamic version of the widest k-dense corridor problem. Given $O(n^2)$ space for maintaining the data structure, our algorithm can update the data structure and can report the widest k-dense corridor in $O(\mathscr{K} + n \log n)$ time for the insertion and deletion of a point. As it is an online algorithm, this reduction in the time complexity is definitely important.

In Ref. [7], another interesting variation of the corridor problem, called the *query problem*, was also studied. The objective was to report the widest k-dense corridor containing a given query point q. They proposed an algorithm which answers the query in $O(\mathscr{K} + \min(n \log^2 n, \mathscr{K} \log n))$ expected time. In order to answer each query, they construct the necessary data structure afresh, which requires $O(\mathscr{K})$ space. We show that, the $O(n^2)$ space data structure, which was proposed for our earlier problem, can easily be used to answer the *query problem* with deterministic time complexity $O(\mathscr{K})$. As the query point need not be inserted, we may avoid the quadratic storage cost by following the same principle of constructing the necessary data structure afresh for each individual query as suggested in [7]. We show that, it may cause an increase in the query answering time by a factor of $\log(\lfloor n/k \rfloor)$ in the worst case.

2. Geometric preliminaries

Throughout the paper, we assume that the points in S are in general position, i.e., no three points in S are collinear, and the lines passing through each pair of points have a distinct slope. Theorem 1, stated below, characterizes a widest corridor among the points S.

Theorem 1 (Chattopadhyay and Das [1], Houle and Maciel [4], Janardan and Preparata [5], Shin et al. [7]). *Let C^* be the widest corridor with bounding lines ℓ' and ℓ''. Then C^* must satisfy the following conditions*:

(A) *ℓ' passes through two distinct points p_i and p_j and ℓ'' passes through a single point p_m, or*

(B) *ℓ' and ℓ'' pass through the points p_i and p_j respectively, such that ℓ' and ℓ'' are perpendicular to the line joining p_i and p_j.*

From now onwards, a k-dense corridor satisfying conditions (A) and (B) will be referred to as *type-A* and *type-B* corridors, respectively (see Fig. 1).

2.1. Relevant properties of geometric duality

We follow the same tradition [1,4,5,7] of using geometric duality for solving this problem. It maps (i) a point $p = (a,b)$ to the line $\mathscr{D}(p)$: $y = ax - b$ in the dual plane, and (ii) a non-vertical line ℓ: $y = mx - c$ to the point $\mathscr{D}(\ell) = (m,c)$ in the dual plane. Needless to say, a point p is below (resp., on, above) a line ℓ in the primal plane if and only if $\mathscr{D}(p)$ is above (resp., on, below) $\mathscr{D}(\ell)$ in the dual plane. A line passing through two points p and q in the primal plane, corresponds to the point of intersection of the lines $\mathscr{D}(p)$ and $\mathscr{D}(q)$ in the dual plane, and vice versa.

Let ℓ' and ℓ'' be the two bounding lines of a non-vertical corridor C in the primal plane. As ℓ' and ℓ'' are mutually parallel, the corresponding two points, $\mathscr{D}(\ell')$ and $\mathscr{D}(\ell'')$ in the dual plane, will have the same x-coordinate. Thus, a corridor C will be represented by a vertical line segment joining $\mathscr{D}(\ell')$ and $\mathscr{D}(\ell'')$ in the dual plane, and will be denoted by $\mathscr{D}(C)$. The width of C is $(|y(\mathscr{D}(\ell')) - y(\mathscr{D}(\ell''))| / \sqrt{1 + (x(\mathscr{D}(\ell')))^2})$, and will be referred to as the *dual length* of $\mathscr{D}(C)$. Here, $x(p)$ and $y(p)$ denote the x- and y-coordinate of the point p, respectively.

Let $H = \{h_i = \mathscr{D}(p_i) \mid p_i \in S\}$ be the set of lines in the dual plane corresponding to the n points of S in the primal plane. Let p be a point inside the corridor C. In the dual plane, the points $\mathscr{D}(\ell')$ and $\mathscr{D}(\ell'')$ will lie on the opposite sides of the line $\mathscr{D}(p)$. Now, we have the following observations.

Observation 1. *Let C be a corridor bounded by a pair of parallel lines ℓ', ℓ''.*

Now, if C is a type-A corridor, ℓ' passes through p_i and p_j, and ℓ'' passes through p_m. This implies that $\mathscr{D}(\ell')$ corresponds to a vertex of $\mathscr{A}(H)$, which is the point of intersection of the lines h_i and h_j (denoted by $h_i \cap h_j$) in the dual plane, and $\mathscr{D}(\ell'')$ corresponds to a point on the line h_m satisfying $x(\mathscr{D}(\ell'')) = x(h_i \cap h_j)$.

If C is a type-B corridor, ℓ' and ℓ'' pass through the two points p_i and p_j, respectively. This implies that $\mathscr{D}(\ell')$ and $\mathscr{D}(\ell'')$ will correspond to the two points on the lines h_i and h_j, respectively, satisfying $x(\mathscr{D}(\ell')) = x(\mathscr{D}(\ell'')) = -(1/x(h_i \cap h_j))$.

Thus, a non-vertical *type-A* corridor may uniquely correspond to a vertex of $\mathscr{A}(H)$, and a non-vertical *type-B* corridor may also uniquely correspond to an edge of $\mathscr{A}(H)$, on which its upper end point lies.

Observation 2. *A corridor C is said to be k-dense if and only if there are exactly k lines of H that intersect the vertical line segment $\mathscr{D}(C)$, representing the dual of the corridor C, and will be commonly referred to as a k-stick.*

Thus, recognizing the widest k-dense non-vertical corridor in the primal plane is equivalent to finding a *k-stick* in the dual plane having maximum *dual length*.

3. Widest vertical k-dense corridor

We cannot apply geometric duality theory for vertical corridors. So, in order to get the widest vertical corridor in a dynamic environment, we maintain a balanced binary search tree, say $BB(\alpha)$-tree [6], with the existing set of points in the primal plane. Here each point in S appears at the leaf level, and is attached to the width of the widest k-dense vertical corridor with its left boundary passing through that point. At each non-leaf node, we attach the width of the widest vertical k-dense corridor in the subtree rooted at that node. It can be easily shown that for each insertion/deletion of a point, the necessary updates in this data structure and the reporting of the widest k-dense vertical corridor can be done in $O(k + \log n)$ time.

4. Widest non-vertical k-dense corridor

We now explain an appropriate scheme for maintaining the widest non-vertical k-dense corridor dynamically. Let $\mathscr{A}(H)$ denote the arrangement of the set of lines H [2]. The number of vertices, edges and faces in $\mathscr{A}(H)$ are all $O(n^2)$. In the dynamic scenario, we need to suggest an appropriate data structure which can be updated for

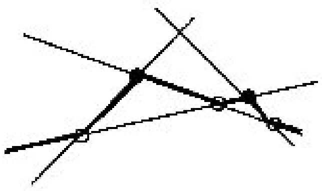

Fig. 2. Demonstration of levels in an arrangement of lines.

insertion/deletion of points, and the widest k-dense corridor can be reported efficiently in the changed scenario. As the deletion is symmetric to insertion, we shall explain our method for insertion of a new point in S only.

4.1. Data structures

We dynamically maintain the following data structure which stores the arrangement of the lines in H. It is defined using the concept of *levels* as stated below.

Definition 1 (*Edelsbrunner* [2]). A point π in the dual plane is at level θ ($0 \leqslant \theta \leqslant n$) if there are exactly θ lines in H that lie strictly below π. The θ-level of $\mathscr{A}(H)$ is the closure of a set of points on the lines of H whose levels are exactly θ in $\mathscr{A}(H)$, and is denoted as $L_\theta(H)$. See Fig. 2 for an illustration.

Clearly, $L_\theta(H)$ is a monotone polychain from $x = -\infty$ to ∞. In Fig. 2, a demonstration of levels in the arrangement $\mathscr{A}(H)$ is shown. Here the thick chain represents $L_1(H)$. Among the vertices of $L_1(H)$, those marked with empty (black) circles appear in level 0 (2) also. Each vertex of the arrangement $\mathscr{A}(H)$ appears in two consecutive levels, and each edge of $\mathscr{A}(H)$ appears in exactly one level. We shall store $L_\theta(H)$, $0 \leqslant \theta \leqslant n$, in a data structure as described below.

Level-structure: It is an array of size n, whose θth element consists of the following three fields:

Level-id: An integer containing the level-id θ.

Root-ptr: Pointing to the root node of a height-balanced binary tree $\mathscr{T}_\theta$, defined below.

List-ptr: Pointing to a linear link list, called *change-list*, whose each element is a tuple (ℓ, r) of pointers. The *change-list* data structure will be explained after defining $\mathscr{T}_\theta$.

$\mathscr{T}_\theta$: It is a height-balanced binary tree (AVL-tree), whose nodes correspond to the vertices and edges at level θ in left-to-right order.

Two integer fields, called *LEN* and *MAX*, are attached to each node. The *LEN* field contains the *dual length* of the *k-stick* attached to it. We explicitly mention that, if a node corresponds to a vertex of the arrangement, it defines at most one *k-stick*, but if it corresponds to an edge, more than one *k-stick* may be defined by that edge. In that case, the *LEN* field will contain the length of the one having maximum dual length among them. A node (corresponding to an edge) defining no *k-stick* will contain a value 0 in its *LEN* field. The *MAX* field contains the maximum value of the *LEN* fields among all the nodes in the subtree rooted at that node. This actually indicates the widest one among all the k-dense corridors stored in the subtree rooted at that node.

Apart from the child pointers, each node of the tree is attached with three more pointers.

Parent-pointer: It helps in traversing the tree from a node towards its root. The parent-pointer of the root node points to the corresponding element in the *level*-structure.

Neighbor-pointer: It points to the in-order successor of the node.

Self-indicator: As a vertex of the arrangement appears in two consecutive levels, say θ and $\theta+1$, it appears in both $\mathscr{T}_\theta$ and $\mathscr{T}_{\theta+1}$, and each of them is connected to the other one using this pointer.

By Observation 1 and succeeding discussions, a *type-A* k-dense corridor corresponds to a vertex of the arrangement. A vertex $v \in \mathscr{A}(H)$ appearing in levels, say θ and $\theta+1$, may correspond to at most two k-sticks (corresponding to two different *type-A* k-dense corridors); one end point of both of these *k-sticks* are positioned at vertex v, and their other end points lie on some edge at levels $\theta-k-1$ (if $\theta-k-1 > 0$) and $\theta+k+2$ (if $\theta+k+2 < n$), respectively. These *k-sticks* are attached to the vertex v appearing in two consecutive levels, θ and $\theta+1$. An edge e appearing at level θ stores at most one *k-stick* which is defined by it and another edge in the $(\theta-k-1)$th level which appears vertically below the edge e.

Change-list: After the addition of a new point p in S, its dual line $h=\mathscr{D}(p)$ is inserted in $\mathscr{A}(H)$ to get an updated arrangement $\mathscr{A}(H')$, where $H'=H\cup\{h\}$. This may cause redefining the *k-sticks* of some vertices and edges of $\mathscr{A}(H')$. In order to store this information, we use a linear link list at each level θ of the *level*-structure. Each element of this list is a tuple (ℓ,r). Here ℓ and r point to two elements (vertex/edge) at level θ, and the tuple (ℓ,r) represents a set of consecutive elements (vertices/edges) in $\mathscr{T}_\theta$ such that the *k-sticks* defined by all the vertices and edges in that set have been redefined due to the appearance of the new line h in the dual plane. Note that, the list attached to a particular level, say θ, of the arrangement may contain more than one tuple after computing the *k-sticks* for all the vertices and edges of $\mathscr{A}(H)$ which are affected by the inclusion of h. In that case, the set of elements represented by two distinct tuples, say (ℓ_1,r_1) and (ℓ_2,r_2) in that list must be disjoint, and the elements represented by r_1 and ℓ_2 must not be consecutive in $\mathscr{T}_\theta$.

$\mathscr{L}$-list: A linear link list created during the processing of an insertion/deletion of a line h in the arrangement $\mathscr{A}(H)$. This contains the vertices and edges of $\mathscr{A}(H')-\mathscr{A}(H)$ in a left-to-right order.

4.2. Algorithm for the insertion of a new point

Let p be a new point added in S, and $h=\mathscr{D}(p)$. Below we explain the different steps of updating the *level*-structure and computation of the widest k-dense corridors in the changed scenario.

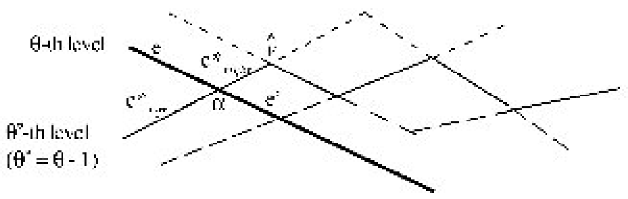

Fig. 3. Processing of a new vertex of $\mathscr{A}(H')$.

We first compute the leftmost intersection point of the newly appeared line h with the existing lines in H by comparing all the lines. Let the intersection point be α and the corresponding line be h_i. In order to find the edge $e^* \in \mathscr{A}(H)$ on which α lies, we traverse along the line h_i from its left unbounded edge towards the right in the tree, say $\mathscr{T}_{\theta*}$, in which the edge e^* belongs. The *neighbor-pointers* and *self-indicators*, attached to the nodes, facilitate this traversal.

Next, we use *parent-pointers* from the edge e^* up to the root of $\mathscr{T}_{\theta*}$ and finally, from the parent-pointer of the root, we may reach the θ^*th element of the *level*-structure. The level of the left unbounded edge e on the newly inserted line h in the updated arrangement $\mathscr{A}(H')$ will be θ ($= \theta^*$ or $(\theta^* + 1)$) depending on whether h intersects e^* from below or from above.

We update the old *level*-structure by copying it in a new array of size $n + 1$, and inserting a new level corresponding to the left unbounded edge e in the appropriate place. The *list-ptr* for all the levels are initialized to NULL. This step requires O(n) time.

4.2.1. Updating of $\mathscr{T}$

Let the level of e (the unbounded portion of h to the left of α) be θ in $\mathscr{A}(H')$, and the edge $e^*(\in \mathscr{A}(H))$, which is intersected by h, be at level $\theta^*(=\theta - 1$ or $\theta + 1)$. We now describe the creation of the new edges and vertices generated due to the inclusion of h in $\mathscr{A}(H)$.

The portions of h to the left and right of α are denoted as e and e', respectively, and the portions of e^* to the left and right of the point α by e^*_{left} and e^*_{right}, respectively. Note that, the vertex α appears in both the levels θ and θ^*, we need to make the following changes in $\mathscr{T}(\theta)$ and $\mathscr{T}(\theta^*)$ for the inclusion of the new vertex α and its adjacent edges in the *level*-structure. Refer to Fig. 3.

e and the vertex α are added in $\mathscr{T}_\theta$. e^*_{left} remains in its previous level θ^*, so e^* is replaced by e^*_{left} in $\mathscr{T}_{\theta^*}$. e^*_{right} goes to level θ. So, first of all e^*_{right} is added to $\mathscr{T}_\theta$.

Let $\hat{v}$ be the vertex at the right end of e^* (recently modified to e^*_{left}) in $\mathscr{T}_{\theta^*}$. The tree $\mathscr{T}_{\theta^*}$ is split into two height balanced trees, say $\mathscr{T}_{\text{R}}$ and $\mathscr{T}_{\text{L}}$, where $\mathscr{T}_{\text{R}}$ contains all the elements (vertices and edges) in that level to the right of $\hat{v}$ including itself, and $\mathscr{T}_{\text{L}}$ contains all the elements in the same level to the left of e^*_{left} and including itself. This requires O(log n) time [6].

Next, we concatenate $\mathscr{T}_{\mathrm{R}}$ to the right of e^*_{right} in $\mathscr{T}_\theta$. The *neighbor-pointer* of e^*_{right} is immediately set to point $\hat{v}$, and the *parent-pointers* of the affected nodes are appropriately adjusted. This can be done in $O(\log n)$ time [6].

Finally, the vertex α is added in $\mathscr{T}_{\mathrm{L}}$ as the rightmost element, and $\mathscr{T}_{\mathrm{L}}$ is renamed as $\mathscr{T}_{\theta^*}$. Note that a portion of e' will be the right neighbor of the vertex α in $\mathscr{T}_{\theta^*}$. Now, if we have already considered all the n newly generated vertices, the right end of e' will be unbounded. In that case, e' is added in T_{θ^*} as the rightmost element. Otherwise, the right end of e' is yet to be defined, and its addition in $\mathscr{T}_{\theta^*}$ is deferred until the detection of the next intersection. In the former case, the updating of the data structure is complete; however, in the latter case, we proceed with e', the portion of h to the right of α.

Now two important things need to be mentioned:

- For all newly created edges/vertices, we set the width of the k-dense corridor to 0. They will be computed afresh after the update of the *level*-structure.
- During this traversal, we create the $\mathscr{L}$-list with all the newly created edges and vertices on h in a left-to-right order. The edges are attached with their corresponding levels. As the newly created vertices appear in two consecutive levels, they show their lower levels in the $\mathscr{L}$-list.

Lemma 1. *The time required for constructing $\mathscr{A}(H')$ from the existing $\mathscr{A}(H)$ is $O(n \log n)$ in the worst case.*

4.2.2. Computing the new k-dense corridors

We now describe the method of computing all the *k-sticks* which intersect the newly inserted line h. The $\mathscr{L}$-list contains the pieces of the line h separated by the vertices in $(\mathscr{A}(H')-\mathscr{A}(H))$, which will guide our search process. We process edges in the $\mathscr{L}$-list one by one from left to right. For each edge $e \in \mathscr{L}$-list, we locate all the vertices and edges of $\mathscr{A}(H')$ whose corresponding *k-sticks* intersect e.

We proceed with an array of pointers P of size $2k+3$, indexed by $-(k+1),\ldots,0,\ldots,(k+1)$. While processing an edge $e \in \mathscr{L}$-list, $P(0)$ points to the edge e; $P(-1),\ldots,P(-k-1)$ point to $k+1$ edges below the left end vertex of e and $P(1),\ldots,P(k+1)$ point to $k+1$ edges above the left end vertex of e.

In order to evaluate all the *k-sticks* intersecting e and having its bottom end at level i $(i=\theta-k-1,\ldots,\theta)$, we need to consider the pair of levels $(i,i+k+1)$. Consider an x-monotone polygon bounded below (resp. above) by the x-monotone chain of edges and vertices at level i (resp. $i+k+1$) and by two vertical lines at the end points of e (see Fig. 4). This can easily be detected using the pointers $P(i-\theta)$ and $P(i+k+1-\theta)$. The *LEN* fields of all the edges and vertices on the above two x-monotone chains are initialized to zero. We draw vertical lines at each vertex of the upper and lower chains, which split the polygon into a number of trapezoids.

Each of the vertical lines drawn from the convex vertices defines a *type-A k-stick*. Its *dual length* is put in the *LEN* field of the corresponding node of $\mathscr{T}_i$ or $\mathscr{T}_{i+k+1}$.

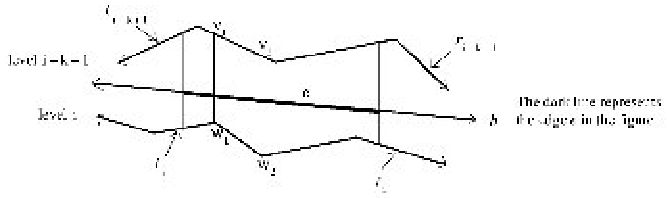

Fig. 4. Computation of *type-A* and *type-B* corridors while processing an edge $e \in \mathscr{L}$-list.

In order to compute the *type-B k-sticks*, we consider each of the vertical trapezoids from left to right. Let $\Delta = v_1v_2w_2w_1$ be such a trapezoid whose v_1w_1 and v_2w_2 are two vertical sides, I denotes the x-range of Δ. Let v_1v_2 be a portion of an edge e^*, which in turn lies on a line $h^* \in H'$, and w_1w_2 lies on $h^{**} \in H'$. We compute $-1/x(h^* \cap h^{**})$ and check whether it lies in I. If so, the vertical line at $x = -1/x(h^* \cap h^{**})$, bounded by v_1v_2 and w_1w_2, indicates a *type-B k-stick* corresponding to the edge e^*. We compute its *dual length*; this newly computed *k-stick* replaces the current one attached with e^* provided the *dual length* of the newly computed *k-stick* is greater than the *LEN* field attached with e^* in the data structure $\mathscr{T}_i$.

Let ℓ_i and r_i (resp. ℓ_{i+k+1} and r_{i+k+1}) denote the edges at level i (resp. $i+k+1$), which are intersected by the vertical lines at the left and right end points of the edge $e(\in \mathscr{L}$-list), respectively. Note that, the definition of *k-sticks* for the edges and vertices of the ith level between ℓ_i and r_i may have changed due to the presence of $e \in \mathscr{A}(H')$. So, we need to store the tuple (ℓ_i, r_i) in the *change-list* attached to level i of the *level*-structure. But, before storing it, we need to check the last element stored in that list, say (ℓ^*, r^*). If the neighbor-pointer of r^* points to ℓ_i in $\mathscr{T}_i$, then (ℓ^*, r_i) is a continuous set of elements in level i which are affected due to the insertion of h. So, the element (ℓ^*, r^*) of the $\mathscr{L}$-list attached to level i, is updated to (ℓ^*, r_i); otherwise, the new tuple (ℓ^*, r^*) is added in the *change-list*. We store $(\ell_{i+k+1}, r_{i+k+1})$ in the *change-list* attached to level $i+k+1$ of the *level*-structure in a similar way.

At the end of the execution of edge e, if e is right unbounded, our search stops; otherwise we proceed by setting $P(0)$ to the next edge e' of $\mathscr{L}$-list. Each of the pointers $P(i)$, $i = -k-1, \ldots, k+1$, excepting $P(0)$, needs to point to an edge either at level $(i+\theta-1)$ or at level $(i+\theta+1)$ which lies just below or above the current edge pointed by $P(i)$ in the *level*-structure, depending on whether e' lies at level $\theta - 1$ or $\theta + 1$ in the *level*-structure. From the adjacencies of vertices and edges in the *level*-structure (maintained using *neighbor-pointers*), this can be done in constant time for each $P(i)$.

Lemma 2. *The time required for computing the k-sticks intersecting the line h in* $\mathscr{A}(H')$ *is* $O(nk)$.

Proof. Follows from the above discussions, and the fact that the complexity of the $\leqslant k$-levels of n half-lines lying above (below) the newly inserted line h in $\mathscr{A}(H')$ is $O(nk)$ [7]. □

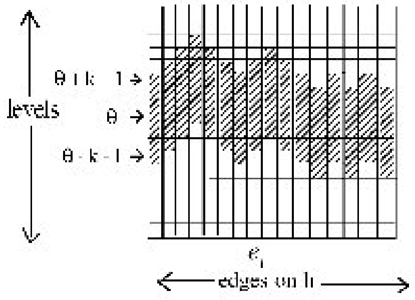

Fig. 5. Grid $\mathscr{M}$ estimating the number of elements in the *change-list*.

4.2.3. Location of the widest k-dense corridor

The *change-list* for a level, say θ, created in the earlier subsection, is used to update the *MAX* fields of the nodes of the tree $\mathscr{T}_\theta$, by considering its elements from left to right. Let the tuple (ℓ, r) be an entry of the *change-list* at level θ. Let q be the common predecessor of the set of nodes represented by the tuple (ℓ, r). Let P_{IN} be a path from the root of $\mathscr{T}_\theta$ to the node q, and P_{L} and P_{R} be two paths from q to ℓ and q to r, respectively. In $\mathscr{T}_\theta$, the *MAX* fields of all the nodes in the interval (ℓ, r), and the set of nodes in P_{IN}, P_{L} and P_{R} may be changed. So, they need to be inspected in order to update the *MAX* fields of the nodes in $\mathscr{T}_\theta$. Now we have the following lemma.

Lemma 3. *For each entry (ℓ, r) of the change-list of a level, say θ, the number of nodes of $\mathscr{T}_\theta$ which need to be visited to update the MAX fields is* $O(\log n + \chi)$, *where χ is the number of consecutive vertices and edges of the arrangement at level θ represented by (ℓ, r).*

In order to count the total number of elements attached to the *change-list* at all the levels let us consider an $n \times n$ square grid $\mathscr{M}$ whose rows represent the n levels of the arrangement and each of its columns represents an edge on h in a left-to-right order (see Fig. 5). Consider the shaded portion of the grid; observe that its ith column spans from row $\theta-k-1$ to $\theta+k+1$, where θ is the level of e_i in $\mathscr{A}(H')$. This corresponds to the levels which are affected by e_i. The shaded region is bounded by two x-monotone chains. Now, let us define a *horizontal strip* as a set of consecutive cells on a row which belong to the shaded portion of the grid. A horizontal strip which spans from the first to the last column of the grid, is referred to as *long* strip. The strips which are not *long*, are called *short* strips. Note that, each strip attached to a row represents an element of the *change-list* attached to the corresponding level. It is easy to observe that the number of such *short strips* is $O(n)$, and the number of such *long strips* may be at most $2k-3$ in the worst case. Now we have the following lemma.

Lemma 4. *The time required for locating the longest k-stick in $\mathscr{A}(H')$ is* $O(nk + n\log n)$ *in the worst case.*

Proof. By Lemma 3, the tree traversal time for all the entries of the *change-lists* of all the levels is equal to $\sum_{i=1}^{n}\sum_{j=1}^{m_i}(\chi_{ij}+\log n)$, where m_i is the number of entries in the *change-list* of the ith level, and χ_{ij} is the length of the strip represented by the jth entry in the *change-list* of the ith level. Now the lemma follows from the following two arguments:

- the number of *k-sticks* $\mathscr{K}$, which are newly defined due to the inclusion of h in $\mathscr{A}(H)$, is $\sum_{i=1}^{n}\sum_{j=1}^{m_i}\chi_{ij}=\mathrm{O}(nk)$ (by Lemma 2), and
- as discussed above, $\sum_{i=1}^{n} m_i$ is equal to the total number of long and short strips in the matrix $\mathscr{M}$, which may be $\mathrm{O}(n)$ in the worst case. □

Finally, the roots of the trees at all the levels need to be inspected to determine the widest k-dense corridor.

4.3. Complexity

Lemmas 1, 2 and 4 lead to the main result of this work stated in the following theorem.

Theorem 2. *Addition of a new point requires*
(a) $\mathrm{O}(n\log n)$ *time for updating the data structure.*
(b) $\mathrm{O}(nk)$ *time to compute the k-dense corridors containing that point.*
(c) $\mathrm{O}(nk+n\log n)$ *time to report the widest k-dense corridor.*

As we are preserving all the vertices and edges of the arrangement of the dual lines in our proposed *level*-structure, the space complexity is $\mathrm{O}(n^2)$, where n is the number of points on the floor at the current instant of time.

5. Related problems

In this section, we study two different schemes for the *query problem*, considered in [7]. Here, given a query point q, the objective is to report the widest k-dense corridor containing it.

5.1. Scheme-1

Let us assume that the *level*-structure is already available in the dual plane. As in the previous problem, we find intersections of the line $h=\mathscr{D}(q)$ with the other lines in H. These pieces of h intersected by the lines in the arrangement $\mathscr{A}(H)$ are stored in the $\mathscr{L}$-list along with their levels in a left-to-right order. Next, we proceed through the arrangement $\mathscr{A}(H)$ to compute the set of vertices and edges whose corresponding *k-sticks* intersect h, as described in Sections 4.2.2 and 4.2.3. Note that, as h need not be inserted in $\mathscr{A}(H)$ here, $\mathscr{T}$ may be represented using linear link lists, instead of height-balanced binary trees, for storing the vertices and edges in different levels of the arrangement. Thus an $O(\log n)$ time is saved while processing each edge of $\mathscr{L}$-list. Moreover, we need to report only the *k-stick* having maximum dual length among the members of the aforementioned set, which can be done in time proportional to $\mathscr{K}$, where $\mathscr{K}$ is the combinatorial complexity of ($\leqslant k$)-*level* of an arrangement of n half-lines, each of them belonging to and touching the same side of the line $\mathscr{D}(q)$, the dual line of the point q. As $\mathscr{K}$ may be $O(nk)$ in the worst case, we have the following theorem:

Theorem 3. *Given a set of points, we can maintain an* $O(n^2)$ *space data structure, which can report the widest k-dense corridor containing a query point q in* $O(nk)$ *time in the worst case.*

5.2. Scheme-2

As the storage requirement of Scheme-1 is $O(n^2)$, it seems worthwhile not to maintain the arrangement $\mathscr{A}(H)$ permanently. Below we describe a divide-and-conquer strategy which constructs a data structure afresh for each query, in $O(nk\log(\lfloor n/k\rfloor))$ time and $O(nk)$ space.

We have the set $H=\{h_1,h_2,\ldots,h_n\}$ of lines in the dual plane, where $h_i=\mathscr{D}(p_i)$. Let $H'=\{h'_1,h'_2,\ldots,h'_n\}$ denote the set of half lines, where h'_i is the portion of h_i above $h=\mathscr{D}(q)$. Similarly, $H''=\{h''_1,h''_2,\ldots,h''_n\}$ is defined below the line $h=\mathscr{D}(q)$. As in [7], we also denote the ($\leqslant k$)-*level* above (resp. below) h by $L_{\leqslant k}(H',h^+)$ (resp. $L_{\leqslant k}(H'',h^-)$). Below we describe a divide-and-conquer approach to construct $L_{\leqslant k}(H',h^+)$.

First, we split H' into two parts H'_1 and H'_2 of size at most $\lceil n/2\rceil$. Then compute $L_{\leqslant k}(H'_1,h^+)$ and $L_{\leqslant k}(H'_2,h^+)$ separately. Next, we construct $L_{\leqslant k}(H',h^+)$ by overlaying $L_{\leqslant k}(H'_1,h^+)$ and $L_{\leqslant k}(H'_2,h^+)$. The time of computing the overlay of two simply connected planar subdivisions is $O(m_1+m_2)$ [3], where m_1 is the total number of edges in the two subdivisions, and m_2 is the number of intersections that occur among the edges of different subdivisions during overlay. Note that, $O(m_1+m_2)$ is the complexity of $L_{\leqslant k}(H',h^+)$, which is $O(nk)$. Thus, if $T(n)$ denotes the running time of the algorithm for a set of n half-lines, we have the following recurrence:

$$T(n)=\begin{cases} O(k^2) & \text{if } n\leqslant k,\\ 2T(n/2)+O(nk) & \text{if } n>k. \end{cases}$$

The solution of the above recurrence is $O(nk\log(\lfloor n/k\rfloor))$. Similarly, $L_{\leqslant k}(H'',h^-)$ is constructed. Surely, the space requirement of this algorithm is the total complexity of $L_{\leqslant k}(H',h^+)$ and $L_{\leqslant k}(H'',h^-)$, which is $O(nk)$. Next, we compute all the *k-sticks* from $L_{\leqslant k}(H',h^+)$ and $L_{\leqslant k}(H'',h^-)$) using the method described in Section 4.4. The time required for this step is $O(nk)$. Thus we have the following theorem:

Theorem 4. *Given a set of n points, we can report the widest k-dense corridor containing a query point q in* $O(nk\log(\lfloor n/k\rfloor))$ *time and using* $O(nk)$ *space.*

References

[1] S. Chattopadhyay, P.P. Das, The *k*-dense corridor problems, Pattern Recognitions Lett. 11 (1990) 463–469.

[2] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer, Berlin, 1987.

[3] U. Finke, K. Hinrichs, Overlaying simply connected planar subdivisions in linear time, Proc. 11th. ACM Symp. on Computational Geometry (1995) 119–126.

[4] M. Houle, A. Maciel, Finding the widest empty corridor through a set of points, Report SOCS-88.11, McGill University, Montreal, Quebec, 1988.

[5] R. Janardan, F.P. Preparata, Widest-corridor problems, Nordic J. Comput. 1 (1994) 231–245.

[6] D. Knuth, The Art of Computer Programming: Sorting and Searching, Addison-Wesley, Reading, MA, 1973.

[7] C.-S. Shin, S.Y. Shin, K.-Y. Chwa, The widest *k*-dense corridor problems, Inform. Process. Lett. 68 (1998) 25–31.