Efficient motion planning for an L-shaped object

D. Halperin, M.H. Overmars and M. Sharir

RUU-CS-88-38
December 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

Efficient motion planning for an L-shaped object

D. Halperin, M.H. Overmars and M. Sharir

Technical Report RUU-CS-88-38
December 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

Efficient motion planning for an L-shaped object

D. Halperin[(1)] M. H. Overmars[(2)]
M. Sharir[(1,3)]

(1) School of Mathematical Sciences, Tel-Aviv University

(2) Department of Computer Science, University of Utrecht

(3) Courant Institute of Mathematical Sciences, New-York University

December 5, 1988

Abstract

We present an algorithm that solves the following motion-planning problem. Given an L-shaped body L and a 2-dimensional region with n point obstacles, decide whether there is a continuous motion connecting two given positions and orientations of L during which L avoids collision with the obstacles. The algorithm requires $O(n^2 \log^2 n)$ time and $O(n^2)$ storage. The algorithm is a variant of the cell-decomposition technique of the configuration space ([SS, LS]) but it employs a new and efficient technique for obtaining a compact representation of the free space, which results in a saving of an order of magnitude. The approach used in our algorithm seems applicable to motion-planning of certain robotic arms whose spaces of free placements have a structure similar to that of the L-shaped body.

[0]Work on this paper by the first and the third authors has been supported in part by Office of Naval Research Grant N00014-87-K-0129, by National Science Foundation Grant No. NSF-DCR-83-20085, and by grants from the Digital Equipment Corporation, the IBM Corporation, the U.S.-Israeli Binational Science Foundation, the NCRD - the Israeli National Council for Research and Development, and the Fund for Research in Electronics, Computers and Communication of the Israeli Academy of Sciences and Humanities.

1 Introduction

Let B be a robot system having k degrees of freedom and free to move within a two- or three-dimensional domain V which is bounded by various obstacles whose geometry is known to the system. The motion-planning problem for B is, given the initial and desired final position of the system B, to determine whether there exists a continuous motion from the initial position to the final one, during which B avoids collision with the known obstacles, and if so, to plan such a motion.

Since the general motion planning problem is very hard, a significant effort was devoted to develop efficient algorithms for some special cases, particularly that of motion planning for rigid objects in a 2-D polygonal space. To get some feeling of what "efficient" means in this context, we note that such moving systems B have three degrees of freedom, so their *configuration space*, i.e., the space of parametric representations of placements of B, is 3-dimensional. Let n denote the number of obstacles corners, and suppose that the complexity of B is constant. Then the "free" portion FP of the configuration space, consisting of placements of B in which it does not meet any obstacle, is bounded by $O(n)$ (algebraic) *collision-constraint* surfaces, each being the locus of placements where some specific feature of B makes contact with some specific obstacle feature. By standard arguments from algebraic geometry, the complexity of FP is $O(n^3)$. Moreover, the recent general technique of Canny [Ca] yields a (fairly complicated) algorithm that computes a discrete representation of FP in time $O(n^3 \log n)$. A more specialized algorithm has recently been obtained by Avnaim, Boissonnat and Faverjon [ABF], whose complexity is also $O(n^3 \log n)$.

Thus the general goal of studying motion planning problems for rigid objects in the plane is to obtain subcubic, and ideally near-quadratic, algorithms. A quadratic lower bound for the actual combinatorial complexity of a collision-free motion for a line segment has been given by Ke and O'Rourke ([KeO]). However, the only cases where near-quadratic algorithms have been obtained involve *convex* objects (see [OY] for the case of a disc, [LS, SiS] for the case of a line segment, and [KS2] for the case of an arbitrary convex polygon).

Efficient motion planning algorithms for non-convex polygons in 2-D polygonal space have scarcely been dealt with. An early consideration of the problem appears in Schwartz and Sharir [SS], where they extend their $O(n^5)$ projection algorithm for

moving a line-segment to a non-convex polygon B with similar running time, assuming that B has a fixed number of vertices. As just noted, the algorithms of [Ca] and of [ABF] already improve the complexity to $O(n^3 \log n)$, but no better solutions were known in the non-convex case.

In this paper we present a new approach to motion planning of non-convex objects in the plane. We obtain a near-quadratic algorithm for solving this problem in the special case of an L-shaped object moving amidst a collection of point obstacles in the plane.

We distinguish between the *reachability problem* which is to check whether a continuous collision-free path from the initial to the final placement exists, and the *find-path problem* which is to actually compute such a path if it exists. The former is the concern of this paper, and the latter will be discussed in an accompanying paper.

Our algorithm uses the *decomposition* approach to motion planning ([SS, LS, KS2]) which partitions the space FP of free placements of the robot system into a finite number of simple connected cells. These cells define vertices in a so-called *connectivity graph* CG. Two cells are adjacent in CG if they have a common boundary enabling a direct crossing of the moving object between them. It can be shown that in the worst case, the space FP for our L-robot has $\Omega(n^3)$ connected components, thus, in particular, its total combinatorial complexity can be $\Omega(n^3)$.

(To see this, consider Fig. 1, where there are three sets of $n/3$ points each. Choose an interval between two successive points in the upper horizontal set, choose an interval between two successive points in the lower horizontal set. Now locate the "vertical" bar of L so that it will intersect the two chosen intervals. Finally, choose a pair of consecutive points in the vertical set and locate the "horizontal" bar of L such that it will also intersect the interval between this pair. It is easily verified that there are $\Omega(n^3)$ such choices and that L cannot continuously move between any pair of such placements, if the size of L and the location of the points are chosen in an appropriate manner.)

However, using some interesting data structure techniques, we construct an implicit representation of FP using a compact connectivity graph that requires subcubic space and can be constructed in subcubic time. To be precise, our reachability algorithm requires $O(n^2 \log^2 n)$ time and $O(n^2)$ space.

The paper is organized as follows. In Section 2 we introduce the terminology,

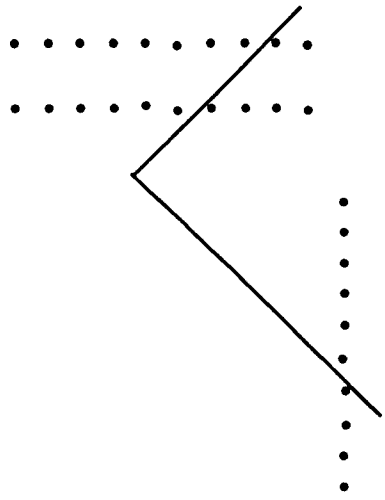


Figure 1: A configuration of obstacles with an $\Omega(n^3)$-size FP

explore the basic ideas, derive some initial observations about the problem structure, and give an overview of our efficient solution. In Section 3 we describe the data structures that support our algorithm. More details concerning the algorithm are described in Section 4, where we also prove its correctness and analyze its complexity. We conclude in Section 5 by discussing the novel ideas used in this paper and their potential applicability to other instances of the motion planning problem, and by proposing some directions for further research.

2 Terminology and Initial Analysis

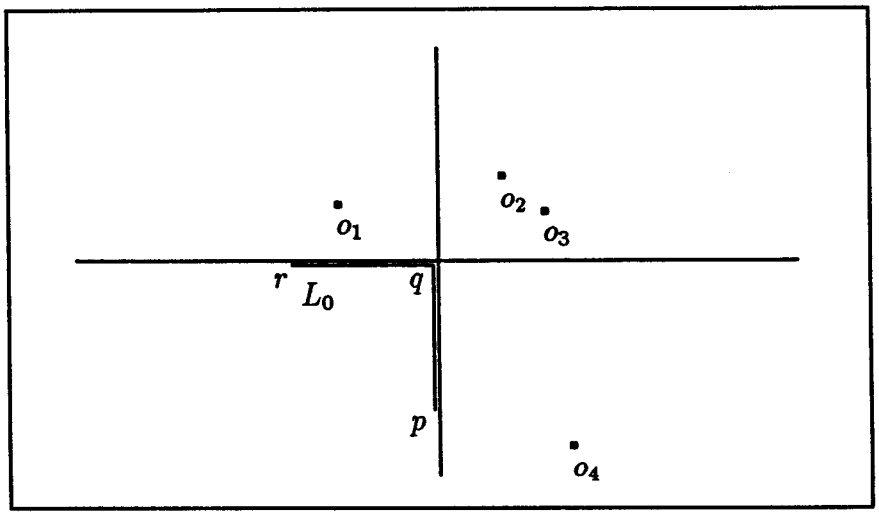
We denote the moving object L by pqr where $\overline{qp}$ is the vertical bar and $\overline{qr}$ is the horizontal bar of L (in some standard axis-parallel position). Thus p and r are the *external vertices* and q is the *internal vertex* of L . Each position of L can be specified as $Z = (X, \theta)$ where X is the position of q and θ is the orientation of $\overline{qp}$. For the purpose of our algorithm, we will always present X in a rotated coordinate frame in which θ becomes an upward vertical direction. We denote by AP the resulting three-dimensional space of all positions of L, which can be identified with $R^2 \times S^1$.

The set of point-obstacles is denoted by $O = \{o_i | i = 1, 2, \ldots, n\}$. For the sake of representation only, we will delimit the planar workspace V in which L is free to move by a sufficiently large rectangle, and assume that this rectangle rotates with the coordinate system, so that it always remains axis-parallel.

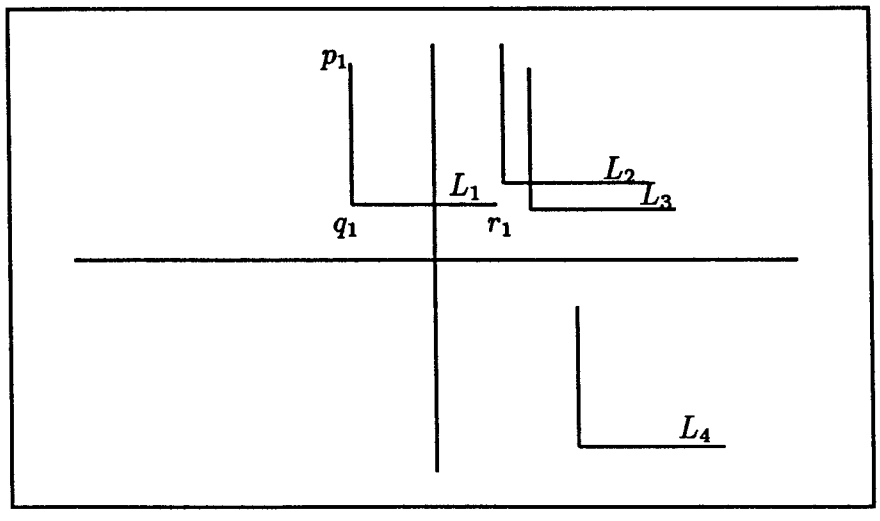
We shall call a position of L at which it does not touch any point a *free position.* The set FP of all free positions of L is an open three-dimensional submanifold of AP.

Similar to [LS] (see also [SS, KS2]) our method decomposes FP into connected subcells of a simple form, using the following two-step approach. First we consider the case in which L is allowed to translate arbitrarily but not to rotate. Then we consider the case in which L is allowed both to translate and to rotate in V.

To build a discrete representation of FP for a fixed orientation, we follow a common practice in motion-planning ([LW, KS1]), and compute the Minkowski (i.e., vector) difference of each obstacle and the robot. Let L_0 denote a position of L in a Cartesian coordinate system in which q coincides with the origin of the system, p lies on the negative y-axis and r lies on the negative x-axis. For any fixed θ, and for $i = 1, 2, \ldots, n$, let $L_i = L_i(\theta) = o_i(\theta) - L_0$ be the Minkowski difference of the obstacle o_i and L_0, in the rotated coordinate frame where θ points upwards; here $o_i(\theta)$ denotes the rotated position of the point obstacle o_i in this coordinate frame. The horizontal bar of L_i extends from $o_i(\theta)$ to the right, and the vertical bar extends from $o_i(\theta)$ upwards (see Fig. 2 for an illustration). Let $S_\theta = \{L_i(\theta) | i = 1, 2, \ldots, n\}$ for some fixed orientation θ. S_θ defines a planar arrangement consisting of horizontal and vertical line segments. The cross-section of FP at a given θ, which we denote by FP_θ, is simply the complement of the union of the $L_i(\theta)$'s. In addition, to simplify the structure of S_θ, we add to it certain horizontal segments emanating from the vertices of the L_j's. We will refer to these extensions as *imaginary walls*; see Fig. 5(a) for an illustration (a formal definition is given below). S_θ together with these extensions (and the big rectangle enclosing the workspace) divide FP_θ into orthogonal (axis-parallel) simply-connected polygons. We call each such polygon a *face* of FP_θ . Some of these faces are simply rectangles, in which case we call the four edges of the face the *northern, southern, western* and *eastern walls* corresponding to the upper, lower, left and right edges of that face. In other cases we call the rightmost edge of the face the *eastern wall.* The imaginary walls will be defined in such a way to ensure that each face has a unique eastern wall.



(a) $L_0,\ o_1 - o_4$



(b) $L_1 - L_4$

Figure 2: The expanded obstacles

If we let θ vary through the range $[0, 2\pi]$, the object $L_i(\theta)$ will trace a 2-D surface σ_i within AP; that is, $\sigma_i = \{(X, \theta) | \theta \in [0, 2\pi], X \in L_i(\theta)\}$. The collection of these surfaces forms an arrangement $\mathcal{A}$ of surfaces in AP, which decomposes the 3-dimensional space AP into pairwise disjoint connected cells, each of 0,1,2 or 3 dimensions. We shall use the unquantified term *cell* for a 3-dimensional cell of $\mathcal{A}$. A cell c of $\mathcal{A}$ is *interesting* if at least one cross-section of its closure $\bar{c}$ contains a vertex of some $L_i(\theta)$. All other cells of the arrangement are called *dull* (this terminology is borrowed from [AS]).

Remark 2.1 . Formally speaking, the surfaces σ_i are not algebraic, because they depend trigonometrically on θ. However, they can be easily made algebraic if one replaces θ by, say $t = \tan\frac{\theta}{2}$. For simplicity of exposition we will continue to use θ as one of our coordinates, but consider this transformation as available whenever needed.

In our analysis we will sometimes allow the motion of L to become *semifree*, namely allow L to touch an obstacle. In the configuration space, the corresponding path is allowed to touch the union BFP of the surfaces σ_i. However, the path is not allowed to cross BFP transversally (which amounts in the physical space to L sweeping through an obstacle). We will therefore consider each σ_i to be "two-sided", where each side bounds, and can be reached from, a different portion of FP, and where no direct crossing from one side to the other is allowed.

How does the 2-dimensional arrangement (of S_θ) change as the orientation θ of the original L varies? As the coordinate system rotates, FP_θ changes continuously, but its combinatorial structure remains unchanged, unless one of the following two types of critical events occurs at θ:

I. A vertex of one L_i meets an edge of another L_j.

II. Two parallel edges of two L_i's overlap.

These two types of events are substantially different in their effect on FP_θ . An event of type I has only a local effect on FP_θ — either a face is split into two, or two faces are merged into one. When an event of type II occurs, its effect is more global and causes four contiguous sets of faces, each set containing $O(n)$ (and, in the worst case, $\Omega(n)$) faces, to change some of their edges, to disappear, or to newly appear.

An orientation θ at which one of these events happens, will be called a *critical orientation.*

Remark 2.2 . When two parallel edges overlap it is also the case that a vertex of one expanded obstacle meets an edge of another. We will treat this part of the type II event in a way similar to a type I event. Indeed, it will be more suitable to regard each overlap of two parallel edges as a combination of events of type I and of type II. From this point on type II will refer only to the effect of the overlap on the "internal" faces involved, taking care of the "external" faces (i.e., the faces adjacent to the vertices of the overlapping edges) as effected by a type I event; see below for more details.

How many criticalities are there of each type?

Lemma 2.1 *The number of critical orientations induced by a fixed pair of obstacles is bounded by a constant.*

Proof. Trivial, by elementary geometric considerations. □

Since there are $\binom{n}{2}$ pairs of expanded obstacles, it follows that:

Corollary 2.2 *There are $O(n^2)$ critical orientations of type I and $O(n^2)$ critical orientations of type II; they can all be easily calculated in $O(n^2)$ time.*

An Overview of the Reachability Algorithm

We now give a first rough description of our algorithm for testing whether two given placements of L can be reached from one another by a continuous collision-free motion.

The proposed algorithm consists of two parts:

1. Building the connectivity graph (preprocessing), and

2. Searching for a path from the initial placement of L to its destination placement.

Our goal in the preprocessing part is to build a compact and space-efficient version of the connectivity graph, in the sense that *(i)* it (almost) does not contain nodes representing dull cells, and *(ii)* it contains only partial information about interesting cells.

Since our goal is to obtain an algorithm with a close-to-quadratic performance, the main difficulty we face is the handling of type II events. The problem is that there are $O(n^2)$ criticalities of type II where each such criticality may involve up to $O(n)$ faces in the worst case. If we were to handle each face separately, we would end up with an algorithm having a worst-case $\Omega(n^3)$ performance. Instead, we wish to handle these $O(n)$ faces in an implicit manner, so that each type II event can be "encoded" into our data structures in only polylogarithmic time, in a manner that still allows us to trace these changes efficiently when needed. (Note in contrast that type I events are "harmless" — there are $O(n^2)$ such events and each induces only $O(1)$ changes in the combinatorial structure of FP.)

To achieve this, we make use of the following observations which are essential to the subsequent analysis:

Observation 1. If we *ignore the labeling* of the segments in S_θ then a type II criticality does not change the combinatorial structure of the planar arrangement of S_θ (again we remind the reader that in this statement we ignore the effects of this criticality on the endpoints of the corresponding L_i's, which are treated separately as type I events).

Observation 2. Taking the labeling of the segments in S_θ into account, each of the $O(n)$ adjacent faces f participating in a type II critical event goes only through one of the four following combinatorial changes (where we assume that the vertical bars of two expanded obstacles L_i, L_j overlap and that L_j moves westwards relative to L_i),

(i) f changes its eastern wall from L_j to L_i (see, e.g., f_1 in Fig. 3);

(ii) f changes its western wall from L_i to L_j (f_3 in Fig. 3);

(iii) f is "squashed" between L_i and L_j i.e., shrinks to a segment and then disappears (f_2 in Fig. 3); or

(iv) f newly appears, initially as a segment and then expanding into a rectangle bounded between L_i and L_j (and two other horizontal bars) (f_2' in Fig. 3).

Similar changes occur when two horizontal bars of some L_i, L_j overlap.

Our idea is to take advantage of these facts for obtaining an economical representation of FP. Informally, we decompose FP into connected *subcells.* Each subcell c_f corresponds to a face f in the *unlabeled* arrangement S_θ, and consists of the union of all the slices of FP that correspond to f, obtained as we vary θ in a maximal interval τ in which no type I criticality effects the structure of f and no type II criticality causes f to be squashed or disappear (as in (iii) above, or its θ-symmetric counterpart (iv)). That is $c_f = \{(X, \theta) : \theta \in \tau, X \in f_\theta\}$ where τ is as above and f_θ denotes the portion of FP_θ that corresponds to f. (Note that in this context f is represented in a purely combinatorial manner, by its "location" in the unlabeled arrangement S_θ. More details on this representation will be given in the following sections.)

Some of the subcells of FP are stored as the nodes of a *connectivity graph* CG, with edges connecting pairs of adjacent subcells, in the sense that they admit direct crossing between them either at a fixed θ through some imaginary wall, or by crossing a critical orientation (of type I) delimiting both subcells.

The main ideas of our algorithm are (a) to avoid representing dull cells as much as possible, and (b) to store each subcell of an interesting cell as a single entity as long as it is not effected by any type I change. Thus even though the walls of the cross-section of this subcell may change repeatedly due to type II events, we do not record these changes in the subcell. Instead, these changes are stored in a more global manner in certain auxiliary data structures, to be described below, and their "overall effect" is retrieved upon demand in an efficient manner.

Our algorithm proceeds roughly as follows. It starts with computing and sorting all the critical orientations. Then the FP cross-section FP_{θ_0}, for some initial non-critical orientation θ_0, is built. All the faces of FP_{θ_0} are marked *active* and the "initial layer" of the connectivity graph CG is laid, with a node corresponding to each face of FP_{θ_0} (more precisely to each subcell of FP whose cross-section at θ_0 is that face) and edges connecting pairs of faces (subcells) adjacent along some imaginary wall. As θ increases from θ_0 and criticalities occur, we do the following:

When a type I criticality occurs, we update the topological structure of

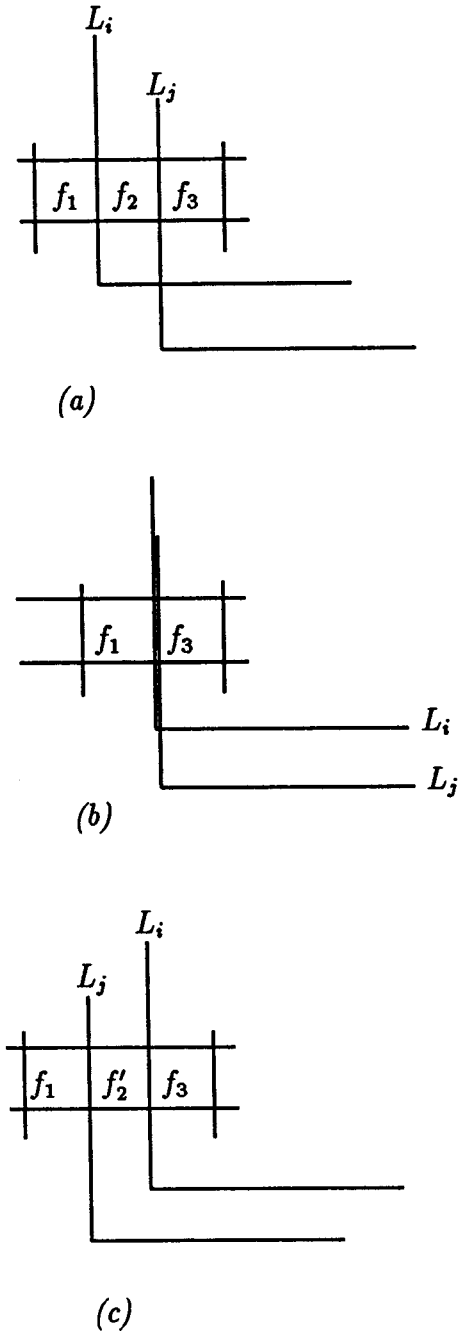


Figure 3: The four kinds of faces effected by a type II event

the current FP cross-section around the critical contact, add new nodes and edges to the connectivity graph to reflect this local change, and make all the faces that are involved in the change *active*.

At a type II critical orientation, we mark all the faces that get squashed in this event as being *inactive*. More precisely, each such face f is replaced in the current arrangement S_θ by a similar looking face f', in which a pair of opposite walls have been swapped. Even though f and f' occupy the same place in the (unlabeled) arrangement S_θ, there is clearly no direct passage between them. It is therefore important to record the fact that the corresponding face in S_θ is no longer *active*, meaning that it is as yet not connected to any *active* subcell (where a type I event occurs).

When a type I event occurs, it generates potentially new nodes (subcells) in the connectivity graph (corresponding to the faces that participate in this change). However, such a node could designate a subcell already represented by another node of CG (which has been created by a former type I event or at the initial θ_0, and which may have undergone a sequence of type II changes to reach the current face). Since these changes are not stored directly in the subcells, a major novel component of our algorithm is to determine, for a given type I contact, which faces of the unlabeled arrangement S_θ participate in it, and whether any of these faces is a portion of an already defined node of the connectivity graph. How this is achieved is described below.

More formally, we have

Definition. A face of the current cross-section FP_θ of FP is called *active* if the subcell containing it has already a representing node in CG ; otherwise, the face is called *inactive*.

We will describe below how this *active/inactive* information is handled efficiently.

In addition we also update, at each type II event, auxiliary global information, such as the horizontal ordering of the vertical bars of the L_i's, the vertical ordering of the horizontal bars, etc.

Reaching $\theta = \theta_0 + 2\pi$ we perform some additional work to "wrap" the connectivity graph around.

There are two additional events during the θ-sweep. These occur at the orientations θ_s and θ_d of the initial and final placements of L respectively. The discussion of what happens at these events is postponed to the full description of the algorithm in Section 4. We merely note that the introduction of these θ's enables us to locate the two nodes v_s and v_d of CG corresponding to the subcells containing the initial and final placements of L.

Since we create new nodes and edges in CG only at type I events, and each such event involves only $O(1)$ faces of FP_θ, the size of CG will be at most $O(n^2)$.

The reachability query is then treated by searching the connectivity graph for a path between v_s and v_d. If a path is found then the answer to the query is YES, otherwise the answer is NO. As already noted, this still falls short of producing the path when one exists. See below for a discussion of this issue.

For the convenience of the analysis we assume that no two critical orientations coincide. In particular, this requires that no three obstacle points be collinear and no two pairs of obstacle points lie on parallel lines. However, such degenerate cases can be handled by an appropriate and slight modification of the algorithm.

3 Data Structures

Throughout the algorithm we use the following data structures:

- CG — the connectivity graph
- The horizontal package — retaining all the necessary information about the current cross-section of FP from a "horizontal" point of view. The package consists of:

  Q_H — a balanced binary tree storing the n vertical bars of the L_i's in a left-to-right order;

  R_H — a left-to-right ordered list of $n+1$ segments trees, where each tree describes (certain horizontal bars intersecting) a vertical slab of FP_θ between two lines containing adjacent vertical bars of the L_i's;

  U_H — a data structure for answering queries of the form: "how many horizontal segments of S_θ are stabbed by a query vertical segment?"

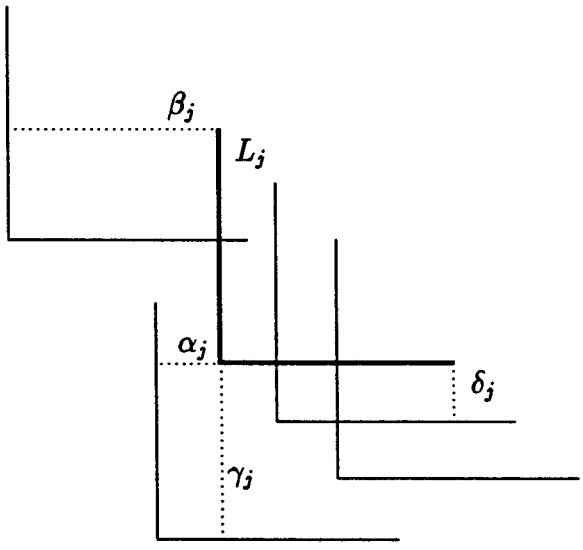


Figure 4: The imaginary walls

- The vertical package — storing all the necessary information about the current cross-section of FP from a "vertical" point of view; it consists of three substructures Q_V, R_V and U_V, defined in an analogous manner.

Let us now describe the structures in detail.

3.1 The connectivity graph

To better describe the structure of the connectivity graph, let us first be more precise about the imaginary walls. Four segments extend from each L_j (Fig. 4), each segment extends until it hits some orthogonal segment:

α_j — a westwards extension of the horizontal bar of L_j;

β_j — a westward-directed segment emanating from the upper external vertex p_j of L_j;

γ_j — a southwards extension of the vertical bar of L_j; and

δ_j — a southward-directed segment emanating from the right external vertex r_j of L_j.

However, to simplify our structures we will use only the walls α_j,β_j to decompose FP_θ into faces; γ_j and δ_j will be used only in some auxiliary data structures, as described below. With this convention, each face f of FP_θ has a unique eastern wall e_f, which is a connected interval of the vertical bar of some L_j, and f consists of all points z for which there exists a horizontal segment connecting z to a point on e_f whose relative interior does not meet any L_k (i.e., z can "see" e_f when looking directly eastwards). See Fig. 5(a) for an illustration of FP_θ.

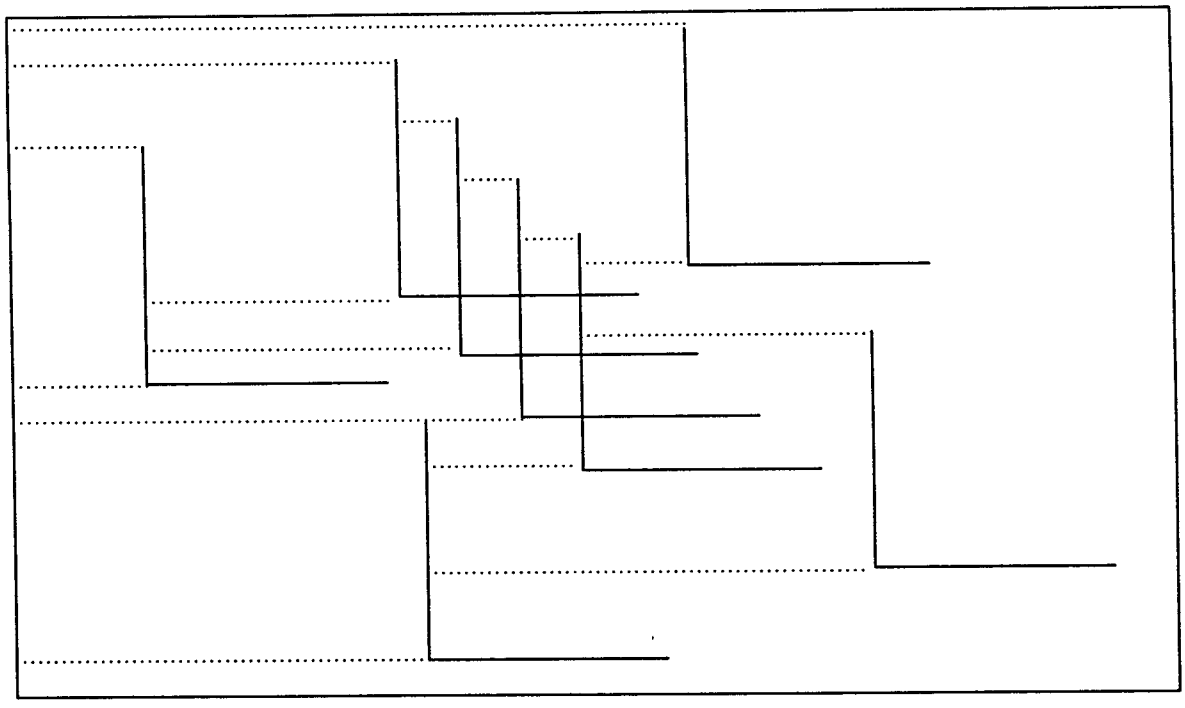
The connectivity graph $CG = (V, E)$ has a set of nodes V and a set of edges E. Each node $v \in V$ corresponds to a subcell c of FP. New nodes are created and added to CG only when the corresponding subcells participate in a type I change. A subcell can be characterized at each orientation θ by its eastern wall (in a manner to be made more precise below). However, this information is stored in the instantaneous data structures describing FP_θ and is not encoded into the nodes of the connectivity graph. CG describes, at a certain orientation, some portion of the planar arrangement induced by $\{L_j, \alpha_j, \beta_j | j = 1, \ldots, n\}$; we denote this extended collection of segments by S^*_θ.

The edges of CG connect adjacent subcells in FP according to the following two rules, which also give details about the changes in the corresponding faces of FP_θ and the generation of corresponding nodes in CG:

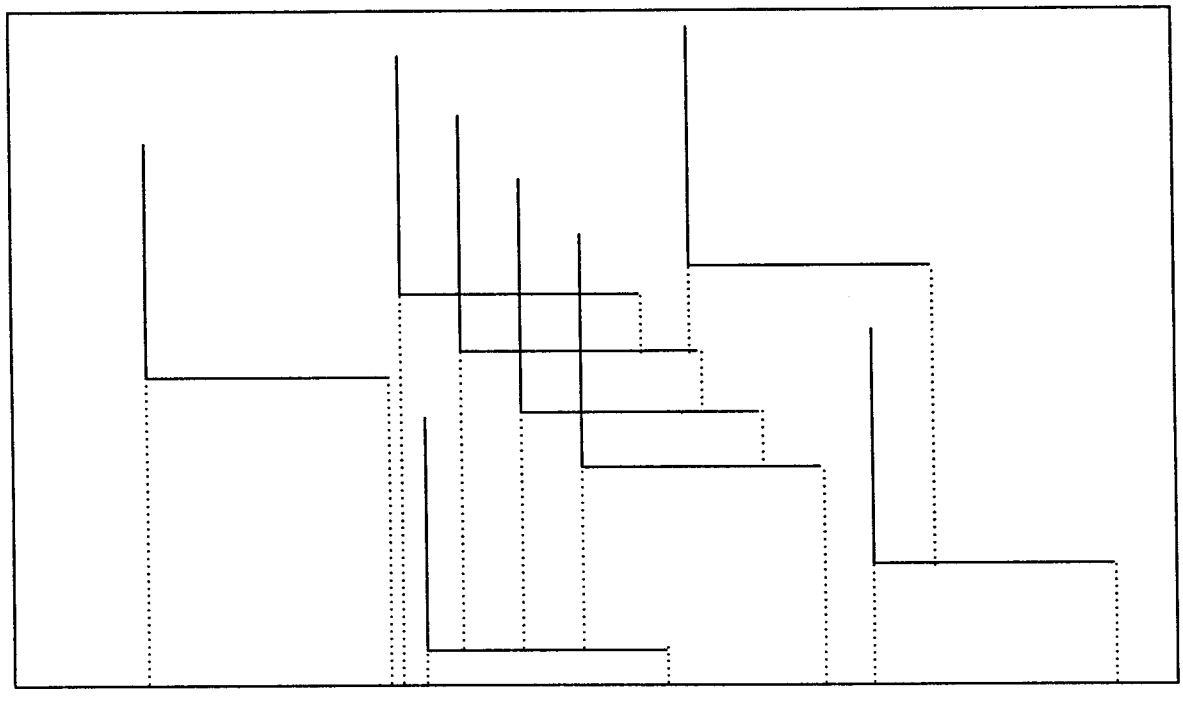
- *Neighboring θ intervals:*

  When a type I critical event occurs, the topological structure of FP_θ changes. For example, as a horizontal bar of some L_k hits a vertical bar l_i of another L_j while moving eastwards (Fig. 6), some face f is split into two adjacent faces f_1 and f_2. If f is *active* at the time of the split (in the particular situation depicted in Fig. 6 this will be the case, as our construction will imply), then its containing subcell c has already a representing node v in the connectivity graph. If f is not *active* (as might happen in certain cases) we add a new node to CG to represent f. Since the two subcells of FP containing f_1 and f_2 are adjacent to c and admit direct crossing between them and c, each of the two respective newly generated nodes of CG will be connected by an edge to v (by definition, both f_1 and f_2 become *active*).

  Similarly, when the motion of L_k with respect to L_j is reversed, two faces of FP_θ are merged into a new face f (which by definition becomes *active*). Again, if any



(a)



(b)

Figure 5: The "horizontal" and "vertical" planar arrangements

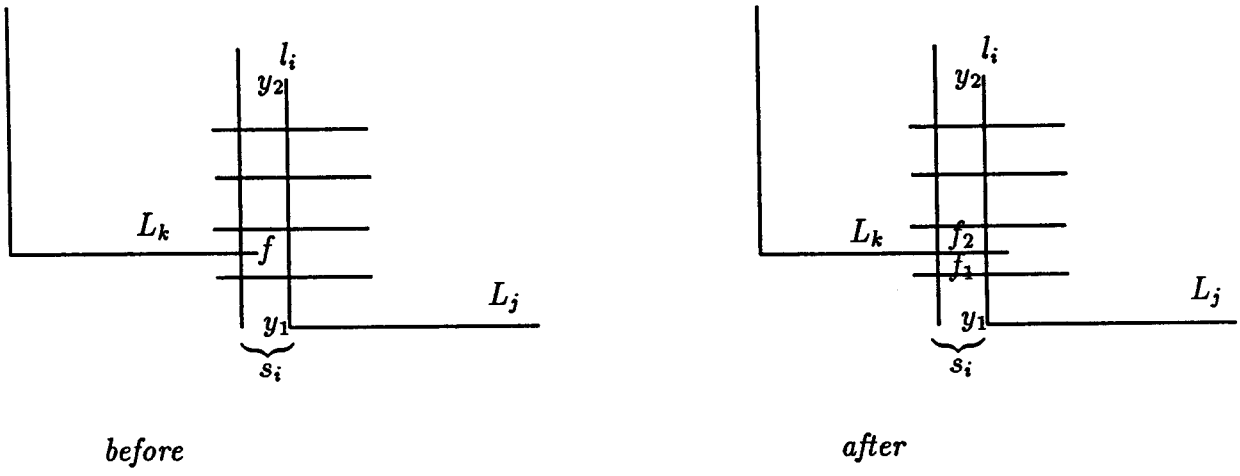


Figure 6: A type I event

of these two faces is *active*, its representing node in CG is connected to the new node representing f in CG. (If any of the merged faces were *inactive* we do not allocate a node in CG for it. The connectivity of our graph is not hindered by this overpass since the subcell of FP corresponding to this *inactive* face would have been a "dead-end". If, on the other hand, either the initial placement or the final placement of L is in that subcell, its face would be activated, as we will later describe.)

Similar generation of new faces of FP_θ, corresponding nodes of CG and connecting edges occurs when the end of the vertical bar of L_j hits the horizontal bar of another L_k, or when an endpoint of the horizontal bar of some L_j comes to lie on the horizontal bar of another L_k, or on one of the horizontal extensions α_k, β_k of another L_k, or, similarly, for an overlapping of vertical bars. We leave the routine details of this generation to the reader.

- *Overlapping θ intervals:*
  The extensions α_j,β_j — the imaginary walls — do not portray real obstacles. Therefore, the two nodes of CG representing subcells that contain two faces that border on such a wall are connected in the connectivity graph by an edge. Such

connections are made either in the initial layer of $\mathcal{CG}$ at θ_0, or at the appropriate type I event.

$\mathcal{CG}$ is the only structure pertaining to the 3-dimensional space FP. The rest of the structures aim to represent the dynamically varying 2-dimensional cross-section FP_θ of FP.

3.2 The horizontal package

The main structure in the horizontal package is R_H and it is supported by the auxiliary structures Q_H and U_H .

As mentioned before, FP_θ refers to the planar arrangement S_θ^* of the L_j's and their horizontal extensions. The horizontal package deals with this arrangement, whereas the vertical package deals with the arrangement induced by $\{L_j, \gamma_j, \delta_j | j = 1, \ldots, n\}$. (Fig. 5 illustrates the two arrangements for the same collection of expanded obstacles. Fig. 5(a) illustrates the "horizontal" arrangement and Fig. 5(b) illustrates the "vertical" arrangement.) Thus there is a slight asymmetry between these packages: while the horizontal package faithfully represents FP_θ, the vertical package assumes a more auxiliary role and represents faces of a different (though related) arrangement. The packages are designed to store the *active/inactive* status of faces. The vertical package serves a single purpose — recording "horizontal squashes". In a horizontal (or vertical) squash, a set of faces is deleted by two segments that partially overlap (and similar faces newly emerge). Since the extremal faces in the squash are treated separately, it follows that all the faces that are involved in the squash are rectangular with all walls real. Such faces have the same representation in both packages. This property will be important in the analysis to follow, as it will allow us to retrieve the *active/inactive* statuts of such a face by querying both packages without any ambiguity concerning the identity of the face.

There is an additional technical issue that needs to be discussed. To exploit observations 1 and 2 made in Section 2, the horizontal package represents the *unlabeled* arrangement S_θ^* of the L_j's, α_j's and β_j's. Thus faces in this arrangement are represented only by their location in this unlabeled arrangement, with no immediate relationship to their actual location in FP_θ. The role of the auxiliary structures Q_H and U_H is to provide a mechanism for locating actual portions of FP_θ in the unlabeled

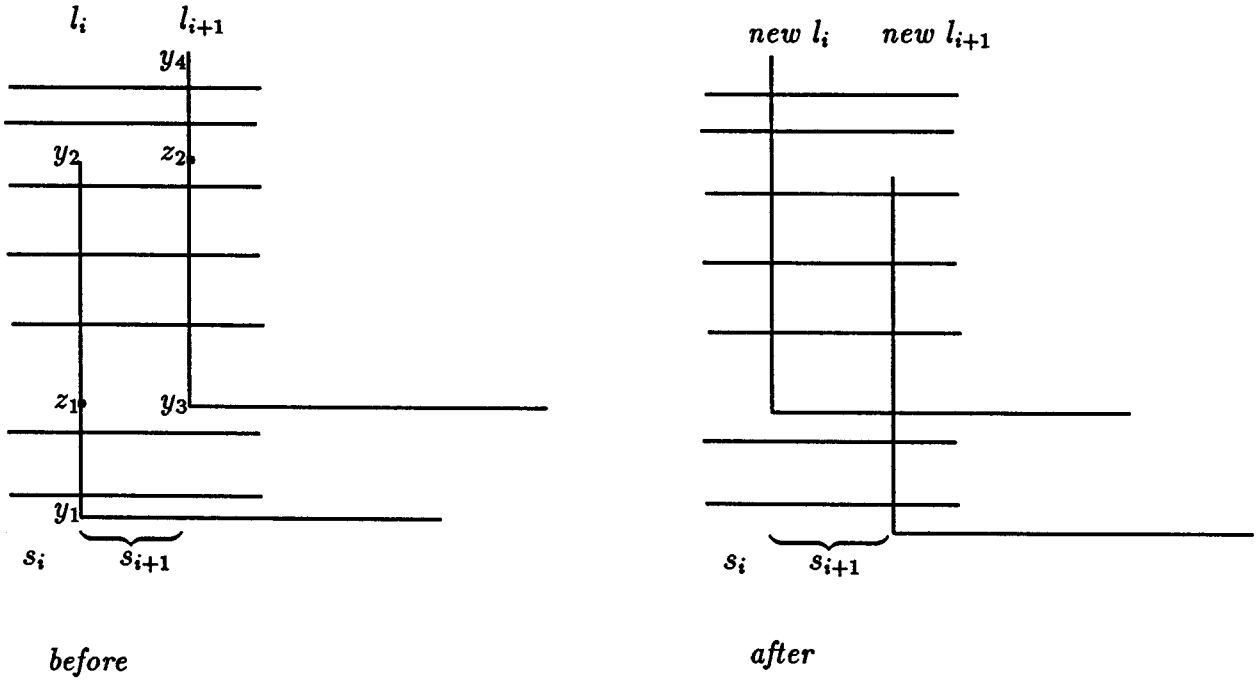


Figure 7: A type II event

arrangement S_θ^* or for identifying features of S_θ^* in the actual cross-section FP_θ. See below for more details.

We divide FP_θ into $n+1$ vertical slabs. The right side of a slab s_i is covered partially by the vertical bar of some L_j (or, for the rightmost slab, by the right side of the surrounding rectangle), which we denote by l_i (the identity of L_j is not stored directly at l_i). l_i is divided into several segments, each of which is an eastern wall of some face. Recall that, by our preceding definition, a face of FP_θ is a maximal connected 2-dimensional component of FP_θ whose eastern wall is a connected portion e of the eastern wall l_i of some slab s_i and contains all the points that are horizontally visible to the west from e. This connected portion e of the eastern wall of s_i is delimited on both ends either by an endpoint of the vertical bar of the corresponding L_j or by the intersection of the vertical bar of L_j with a horizontal bar of another L_k. See Fig. 7 for an illustration.

For each slab s_i we keep information about the faces whose eastern wall is a portion of l_i. (Note that the L_j containing l_i may change during type II events, but that these changes are not stored directly in these faces.) For each such face f we store (in part, implicitly) the following information:

(i) Is f *active* or *inactive*?

(ii) If f is *active* then we also keep a pointer to the node in $\mathcal{CG}$ representing the subcell which contains f.

What happens to R_H when a type II critical event occurs? When two vertical parallel edges overlap, a set of adjacent faces belonging to some slab s_i is squashed (Fig. 7) and a new set of corresponding faces newly emerge. Suppose s_i and s_{i+1} are two adjacent slabs (we number the slabs from west to east), and let l_i and l_{i+1} denote their right delimiting vertical bars. Suppose a type II criticality (partially) overlaps l_i and l_{i+1}. At the instant of the overlap, let

$$l'_i = l_i \setminus l_{i+1} \ , \quad l'_{i+1} = l_{i+1} \setminus l_i$$

and

$$\bar{l}_i = l_i \cap l_{i+1}$$

Then s_i and s_{i+1} should be updated by:

(i) moving the l'_i part from s_i to s_{i+1},

(ii) moving the l'_{i+1} part from s_{i+1} to s_i, and

(iii) marking the $\bar{l}_i$-faces of s_{i+1} as *inactive*.

Recall that the marginal effects of the overlap, that is, the changes in the faces neighboring the ends of $\bar{l}_i$, are treated separately as type I events. We also allow $\bar{l}_i$ to be empty, which is the case when two vertical bars become collinear without actually overlapping. The foregoing discussion applies to this case as well.

Before proceeding with the algorithmic details we describe the various ingredients of the horizontal package.

Q_H is a balanced binary search tree which stores the left-to-right ordering of the vertical bars of the L_j's.

In R_H, for each slab s_i we keep a segment tree T_i. Segment trees are a useful tool for storing and updating sets of intervals when the endpoints of the intervals are known in advance. Usually, some additional information is stored with the intervals (see [Me]). The leaves of the tree T_i correspond to atomic segments which are in our case the eastern walls of the faces of s_i, ordered from south to north. We use a segment tree, since it enables us to mark a contiguous set of elements (in our case, a contiguous set of faces) efficiently.

In order to locate a face in R_H we use U_H, which is a structure for solving the following problem: *Let H be a set of n horizontal bars (of the L_j's), and let s be a query vertical line segment; report the number of segments in H that are intersected by s.* The structure U_H that we use is a *dynamic* version of a well-known two-level combination of a primary segment tree and auxiliary range trees (as in [Ov], for example). More specifically, we project all the segments in H on the x-axis. We construct a segment tree P for the intervals obtained. At each internal node ν of P we store in a binary tree M_ν the segments that are assigned to ν, ordered by y-coordinate. M_ν is a one-dimensional range tree. The structure uses $O(n \log n)$ storage. Let $s = \overline{(x, y_1)(x, y_2)}$ be a query vertical segment. We search in P for x. For each node ν on the search path we count how many segments in M_ν lie between (x, y_1) and (x, y_2); summing up all the counts, we get the number of line segments stabbed by s. The query time is $O(\log^2 n)$, because we perform $O(\log n)$ searches in the auxiliary trees. The structure is dynamic and can be updated in $O(\log^2 n)$ time ([Ov]).

We now continue with describing how each step in handling a type II event as above is executed. We assume that each critical orientation was precomputed with some additional information about the exact location of the corresponding event. So we know the location of l_i and l_{i+1} in our (rotated) coordinate system. With this information we first search the tree Q_H, to find the rank of the relevant slabs in R_H. This is easily done in time $O(\log n)$.

To compute the portions l_i', l_{i+1}', and $\bar{l}_i$ — we consult U_H. For example, let us see how the portion l_i' of Fig. 7 is computed. It is delimited by the lower endpoints of the vertical bars of the two participating L_j's. Using U_H we can easily determine the number of horizontal segments stabbed by the segment l_i'. This number, increased by 1, is also the rank of the face of s_i at which we should split the segment tree T_i

into l'_i and $\bar{l}_i$. In a similar way we compute the portion l'_{i+1}, and as a result of these computations we also have available the $\bar{l}_i$-parts of s_i and s_{i+1}. This takes $O(\log^2 n)$ time.

To update T_i at a type II event at which the eastern sides of s_i and s_{i+1} overlap, we proceed as follows. Consider first the faces that have just become *inactive*, and let $F = \{f_j, f_{j+1}, \ldots, f_{j+m}\}$ be the contiguous sequence of these faces. We record this change in T_i so that a node w of T_i is marked as becoming *inactive* at θ if all the faces corresponding to the leaves of the subtree of w are in F and the faces corresponding to the leaves of the subtree of the father of w are not all contained in F (this corresponds to the usual way of storing a segment in a segment tree). The orientation θ at which this change occurs is also stored at w, serving as a *time stamp*. If for some $\theta_1 > \theta$ we want to mark w as again becoming *inactive*, we just update the "time" of the event, increasing it to θ_1. So each node of T_i stores only $O(1)$ information. (Note that this is a bit different from the standard usage of segment trees in which each node can store a list of segments that "cover" it. Thus our segment trees require only linear storage, in contrast to standard segment trees which may require $O(n \log n)$ storage.) To record the inactivity of the set F in T_i, we begin by finding the ranks of f_j and f_{j+m} in T_i, using U_H as above. Then we search for the corresponding leaves of T_i. During the search we update the relevant nodes of T_i with the "inactivity at θ" stamp, according to the above rule. Only $O(\log n)$ nodes are updated, and the entire operation takes $O(\log n)$ time.

A complementary change occurs when some faces of S^*_θ become *active*. This happens only at events of type I (including those accompanying a type II change), and involves only a constant number of faces, which are simply marked as *active* (with the corresponding θ time-stamp) in the corresponding leaves of the appropriate trees T_i. However, in this step we need to know whether any of the relevant faces is already *active*, so that we can use the same node of CG already representing that face.

To query the *active/inactive* status of a face f we first find to what slab s_i does f belong. Since we know a point on the eastern wall of f (by the geometry of the type I event) we can obtain s_i by consulting Q_H. Next, we look for the rank of f using U_H. Then we search T_i (the tree that describes s_i) for the leaf with this rank. During the search we compute the maximum θ inactivity stamp along nodes in the search path;

call this maximum $\theta_{inactive}$. When we get to the desired leaf, if it is marked *inactive*, then we conclude that the face is currently *inactive*, if the leaf is *active*, we compare the θ at which it has become *active* with $\theta_{inactive}$. The larger of these two θ values determines whether f is *active* or *inactive*. The time needed to perform this query is $O(\log n)$. If f is *active*, we also obtain the corresponding node v of $\mathcal{CG}$ to which it points; we can now use v to link it to the new (*active*) subcells that appear at the current type I event. (However, some additional steps are needed in determining the *active/inactive* status of f — see the discussion below concerning the handling of type I changes.)

In addition to marking the *active/inactive* status of nodes of the trees T_i, we also have to transfer portions of one tree to an adjacent one (those corresponding to l'_i, l'_{i+1} as defined above). For this purpose, instead of using balanced binary segment trees, we use 2-3 segment trees which allow for an efficient splitting and concatenation of such trees in time $O(\log n)$ per operation. (See [KrO]; note that our case is analogous to that of "stabbing counting queries" of [KrO] since in each node we keep only $O(1)$ information, i.e., the θ-inactivity stamp).

As was mentioned before, in order to locate a face in R_H we have used U_H. This structure should be updated at each type II event that overlaps two vertical bars, and at each type I event in which an endpoint of one horizontal bar meets another vertical bar, because then the horizontal order of the endpoints of the horizontal bars in H changes. Each such update costs $O(\log^2 n)$; for details see [Ov]. A type II event that overlaps two horizontal bars, changes the vertical ordering of the horizontal bars; this does not effect the counting, but since we want to take some additional advantage of U_H (to be described below), we update U_H upon such events as well.

Most of the time we employ the *unlabeled* arrangement. There are, however, occasions when we have to resolve the geometric anonymity of the segments in that arrangement. In some of these cases it is sufficient to consult the binary search trees Q_H,Q_V, in other cases we have to allow for queries of the following kind: "Given a point z in the current cross-section of FP, which is the nearest segment (corresponding to some horizontal bar) above z that is vertically visible from z?" Such queries can be answered using U_H. Let $z = (x_1, y_1)$. We search in P for x_1. In each node ν along the search path we search in M_ν for the lowest segment that is higher than y_1, and the lowest of these $O(\log n)$ candidates is the desired segment. This procedure takes

$O(\log^2 n)$ time. The orthogonal type of such a "ray shooting" query, i.e., finding the nearest eastern bar horizontally visible from a point, can be answered in a completely symmetric manner by consulting U_V.

Finally, we update Q_H by interchanging (in $O(\log n)$ time) the two adjacent vertical bars l_i and l_{i+1}.

So far we have described the effect on the horizontal package of type II events at which two vertical bars overlap. Consider next a type II event where two horizontal bars of two expanded obstacles L_j, L_k overlap, or simply become collinear. The (unlabeled) arrangement S_θ^* undergoes several combinatorial changes, all accountable by the type I events accompanying the overlap at its endpoints, plus a "horizontal squash" of some contiguous sequence of faces (see Fig. 8). The horizontal squash is not recorded at all in R_H (but is recorded in the vertical structure R_V) and, in itself, does not effect the combinatorial structure of the unlabeled S_θ^*. However, some of the type I changes do need to be recorded in R_H. For example, in the situation depicted in Fig. 8, we need to insert a new leaf into the segment tree T_i of the slab corresponding to L_k. This leaf stands for the new face f whose eastern wall is the lowest segment of L_k. f is marked *active*. In the symmetric situation, an extreme leaf of some T_i may have to be deleted. The effects of these type I changes on CG are described below.

It is interesting to note that the right endpoints of L_j,L_k do not cause any updating of R_H. We have also observed above that the structure Q_H is not effected by such a type II event.

How is R_H effected by a type I event? Consider for example a horizontal bar of some L_j hitting the vertical bar l_i of the slab s_i while moving eastwards (Fig. 5). As in type II events, using the actual geometric data accompanying this critical event we can identify the corresponding slab s_i of R_H and in this slab we can find (the rank of) f_j — the face that is going to split. We then split f_j into two new subfaces, update s_i and T_i accordingly, add two new nodes to the connectivity graph to represent these two new (and *active*) faces, and point to the new nodes from the two corresponding newly generated leaves of T_i. Since f_j was not rectangular prior to the change, it was necessarily *active*, so we connect the two new nodes of CG to the node representing f_j. By the same token, we make the new face into which the tip of L_j penetrates *active*.

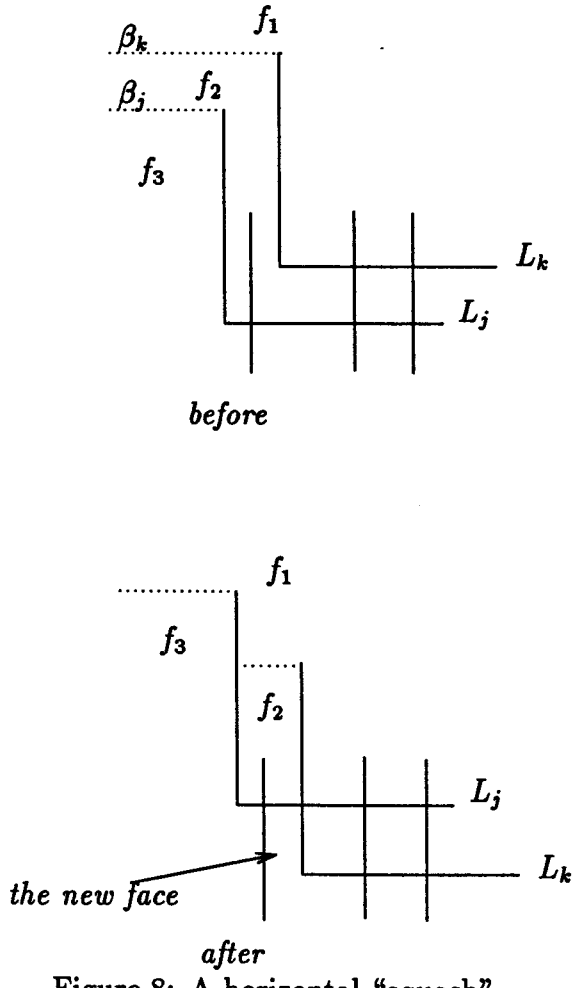


Figure 8: A horizontal "squash"

Note that to determine the *active/inactive* status of the face f_j it is not enough to consult R_H only, because f_j could have already become *inactive* in a horizontal squash, and such an event is marked only in R_V. A face is really *active* if and only if it is determined to be *active* in both R_H and R_V. A technical difficulty arises here, because R_V does not represent exactly the same faces as does R_H, since it stores the partition created by the L_j's and their *vertical extensions*. However, as already noted, a face that had been involved in a horizontal squash and was not re-activated later, must be rectangular, with its four sides *real*, and, as such, is stored identically in both packages. We thus proceed as follows. Assume f_j has been determined to be *active* in R_H. Find the horizontal bar passing through the top endpoint of the eastern wall of f_j (using U_H and the "ray-shooting" technique previously described). Regard that portion of that bar immediately to the left of this endpoint as the northern wall of some face f' in R_V, and query R_V to determine the *active/inactive* status of f'. If f' is *inactive* then so is f_j. Otherwise f_j is *active*.

When a horizontal bar leaves a vertical bar while moving westwards (e.g., exchange *before* and *after* in Fig. 5), the steps are quite similar to those described above, and consist of checking the *active/inactive* status of the relevant faces in R_H and in R_V, adding nodes to the connectivity graph, adding the relevant edges to CG, etc.

Concerning the additional type I-like updating that should be done at the extreme faces of a type II overlap, consider, for example, the situation shown in Fig. 7. The face in s_i whose eastern wall contains z_1 is split into two *active* faces, with appropriate connections made in CG; similar changes apply to the face in s_{i+1} whose eastern wall contains z_2.

Finally, consider the effects of the type I changes accompanying a type II event in which the horizontal bars of L_j and L_k overlap, for some j and k. We have noted above how these changes effect R_H. Their influence on CG has to reflect the possible changes in the combinatorial structure of the faces bordering the imaginary horizontal extensions α_j,α_k,β_j,β_k. For example, in the situation shown in Fig. 8, before the overlap we have a face f_2 bounded by β_j from above and by β_k from below and connected, via these extensions, to the face f_1 above it and to the face f_3 below. After the overlap, f_3 becomes directly connected to f_1. To keep track of these changes, we create new nodes in CG to represent f_1,f_2 and f_3 after the change, connect each node to the corresponding old node, and add edges connecting the new

f_1 to the new f_2 and to the new f_3. Similar action is taken to handle the possible overlap between the extensions α_j,α_k. (Note however that these changes take place only if the extensions β_j,β_k (or α_j,α_k) actually overlap. If β_k ends on a vertical bar of another L_t that lies to the right of the vertical bar of L_j, no changes are required.)

3.3 The vertical package

We mentioned before that the horizontal package and the vertical package each describes a different planar arrangement. They coincide, though, in the description of (the rectangular) faces which become *inactive*, are *inactive* or turn from *inactive* to *active*, as follows from the discussion in the previous subsection. The vertical package is handled in a manner completely symmetric to that of the horizontal package with regard to the respective unlabeled planar arrangement (namely, the L_i's and their vertical extensions); we thus omit the details of the manipulation of this package.

4 Algorithmic Details and Complexity Analysis

In this section we complete the details of the algorithm, prove its correctness, and analyze its time and space requirements.

4.1 The Algorithm

Construction of FP_{θ_0}. First, the critical orientations are computed and sorted. Then a non-critical θ_0 is chosen and the Minkowski differences L_i for θ_0 are computed. We sort the vertical bars of the L_i's according to their x-coordinate and store this ordering in Q_H. Now we start sweeping a vertical line across the plane from left to right while maintaining a sorted list F of all the horizontal line segments intersecting the line being swept. Each time we sweep across an internal vertex q_j of some L_j, we construct a new segment tree T_i that describes the slab s_i. To build T_i we first locate the position of p_j (the upper external vertex of L_j) and of q_j in F and then allocate a segment tree for the intervals in F from p_j to q_j. We mark all the faces *active*. Each time we sweep across a right external vertex r_j of some L_j, we remove the segment $\overline{q_j r_j}$ from F. The last stop of the sweep is the eastern wall of the surrounding rectangle,

where we have exactly one face which we will keep in T_{n+1}. The set of all the segment trees $\{T_i | i = 1, 2, \ldots, n+1\}$ constitutes R_H. Similarly, we construct Q_V and R_V.

During the line sweep we also lay an initial layer of CG. Every time we create a segment tree T_i, we add a node to CG for every leaf of T_i. These nodes represent all the faces whose eastern wall is in l_i (l_i — the vertical bar on the right side of s_i). Each pair of nodes whose corresponding faces share an imaginary horizontal wall is connected by an edge. Note that at least one node of such a pair is an extreme face of some T_i. To obtain the desired connections we maintain a list W of all corners of L_j's that have already been swept through and that are still "visible" from the sweep line. Whenever we sweep through a new vertical bar l_i we remove from W all endpoints horizontally visible from l_i, and connect the extreme nodes of the corresponding segment trees to the appropriate new faces of s_i. The endpoints of l_i are then added to W.

The overall time of this intial phase of the algorithm requires $O(n^2)$ time, since the sweep itself is easily seen to require $O(n \log n)$ time and for each slab we build a segment tree when the enpoints of the segments are already sorted by the sweep structure.

The second phase of the algorithm, updating the structures as the orientation changes, is discussed in detail in Section 3. A missing link there, though, is the wrap-around of CG. Reaching $\theta = \theta_0 + 2\pi$ our task is to identify the nodes of the *active* faces with their matching peers in the first layer of CG. But, at $\theta_0 + 2\pi$ there is no longer a way to know to which face in $FP_{\theta_0+2\pi}$ does a node of the first layer of CG belong. A simple solution to the problem is to keep a duplicate copy of the horizontal package at θ_0. Getting to $\theta_0 + 2\pi$ we scan the current updated R_H and for every *active* face of $FP_{\theta_0+2\pi}$ we search for its matching (identical) face in the original version of R_H, and identify the two corresponding nodes in CG.

Some final details of the processing involve the reachability query itself. Let $Z_s = (X_s, \theta_s)$ be the initial placement of L, and $Z_d = (X_d, \theta_d)$ be the final placement. Let c_s and c_d be the subcells of FP containing Z_s and Z_d respectively. To ensure that these subcells will be represented in CG, we add two artificial critical events, θ_s and θ_d, during the θ-sweep. The purpose of these events is to identify or otherwise introduce the nodes v_s,v_d of CG corresponding to c_s,c_d respectively. When we reach θ_s we look for the face f_s containing X_s, using first U_V to identify the nearest vertical bar to the east of X_s that is horizontally visible from X_s, and then U_H to find (the

rank of) f_s in the corresponding slab. If f_s is *active* we obtain from it a pointer to v_s, otherwise we make it *active*, update R_H,R_V accordingly, create a new node v_s in CG to represent f_s, and keep a pointer to this newly generated node. Similar steps are taken when we reach θ_d. After the completion of the θ-sweep, we search for a path in CG between v_s and v_d. If a path is found, the algorithm outputs YES, otherwise it outputs NO.

The following proposition justifies the reduction of our motion planning problem to the purely combinatorial path searching through CG.

Proposition 4.1 *If both Z_s and Z_d are free positions of L, then there is an obstacle-avoiding motion between Z_s and Z_d if and only if v_s and v_d belong to the same connected component of CG.*

Proof. For the "if" part, let ϕ be a path between v_s and v_d in CG. First, it is easily verified that any single node v of CG represents a connected portion c of FP, that is, any two placements of L within c can be reached from one another along a collision-free path that remains in c (see also Remark 4.1 below). Next, each edge e along ϕ represents one of the following "crossings":

(i) e connects two nodes representing subcells whose cross-sections at some θ are adjacent along some imaginary wall; in this case there is a direct translational crossing of L at this θ between the subcells (although this is not required in this part of the proof, we note that our construction ensures that if this crossing is possible at one θ, it is possible at all θ's at which both subcells exist); or

(ii) e connects nodes representing subcells that were both influenced by the same type I event; in this case it is easily verified that there is some rotational crossing between the two relevant subcells. (There is one exception: If these subcells represent two faces that were split from one *inactive* subcell, our procedure has created a shortcut connection between them, which does not correspond to direct crossing between the subcells. Nevertheless, it is still possible to cross from one of them to the other by passing through the *inactive* subcell adjacent to both.)

These observations clearly imply that the given path in CG can be transformed into a collision-free path within FP.

As for the "only if" part, define a retraction-like mapping $\Psi_H : FP \to BFP$ as follows. For each $Z = (X, \theta) \in FP$ move the object L by translating it in the direction of its 'horizontal' bar $\vec{rq}$ (so that r moves "towards" q) until the vertical bar hits an obstacle (or L reaches the enclosing rectangle). The resulting position is $\Psi_H(Z)$. (Even if the horizontal bar touches an obstacle at Z we still allow this sliding motion to be performed.) It is easily checked that Ψ_H is continuous in the interior of FP except at points that lie on the imaginary extensions α_j,β_j for some L_j.

Now suppose there is a collision-free path $F : [0,1] \to FP$ connecting $Z_s = F(0)$ to $Z_d = F(1)$. The composition $G = \Psi_H \circ F$ is a piecewise-continuous path which, using standard topological arguments akin to those used in [SS], we can assume to consist of only a finite number of connected pieces. Note that the endpoints of each piece of G lie in *active* faces of the corresponding cross-sections of FP, because each endpoint is either Z_s, or Z_d, or lies on an imaginary horizontal extension which by definition, always bounds two *active* faces. Moreover, by construction of CG, for any two consecutive portions of G, the nodes of CG in which the first subpath ends and in which the second subpath begins are connected by an edge. It therefore suffices to prove that each subpath G' of G induces a path in CG connecting the two nodes that contain the endpoints of G'.

Note that, in the cross-sectional representation of FP that we use, each path G' is represented by a point w varying continuously along an eastern wall of some face(s) of the arrangement S^*_θ (that also varies with θ). Since w begins its motion in an *active* face, it suffices to verify that it always remains in an *active* face, and that whenever this face changes, a corresponding type I event which involves this change occurs (and induces a connecting edge between the corresponding nodes of CG). To show this, we first break G' into a number of pieces, such that on each of them θ varies monotonically and does not cross θ_0; without loss of generality we can assume that there are only a finite number of such pieces. If such a piece starts at an *active* face and proceeds in the direction of increasing θ (including the case of crossing $\theta_0 + 2\pi$ back to θ_0), then our construction is easily seen to imply the property asserted above. (A point to note here is that if w crosses between faces through an imaginary extension, then this crossing must have been possible at a type I event that involved both faces, and therefore created the corresponding edge in CG.) If θ decreases along such a subpath, we have to be more careful as our construction does not guarantee that the

active status is propagated backwards in θ. However, a close inspection of our construction shows that if we move backwards in θ from an *active* face f to an *inactive* face f' then f' must be a "dead-end" face that will eventually (i.e., if we continue to decrease θ) be squashed at some type II event. Since G' ends up in an *active* face, it cannot stay in f' and must exit by reversing its θ direction and cross back from f' to f, or perhaps from f' to another face f'' separated from f by an imaginary horizontal extension. In the first case we simply ignore the excursion of G' into f'; in the second case our construction induces a shortcut connection between the nodes of $\mathcal{CG}$ containing f and f'' respectively. Finally, if a subpath of G' starts at θ_0 (it does so in an *active* face by construction), and moves backwards from θ_0 (θ decreasing), then it might enter an *inactive* face in $FP_{\theta_0+2\pi}$, because at this cross-section not all the faces are necessarily *active*. But the above arguments that the path will then have to move back into θ_0 apply to this case as well. This completes the proof of the "only if" part of the proposition. □

Remark 4.1 . It is instructive to describe a canonical path in FP that corresponds to a given path ϕ in $\mathcal{CG}$. The desired path Π is a concatenation of subpaths, each describing a simple motion of L, as follows. Suppose ϕ has reached a node v of $\mathcal{CG}$ and let v' be the next node along ϕ. Let c,c' be the corresponding subcells of FP. Inductively, suppose Π has already reached some placement $Z \in c$. To continue Π to reach a placement $Z' \in c'$ we proceed as follows. By construction, c is a subcell in which θ varies between two type I critical orientations $\theta_1 < \theta_2$ and for each $\theta_1 < \theta < \theta_2$, $c \cap FP_\theta$ corresponds to the same face f in the unlabeled arrangement S^*_θ. If f contains (on its boundary) a corner w of some L_j, we move (translationally) to w, and stay at w as θ varies. Otherwise f must be rectangular, with all four walls real. In this case we move to, say the northeast corner of f and stay there as θ varies. Note that in the latter case the combinatorial complexity of the motion within c can be $\Omega(n)$, because the northeast corner of f can change whenever its eastern wall or northern wall changes due to a type II event.

In the physical space, the first type of motion becomes a rotation of L while one of its three corners touch an obstacle. The second type of

motion is a "gliding" motion of L, in which it rotates while each of its bars touches an obstacle, such that whenever any of the bars encounters a new obstacle o_i, the gliding continues with that bar touching o_i. Note that the middle corner q of L traverses a circular arc during each portion of the gliding.

Finally, the crossing from c to c' is easy to accomplish. If this crossing is through an imaginary wall at some θ, we move within c, as specified above, to θ (as noted above, any θ at which both c and c' exist will do) and then translate to c' through the wall. If the crossing is through a type I change, we reach the corresponding extreme critical orientation θ as above, then cross locally according to the nature of the type I change (rotating further into the new cell, translating across an imaginary wall, rotating and translating back and forth to realize an indirect connection, etc.).

4.2 Complexity analysis

4.2.1 Analysis of the 2D data structures

In this subsection we summarize the computational cost of maintaining and manipulating the 2-dimensional data structures. We analyze below the cost of the horizontal package but the analysis of the vertical package is essentially identical.

Q_H, the structure retaining the horizontal ordering of the vertical bars of the L_i's, is a balanced binary tree with n elements. Its initial construction takes $O(n \log n)$ time. Upon each type II criticality that overlaps vertical bars, we interchange these two adjacent elements in $O(\log n)$ time. Upon each type I criticality bringing an endpoint of a horizontal bar to cross a vertical bar, we query Q_H in $O(\log n)$ time. Therefore the usage of Q_H costs $O(n^2 \log n)$ time. Being a balanced binary tree with n elements it demands $O(n)$ space. To summarize:

Lemma 4.2 *Q_H requires $O(n^2 \log n)$ time and $O(n)$ space, and so does Q_V.*

As to R_H, initially we build n 2-3 segment trees, as part of the initial sweep; this construction requires $O(n^2)$ time (as noted before). Afterwards, each operation on

any of these trees — delete, insert, concatenate, split or query — requires $O(\log n)$ time. The number of operations on R_H that are required at each criticality is bounded by a constant. Thus, the usage of R_H costs $O(n^2 \log n)$ time. See [KrO] for more details. Each tree requires $O(n)$ storage, summing up to $O(n^2)$ storage for R_H. Thus, we have:

Lemma 4.3 *R_H and R_V each requires $O(n^2 \log n)$ time and $O(n^2)$ space.*

U_H is a two-level combination of a primary segment tree and auxiliary range trees. It is built in $O(n \log n)$ time. Each update requires $O(\log^2 n)$ time and a query takes $O(\log^2 n)$ time. There is a constant number of updates and queries per criticality, so the operations on U_H take $O(n^2 \log^2 n)$ time in total. It uses $O(n \log n)$ space. It follows that:

Lemma 4.4 *U_H requires $O(n^2 \log^2 n)$ time and $O(n \log n)$ space, as does U_V.*

For details on the dynamic segment-tree – range-trees combination see [Ov].

4.2.2 Analysis of the overall complexity

The initial phase of the algorithm (at θ_0) includes a vertical sweep and a horizontal sweep, both fairly standard and taking only $O(n \log n)$ time, as is easily seen.

How complex is CG?

Lemma 4.5 *CG has $O(n^2)$ nodes and edges.*

Proof. At the initial phase, FP_{θ_0} has at most $O(n^2)$ *active* faces. So the first layer of CG has $O(n^2)$ nodes. During the θ-sweep we have $O(n^2)$ stops. At each stop we add a constant number of nodes to CG and with each new node we add at most a constant number of edges to connect it to some previously existing, or newly generated, nodes. □

After the completion of the θ-sweep, we identify the nodes of CG corresponding to the *active* faces of $FP_{\theta_0+2\pi}$ with the matching nodes of faces of FP_{θ_0}.

Lemma 4.6 *The wrap-around of $\mathcal{CG}$ requires $O(n^2)$ time.*

Proof. To identify nodes of the two layers, we follow the R_H structure at $\theta_0 + 2\pi$ and the reserved duplicate of the R_H structure at θ_0. For each *active* face of the newer R_H, we identify its node in $\mathcal{CG}$ with that of the first layer. Since we pass sequentially (e.g., in inorder on every tree) through the $O(n^2)$ faces, the traversal requires $O(n^2)$ time. □

Finally, to find a path from v_s to v_d (if one exists), we search through $\mathcal{CG}$.

Lemma 4.7 *The search through $\mathcal{CG}$ for a path from v_s to v_d takes $O(n^2)$ time.*

Proof. The search can be carried out using breadth-first-search which is linear in the number of edges in the graph. Since $\mathcal{CG}$ has $O(n^2)$ edges, the bound follows. □

We are now ready for the main theorem.

Theorem 4.1 *The algorithm answers the reachability query correctly, using $O(n^2 \log^2 n)$ time and $O(n^2)$ space.*

Proof. The correctness of the algorithm follows from the analysis in the proof of Proposition 4.1.

As for the time required by the algorithm. We start by computing and sorting the critical orientations. This can easily be done in $O(n^2 \log n)$ time. The vertical and horizontal sweeps take $O(n \log n)$ time. The building and usage of the 2D data structures take $O(n^2 \log^2 n)$ time (follows from Lemmas 4.2, 4.3 and 4.4). The wrap-around of $\mathcal{CG}$ requires $O(n^2)$ time (Lemma 4.6) and the search through $\mathcal{CG}$ requires $O(n^2)$ time (Lemma 4.7). We see, then, that the usage of the 2D data structures (specifically, the usage of U_H and U_V) dominates the time complexity of the algorithm which is $O(n^2 \log^2 n)$.

As for the storage requirements, $\mathcal{CG}$ consists of $O(n^2)$ elements (Lemma 4.5), and no other structure or procedure requires more space. □

5 Conclusion

We have presented here an $O(n^2 \log^2 n)$ time algorithm for the solution of the reachability problem for an L-shaped object moving amidst point obstacles in the plane. In this section we summarize the new ideas of our approach, assess the algorithm efficiency and point out to possible extensions of this work.

The main innovation of our approach is the condensation of a potentially $\Omega(n^3)$-size configuration space FP into quadratic-storage structures using near-quadratic time. This is achieved by building a skeletal connectivity graph which, in contrast with previously suggested connectivity graphs, (almost) does not contain dull cells and suppresses the explicit representation of most of the changes that occur in interesting cells. The compaction of the connectivity graph is enabled by some auxiliary evanescent data structures which, at every instant of the θ-sweep, store necessary information about the momentary FP cross-section in an implicit compact manner.

As already mentioned, the complexity of the entire FP in our case can be $\Theta(n^3)$ in the worst case. However, in most cases one does not need to calculate the entire FP, but only its connected component C containing the initial given placement Z_s of L. Indeed, as long as L moves in a collision-free manner from Z_s, it will have to remain in C. We would like to pre-calculate only the component C rather than the entire FP, and to show that this will always achieve better performance than the naive cubic worst-case bound for the entire FP. This goal was achieved, to a satisfactory extent, for the general motion planning problem with two degrees of freedom [GSS]. Unfortunately, in problems with three degrees of freedom (such as ours) this goal appear to be much harder. In the special case in which the surface patches bounding FP are n triangles in three-space, it was shown in [AS] that the complexity of a single cell in the complement of their union is at most $O(n^{7/3}\alpha(n)^{2/3}\log^{4/3} n)$. However, in the case of the L-shaped body, the surface patches bounding FP are more complicated. The 3D arrangement induced by these patches seems, nevertheless, analyzable with tools similar to those used in [AS]. Obtaining subcubic bounds on the complexity of a single cell of FP in our case is therefore one of the directions for further research that are suggested by this paper.

Another direction concerns the observation that the reachability problem only decides whether there exists a continuous motion between source and destination.

Naturally, we would like to produce such a motion if it exists, and do so in subcubic time. If we show that the complexity of a single component of FP is subcubic, then we believe that we can elaborate our connectivity graph to record the changing walls, during type II events, in the subcells it represents, while continuing to ignore dull cells (and other components of CG) so that the resulting find-path algorithm will require subcubic resources. Moreover, Remark 4.1 shows that the portions of the required path that lie in faces that contain corners of some L_j's are easy to produce — we simply "take a ride" on the corresponding corner. The difficulty lies in producing these portions of the path that traverse "tube-like" subcells having a real rectangular cross-section (which however can undergo $\Omega(n)$ changes due to type II criticalities). It is interesting to note that from a pragmatic point of view this issue may not be problematic. Assuming our object L to be equipped with tactile sensors all around it, traversing a tube-like subcell is easy to accomplish by executing the corresponding gliding motion (as in Remark 4.1) and using tactile feedback to tell when one of the two obstacles touched by L during the gliding has to be replaced by another. Thus, in this pragmatic setting, our reachability algorithm can easily be adapted to produce the desired path, at no additional overhead.

We believe that our algorithm is extensible to polygonal obstacles (not only point obstacles) and to an arbitrary rectilinear non-convex moving object, without severely increasing the time and space complexity. Moreover, there seems to be a close connection between the problem of the non-convex body moving among polygonal obstacles and the problem of moving certain kinds of planar robot arms with three degrees of freedom (such as a standard 3-link anchored planar arm or the arm studied in [AO]) in the same setting. We are presently investigating these extensions.

Finally, there is the issue of improving the performance of our reachability algorithm. Two problems suggest themselves. First, can we improve the time required by the algorithm to $O(n^2 \log n)$? As noted, the only step which requires $O(n^2 \log^2 n)$ time is the handling of the structures U_H,U_V. Second, can one show a lower bound $\Omega(n^2)$ on the number of nodes of CG that a path must traverse between some pair of placements? For our reachability problem such a bound would not necessarily preclude the possibility of a faster decision procedure, but it would be a strong indication that quadratic complexity is probably a correct worst-case bound.

References

[ABF] F. Avnaim, J. D. Boissonnat and B. Faverjon, A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles, Preprint, 1988.

[AS] B. Aronov and M. Sharir, Triangles in space, or building (and analyzing) castles in the air, *Proc. 4th ACM Symposium on Computational Geometry*, 1988, pp. 381–391.

[AO] B. Aronov and C. Ó'Dúnlaing, Analysis of the motion-planning problem for a simple two-link planar arm, Technical Report 314, Courant Institute of Mathematical Sciences 1987.

[Ca] J. Canny, The complexity of robot motion planning, Ph.D. dissertation, Computer Science Department, M.I.T., May 1987.

[GSS] L. Guibas, M. Sharir and S. Sifrony, On the general motion planning problem with two degrees of freedom, *Proc. 4th ACM Symposium on Computational Geometry*, 1988, pp. 289–298.

[KeO] Y. Ke and J. O'Rourke, Lower bounds on moving a ladder in two and three dimensions, *Discrete and Computational Geometry* **3** (1987), pp. 197–218.

[KrO] M. J. Kreveld and M. H. Overmars, Concatenable Segment Trees, Technical Report RUU-CS-88-36, Computer Science Department, University of Utrecht, 1988.

[KS1] K. Kedem and M. Sharir, An efficient algorithm for planning collision-free translational motion of a convex polygonal object in 2-dimensional space amidst polygonal obstacles, *Proc. 1st ACM Symposium on Computational Geometry*, 1985, pp. 75–80.

[KS2] K. Kedem and M. Sharir, An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space, to appear in *Discrete and Computational Geometry.*

[LS] D. Leven and M. Sharir, An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers, *Journal of Algorithms* **8** (1987), pp. 192–215.

[LW] T. Lozano-Perez and M. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Comm. ACM* **22** (1979), pp. 560–570.

[Me] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.

[OY] C. Ó'Dúnlaing and C. K. Yap, A retraction method for planning the motion of a disc, *Journal of Algorithms* **6** (1985), pp. 104–111.

[Ov] M. H. Overmars, Geometric data structures for computer graphics: an overview, *Theoretical Foundations of Computer Graphics and CAD*, Edited by R. A. Earnshaw, Springer-Verlag, Berlin Heidelberg, 1988, pp. 167–184.

[SiS] S. Sifrony and M. Sharir, An efficient motion planning algorithm for a rod moving in two-dimensional polygonal space, *Algorithmica* **2** (1987), pp. 367–402.

[SS] J. T. Schwartz and M. Sharir, On the "Piano Movers" Problem I. The case of a two dimensional rigid polygonal body moving amidst polygonal boundaries, *Communications on Pure and Applied Mathematics* **36** (1983), pp. 345–398.