Dog bites postman: point location in the moving Voronoi diagram and related problems

Olivier Devillers, Mordecai Golin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

Dog Bites Postman: Point Location in the Moving Voronoi Diagram and Related Problems

Olivier Devillers et

Mordecai Golin

N° 2263

Avril 1994

PROGRAMME 4

Robotique,

image

et vision

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Dog Bites Postman: Point Location in the Moving Voronoi Diagram and Related Problems

Olivier Devillers* et
Mordecai Golin**

Programme 4 — Robotique, image et vision
Projet Prisme

Rapport de recherche n°2263 — Avril 1994 — 29 pages

Abstract: In this paper, we discuss two variations of the two-dimensional post-office problem that arise when the post-offices are n postmen moving with constant velocities. The first variation addresses the question: given a point q_0 and time t_0 who is the nearest postman to q_0 at time t_0? We present a randomized incremental data structure that answers the query in expected $O(\log^2 n)$ time. The second variation views a query point as a dog searching for a postman to bite and finds the postman that a dog running with speed v_d could reach first. We show that if the dog is quicker than all of the postmen then the data structure developed for the first problem permits us to solve this one in $O(\log^2 n)$ time as well.

The proposed structure is semi-dynamic, that is the set of postmen can be modified by inserting new postmen. A fully dynamic structure supporting also deletions can be obtained, but in that case the query time becomes $O(\log^3 n)$.

Key-words: Computational geometry, moving points, Voronoi, Delaunay, randomized algorithms, dynamic algorithms, localisation.

(Résumé : tsvp)

Research of O. Devillers was supported in part by ESPRIT Basic Research Actions 7141 (ALCOMII) and 6546 (PROMotion). Work of M. Golin was supported in part by grant HK RGC CRG grant HKUST 181/93E.

*INRIA, B.P.93, 06902 Sophia-Antipolis cedex (France), Phone: +33 93 65 77 63, E-mail: Olivier.Devillers@sophia.inria.fr.

**Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, Phone: +852 358-6993, E-mail: golin@cs.ust.hk.

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Les chiens mordent les postiers : localisation dans le diagramme de Voronoï mobile et problèmes associés

Résumé : Nous nous intérèssons à deux variantes du problème classique du bureau de poste dans lesquelles les bureaux de poste sont remplacés par n postiers se déplaçant à vitesse constante. La première variante concerne la question : étant donné un point q_0 et une date t_0 quel est le postier le plus proche de q_0 au temps t_0 ? Nous présentons une structure de donnée incrémentale randomisée permettant de répondre à ce type de requête en temps $O(\log^2 n)$. La deuxième variante considère une requête comme un chien cherchant à mordre un postier le plus rapidement possible en se déplaçant à vitesse v_d. Nous montrons que si le chien va plus vite que tous les postiers, la structure développée pour le premier problème permet de répondre à ce type de requête dans le même temps $O(\log^2 n)$.

La structure proposée est semi-dynamique, c'est-à-dire que les postiers peuvent être ajoutés un à un sans que la structure soit entièrement recalculée à chaque étape. Cette structure peut être rendue complètement dynamique (l'ensemble des postiers est modifié avec des insertions et des suppressions) et le temps de réponse aux requêtes est alors de $O(\log^3 n)$.

Mots-clé : Géométrie algorithmique, points mobiles, Voronoï, Delaunay, algorithmes randomisés, algorithmes dynamiques, localisation.

1 Introduction

The *post-office problem* is to pre-process the locations of a city's post-offices (sites in the plane) so that a customer (query point) can quickly find the closest post-office. This problem is usually approached by constructing the Voronoi diagram of the post-offices. The Voronoi diagram of n sites[1] divides the plane into n convex regions, each region consisting of the set of all points closest to a particular site. The post-office problem is then solved by performing a planar-point location in the Voronoi-diagram. The basic problem, along with countless variations, has been extensively studied [Aur91]. In this paper we propose and study yet two more variations, ones in which the sites are allowed to move.

Suppose, then, that the town fathers of our mythical city have decided to cut their fixed expenses, close down the post-offices, and replace them by postmen following appointed rounds. A customer wanting to mail a package at 10:15 AM on Monday morning needs to be able to locate the postman closest to his office at that time. How can the postmen be preprocessed in a way that permits him to efficiently find such a postman?

In this paper we study the problem of point location in the Voronoi diagram of n moving, two-dimensional sites. We asssume that each site (postman) moves with constant velocity:

$$p_i(t) = q_i + v_i t, \qquad i = 1, \ldots, n \tag{1}$$

where $p_i(t)$, q_i, $v_i \in I\!R^2$; $p_i(t)$ is the location of the postman at time t, q_i its location at time 0, and v_i its velocity. As in the static case we would like to preprocess the postmen so as to easily answer the question "given a query point q_0, who is the closest postman to it at specified time t_0?"

In the static post-office problem the meaning of "closest" was quite clear. Because the world did not change, the *nearest* post-office to a customer was also the post-office that the customer could *reach quickest.* In the postman problem we must distinguish between these two different types of closeness. Because the postmen are moving, the nearest postman at time t_0 might not be the postman that can be reached quickest. See Figure 1. With this in mind we define the following two different types of queries: ($|p - q|$ is the Euclidean distance between p and q)
(1) Moving-Voronoi queries: Given a customer at location $q_0 \in I\!R^2$ find the nearest postman at time $t_0 \in I\!R$. That is, given q_0, t_0, return i such that

$$|p_i(t_0) - q_0| \leq |p_j(t_0) - q_0|, \quad j = 1. \ldots, n.$$

[1] To avoid confusion we use "site" to denote a postman or postoffice and "point" to denote an arbitrary query point.

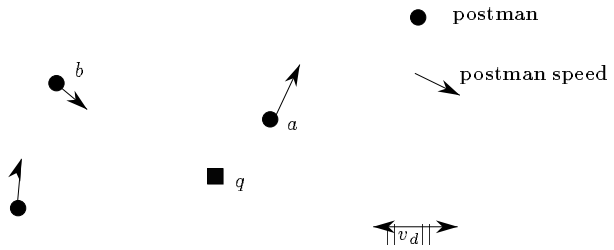

Figure 1: The nearest postman to q is a but the postman that q can reach the quickest is b.

(2) Dog-Bites-Postman queries: The query inputs are $q_0 \in \mathbb{R}^2$, $t_0 \in \mathbb{R}$, and $v_d > 0$. They specify a dog located at q_0 at time t_0 capable of running at maximum speed v_d. The dog is mean; it wants to catch and bite a postman. It is also hungry and impatient; it wants to bite a postman as soon as possible. The problem here is to find the postman the dog can reach quickest. Set

$$t_j = \min\{t \geq t_0 \,:\, (t_j - t_0)v_d = |p_j(t) - q_0|\}, \quad j = 1, \ldots, n$$

to be the first time that the dog can catch postman j. Then the query returns i such that $t_i \leq t_j$, $j = 1, \ldots, n$. It is possible that the dog will not be able to catch any postmen. If this happens the query returns that fact.

A moving-Voronoi query attempts to find the nearest postman to a query point at a given time. This problem is equivalent to freezing time at $t = t_0$ and then locating point q_0 in the Voronoi diagram of $p_1(t_0), \ldots, p_n(t_0)$. An essential characteristic of this problem is that once time is frozen the later movement of the sites is of no further importance.

A dog-bites-postman query – henceforth abbreviated as a dog query – attempts to find the postman that can be reached quickest when starting at point q_0 at time t_0. This type of query has one more parameter than moving-Voronoi queries; this is v_d, the speed at which the query point (dog) can travel. Notice that this type of query also differs from moving-Voronoi queries in that knowing $p_1(t_0), \ldots, p_n(t_0)$ does not provide enough information for its solution; the solution may be highly dependant upon the future movement of the points.

This paper contains two main results. The first is a randomized incremental algorithm that builds a data structure $\mathcal{M}$ which permits efficient solution of moving-Voronoi type queries. This structure answers such queries in $O(\log^2 n)$ expected

time, where the expectation is taken over the order in which the postmen, $p_i(t)$, are inserted into the structure. The second result is an equivalence between moving-Voronoi queries and dog queries that will hold whenever the dog's speed, v_d, is faster than those of all of the postmen. If this condition – guaranteeing that given enough time a query dog can catch every postman – holds, then the equivalence permits the use of $\mathcal{M}$ to also solve dog queries in expected $O(\log^2 n)$ time.

Moving-Voronoi-type queries have already been addressed, albeit tangentally, in the literature. A series of recent papers [Roo91], [Roo90], [AR92], [RN91], [FL91] study the Voronoi diagram of moving points. These papers examine changes in the Delaunay triangulation (the planar dual of the Voronoi diagram) of moving points, and how to detect when they occur, but do not address the problem of point location. [GMR91] give an $O(n^3)$ upper bound on the number of such topological changes when the points are moving with constant speed as in (1) and a slightly more than cubic bound for some more general motions.

General dog-type queries do not seem to have been discussed previously. See, however, the description of Voronoi diagrams in rivers in [Sug92] which discusses a problem equivalent to a special case of a dog-type query, one in which all of the postmen move with the same velocity.

We can imagine two different approaches to the moving-Voronoi problem; one approaches the problem as a collection of an infinite number of planar point-location problems. The other transforms the problem into a three-dimensional point-location one.

The approach followed in this paper is the second one. We start, though, by quickly describing the first. Consider the Voronoi diagram of $p_1(t), \ldots, p_n(t)$, starting with $t = 0$. As time passes and t increases the Voronoi diagram changes. In the beginning, the changes are restricted to expansions, contractions, and movement of its edges but the basic topology of the Voronoi diagram remains constant. During this entire period its dual, the Delaunay triangulation, also remains constant.

After enough time has passed some edges disappear and/or new edges appear in the Voronoi diagram. This causes its topology to change and this in turn corresponds to a change in the Delaunay triangulation of the points. For some further period of time the Voronoi diagram will again maintain the same topology while its edges move and change size until it changes topology yet again, corresponding to yet another change of Delaunay triangulation. This process continues forever but [GMR91] tell us that there are at most $O(n^3)$ changes in the Delaunay triangulation so there are at most $O(n^3)$ corresponding topological changes in the Voronoi diagram. After the

last topological changes have been encountered the only further modifications can the be expansion/contraction and movement of edges.

This approach of viewing the Voronoi diagram as changing with time leads to the first solution to the moving-Voronoi problem. Goodrich and Tammasia[GT91] describe an algorithm that enables point location in a planar subdivision that is modified by shrinking/growing and adding/subtracting edges; their algorithm uses $O(T)$ space and permits $O(\log^2 T)$ point location in every one of the constructed planar subdivisions where T is the complexity of the item to be stored. In the moving-Voronoi case $T = O(n^3)$ which is the size of the original Voronoi diagram plus the number of changes that it undergoes. Their algorithm can therefore be used to build a point location structure that, for arbitrary t_0, permits point location in the Voronoi diagram of $p_0(t_0), \ldots, p_n(t_0)$ in $O(\log^2 n)$ time. In other words moving-Voronoi queries can be solved in $O(\log^2 n)$ time. The reason we do not follow this approach in this paper it that it is not clear how to modify the technique to also solve dog queries.

The approach sketched in the previous paragraph solves a moving-Voronoi query by fixing the time t_0 and performing a point location in the Voronoi diagram of $p_1(t_0), \ldots, p_n(t_0)$. That is, it reduces the general problem to being able to solve an infinite number of two-dimensional point location problems. The approach that we follow in this paper is totally different. It views the problem as one of point-location in a three-dimensional, (space, time), cell structure. This viewpoint permits the exploitation of a correspondence between the moving-Voronoi problem and the dog-bites-postman one.

We construct the point-location data-structure for the moving-Voronoi by using a randomized-incremental procedure which is a variation of one devised by Guibas, Knuth and Sharir [GKS92] for solving the static Voronoi diagram problem. The GKS result for static cases is contained in the following theorem: (the radial Voronoi-diagram, which we describe further in the next section, is a special triangulation of the standard Voronoi diagram)

Theorem 1 *The radial Voronoi diagram of n sites inserted in random order can be maintained in $O(\log n)$ expected update time and $O(n)$ expected space. Any fixed query point is located in $O(\log^2 n)$ expected time (the expectation is taken over the random order of insertion).*

Our modification of the GKS result will construct a data structure which permits finding the nearest postman to q_0 at time t_0 in expected $O(\log^2 n)$ time where the expectation is only taken over the order in which the postmen were inserted into the data structure and not over the values of q_0, t_0.

Our approach to solving the "dog" problem is to show that, if the dog is faster than all of the postmen, i.e., $v_d > |v_i|$, $i = 1, \ldots, n$, then the three dimensional cell-complex that describes the dog problem is topologically equivalent to the one describing the moving-Voronoi one. A proper understanding of this transformation will enable the same data structure to solve both moving-Voronoi queries and dog ones.

The structure of the paper is as follows: in section 2 we quickly review the basic ideas of the GKS algorithm and data-structure. In section 3 we discuss how to solve the moving-Voronoi problem. In section 4 we define a correspondence between moving-Voronoi queries and dog queries which permits us to solve dog queries using the data structure developed in Section 3. In section 5 we describe how to modify the algorithm so that it permits not only insertions but deletions of postmen in a randomized model. We conclude in section 6 by reviewing our results and presenting an open problem.

2 The GKS Algorithm

As previously mentioned our algorithm for the moving-Voronoi problem builds quite strongly upon previous work of Guibas, Knuth and Sharir. To make this paper as self-contained as possible we briefly sketch the ideas and analysis behind the GKS algorithm. For further details we direct the interested reader to [GKS92]. The algorithm sketched below differs from the basic one appearing in [GKS92] in that its implementation uses pointers to travel from a triangle to the exterior triangles that destroy it; as noticed by Guibas *et al.*, this modification does not affect the algorithm's running time. We utilize the modified algorithm because it is easier to generalize to the moving point case.

Strictly speaking, GKS do not construct the Voronoi diagram of their point set. They construct its *radial triangulation.* That is, they triangulate each Voronoi cell by drawing edges from the site at its center to all of the vertices on the cell's boundary. In Figure 2 the Voronoi edges are drawn with bold lines, the other triangulation edges with dotted ones. Their algorithm allows them to locate points in this triangulation and therefore in the Voronoi diagram. In what follows we call the triangulation the *radial Voronoi diagram.*

A fundamental observation is that each (bounded) triangle depends upon exactly four sites (three sites for unbounded triangles). For example, in Figure 2, the sites p, q, r and s define the shadowed triangle ptu (and also qtu), where t and u are the

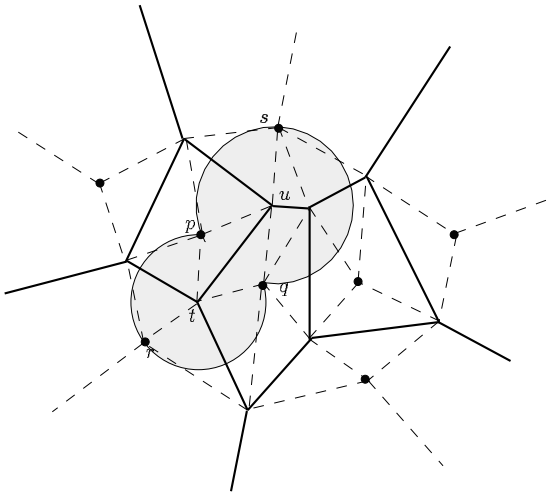

Figure 2: The triangulated Voronoi diagram. The solid edges are Voronoi diagram ones; the dotted edges are added to form the radial triangulation.

Voronoi vertices which are the centers of the empty disks circumscribing Delaunay triangles pqr and pqs.

Given points $p_1, p_2, \ldots, p_n$ the GKS algorithm works by incrementally constructing the radial Voronoi diagrams of the sets $\{p_1, p_2, \ldots, p_k\}$ where $k = 1, \ldots, n$.

Consider what occurs when a new site p_k is inserted into the radial Voronoi diagram of $p_1, \ldots, p_{k-1}$, creating the radial Voronoi diagram of $p_1, \ldots, p_k$. Most of the radial Voronoi-diagram remains the same but, near p_k, some new triangles will be created and some old triangles destroyed. The newly created triangles are of two types. *Interior* type triangles are the ones that triangulate the Voronoi cell of p_k. *Exterior* type triangles are the ones which belong to Voronoi regions of other cells. Note that if $j < k$ then exterior triangles can be created in the Voronoi cell associated with p_j if and only if p_j is a Voronoi neighbor of p_k, i.e. their Voronoi cells share an edge. Furthermore, if p_j is a Voronoi neighbor of p_k at most three exterior triangles may be created in its Voronoi cell.

A triangle that existed in the radial Voronoi diagram of $p_1, \ldots, p_{k-1}$ but does not exist in that of $p_1, \ldots, p_k$ is said to be *destroyed* by the insertion of site p_k.

Suppose that some triangle destroyed by the insertion of p_k was, in the old diagram, part of the Voronoi cell associated with site p_j. In the new diagram the

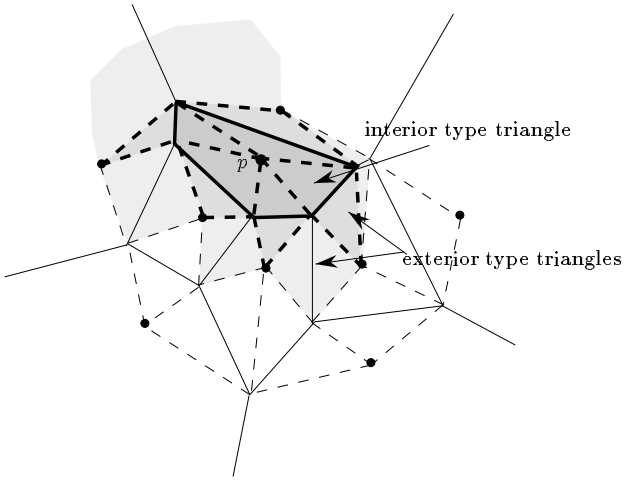

Figure 3: The shaded region contains the triangles created the insertion of site p.

region covered by the old triangle must be contained in the union of the Voronoi cell of p_k with at most two new exterior triangles created in the cell associated with p_j. This fact permits the creation of an efficient point location structure.

See [GKS92] for proofs of the above statements and details.

The location structure

The structure built during the insertion of n sites can be represented as n successive parallel horizontal planes, each plane containing the triangles created by the insertion of the corresponding site.

If a triangle is destroyed when a site p is inserted, then it overlaps with one or two exterior triangles and the Voronoi region of the new point. Pointers are created from the destroyed triangle to the (at most two) exterior triangles that intersect it and from the destroyed triangle to the site p. The interior triangles created by the insertion of p are organized around p in a search structure sorted in polar order; this structure is called a *polar sorted list.*

Finally, the structure contains pointers from each triangle to the ones adjacent to it. Notice that these relations are not necessarily between triangles in the same plane and may change when new sites are inserted.

Let V_i, $i = 1, \ldots, n$ be the radially triangulated Voronoi diagram of points $p_1, \ldots, p_i$. To locate a query point q in the final radial Voronoi diagram V_n, we walk through all the triangles on successive levels that contain q. We use the pointers from destroyed triangles to let us skip levels.

The location process starts by finding the unbounded triangle containing q in the triangulated Voronoi diagram V_3.

The general step in the algorithm assumes that we know the triangle T that contains q on some level k, i.e. in V_k. Either T is in V_n and we are done, or T is destroyed by the insertion of some site p_i. In this second case we must find the triangle on level i containing q.

To do this we first check if q is inside one of the exterior triangles on level i pointed to by T. If q is outside these triangles then q must be located in one the interior triangles arrayed around p_i; this triangle can be found by performing a binary search in the polar sorted list. In both cases the triangle in V_i containing q has been found and the search can continue.

Update algorithm

When a new point is added, it is first located in the structure and found to be in some triangle T. The current radial triangulation is then explored using the adjacency relationships to find all the destroyed triangles. A new horizontal plane is then created, and all new triangles (exterior and interior) are computed from the destroyed ones. Pointers from the destroyed triangles to the appropriate ones on the new level are created. Using the adjacency relations it is possible to find the new interior triangles directly sorted in polar order around the new points. Finally, the adjacency relations are updated.

Analysis

This section give an idea of the randomized analysis of this algorithm using the backward analysis technique [Che85, Sei91, Dev92]. We quickly sketch a proof that the expected cost of locating an arbitrary point in the radial Voronoi diagram is $O(\log^2 N)$ and that the expected cost of inserting point p_n into V_{n-1} is $O(\log n)$.

Location of a given point

During the location of a point q, the algorithm traverses all triangles containing q which have been in the triangulated Voronoi diagram at some stage of the construction. Recall that we only visit a triangle T at the level in which it is first created. The

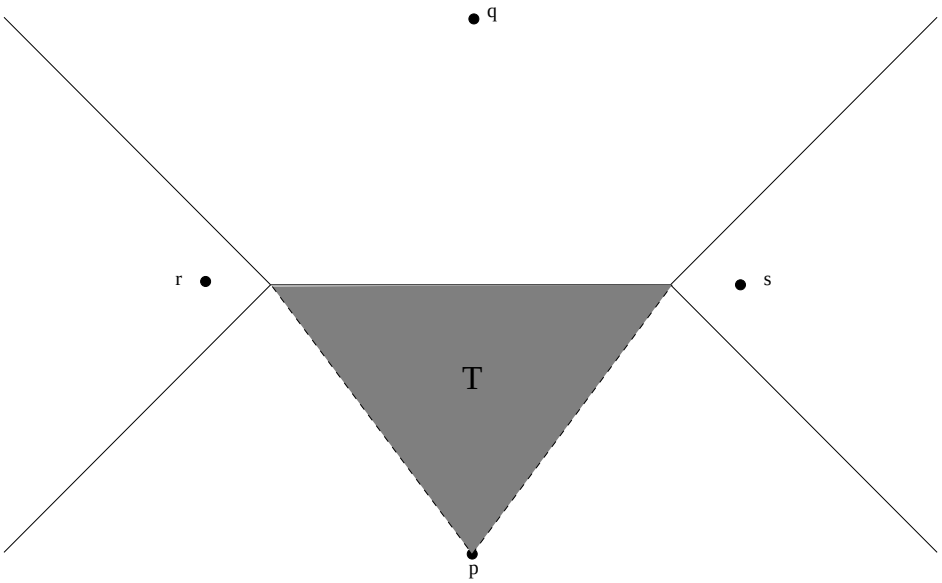

Figure 4: Triangle T in the radial triangulation is defined by at most 4 points p, q, r, and s.

next paragraph proves that, for any query point q, the expected number of levels on which a triangle that contains q is created is $O(\log n)$ if the sites were inserted in a random order. Since the next triangle on the path followed by the location algorithm is determined either directly in $O(1)$ time – if it is an exterior triangle – or by searching in a polar sorted list in $O(\log n)$ time – if it is an interior one – the total expected time complexity of the location process is $O(\log^2 n)$.

Now, we count the number of created triangles. The crucial fact here is that a triangle T in a radial triangulation is fully defined by (at most) 4 sites. (See Figure 4.) Assume that at stage k, the radial triangle containing q is T. T will only be physically present on the horizontal plane at level k if T was created by the insertion of the k^{th} site, i.e., if and only if the k^{th} site is one of the (at most) 4 sites defining T. If k is not one of these sites then triangle T has already been found and level k is not visited. Since the k sites were inserted in random order, a triangle at level k is visited with probability (at most) $\frac{4}{k}$ (if the k^{th} site is not one of these 4 defining ones then T was created at a lower level and was counted at that lower level). Therefore the probability that level k is visited is $O(\frac{4}{k})$.

The result is obtained by summing these probabilities for all levels.

$$\begin{aligned} &\text{expected number of created triangles} \\ &= \sum_{k=1}^{n} \text{expected number of created triangles at level } k \\ &\leq \sum_{k=1}^{n} \frac{4}{k} \;=\; O(\log n) \end{aligned}$$

Insertion of a site

The difference between locating a site to be inserted and locating an arbitrary query point is that a site to be inserted is, by hypothesis, a random one from the set of all sites. This fact can be used to prove that the expected length of the polar sorted list of each level used during the location procedure is constant which in turn will lead to an expected $O(\log n)$ location time for insertion.

To provide some intuition as to why this is so recall that, because the radial Voronoi diagram is a planar graph, the average degree of its vertices is at most 6. Also, the expected length of the polar sorted list at the k^{th} level is exactly the degree of the k^{th} site in the radial Voronoi diagram at stage k. Since the k^{th} site is equally likely to be any site and the average degree of a site is at most 6, the expected length of a polar sorted list is 6. Of course this off-the-cuff argument is not rigorous and we must provide an exact proof.

Our proof uses the well known geometric fact that any site can only be the nearest neighbor of at most 6 other sites

More precisely, we can bound the time spent at level k during the insertion of the n^{th} site q as follows. Let p_k be the k^{th} site to be inserted into the diagram: the k^{th} level is visited if the triangle in V_k containing q has p_k as one of its defining points. This occurs with probability $\frac{4}{k}$ since this triangle is defined by 4 sites among the k present in the diagram. If the nearest neighbor of q (among the k first sites) is not p_k, then the new triangle containing q is an exterior one and is determined in constant time among the two possible candidates. Thus the total expected amount of time spent by the algorithm searching exterior triangles is

$$\sum_{k=1}^{n} \frac{4}{k} = O(\log n).$$

If q is not in an exterior triangle it must be in an interior one created at level k; this only occurs when p_k is the nearest neighbor of q, which happens with probability

$\frac{1}{k}$. In this case, the polar sorted list around p_k must be searched; the cost of this search is certainly bounded by the size of the polar sorted list which is the degree of p_k in the diagram of the k first sites.

Now use the fact that q is also chosen at random from the given set. More particularly, consider the set P of $k+1$ sites formed by the k first sites plus q: q is a random site from this set and p_k is its nearest neighbor, something which occurs with probability $\frac{1}{k}$. Since a given site may be the nearest neighbor of at most 6 other sites in P any point can be the nearest neighbor of a random point with probability at most $\frac{6}{k+1}$. The average degree of the nearest neighbor of a random point q is therefore

$$\sum_{i=i}^{k} \Pr(p_i \text{ is a nearest neighbor to } q) \cdot d_i \leq \frac{6}{k+1} \sum_{i=1}^{k} d_i \leq \frac{6}{k+1} \cdot 6k < 36$$

where d_i is the degree of p_i in V_k and $\sum_{i=1}^{k} d_i \leq 6k$ because the Voronoi diagram is a planar graph.

Multiplying by $1/k$, the probability that p_k is the nearest neighbor of q, yields that the average amount of time spent searching the polar list at level k is $O(\frac{36}{k})$. Summing over all k yields that the total time needed to search polar lists while locating q is $O(\log n)$. Adding the previously calculated time needed to search external triangles we find that the total expected time needed to find q is $O(\log n)$. (This proof differs from that presented in Guibas *et al.* [GKS92].)

To conclude this section we point out that the expected number of triangles at each level is constant. Therefore the expected total number of changes caused by the insertion of q will be $O(1)$. This also implies that the expected size of the total location structure is $O(n)$.

3 Randomized incremental construction of the moving Voronoi diagram

We now address the problem of moving points. A site is a two-dimensional point moving with constant velocity: $p_i(t) = (x(t), y(t)) = q_i + v_i t$, $i = 1, \ldots, n$, where $q_i \in I\!R^2$ is the point's location at time 0, and $v_i \in I\!R^2$ is its velocity. For the sake of simplicity we assume that the velocity is constant but the technique that we introduce can be used in many cases where the motion is not constant, e.g., $x(t)$, $y(t)$ are polynomials in t of bounded degree k. At time t, the moving sites define a radial Voronoi diagram. As t increases, this diagram changes.

A natural visualization of the problem considers the 3D space (x, y, t) in which x, y span the horizontal plane to which t is orthogonal. In this space a moving site is a non-horizontal line (or a curve of degree k if the components of $p_i(t)$ are polynomials of degree k).

Let $V(t)$ be the radial Voronoi diagram of $p_1(t), \ldots, p_n(t)$. Consider any specific Voronoi vertex of $V(t)$. As time increases this vertex sweeps up to become an curve in 3-space. An edge in $V(t)$ will sweep up to become a surface in 3-space and a radial triangle in $V(t)$ sweeps up to become a region in 3-space. We call this collection of curves, surfaces (faces), and regions the *Moving Voronoi-Diagram* and denote it by $\mathcal{M}$. The vertices of $\mathcal{M}$ will be the points at which Voronoi edges/vertices of the two-dimensional $V(t)$ appear and disappear. These are exactly the points which we previously identified as the topological changes in the Voronoi-diagram as it moves.

For algorithmic purposes there are three facts concerning $\mathcal{M}$ which must be emphasized. The first is that $\mathcal{M}$ is in fact a subdivision of 3-dimensional space and finding the region $R \in \mathcal{M}$ which a query point appears immediately solves a moving-Voronoi type query. This is because every query point that appears in the same region has the same postman as a nearest neighbor.

The second fact concerns the number of sites that can define a region in $\mathcal{M}$. A region (cell) of $\mathcal{M}$ corresponds to the space swept out by a triangle of $V(t)$ between the time when it appears and the time when it disappears. The moving triangle is defined by 4 moving sites (as in the non-moving case), and the appearance and the disappearance of the triangle correspond to topological changes in the 2D radial Voronoi diagram, i.e., the disappearance and the appearance of another site in the (empty) disk circumscribing three of the four sites. To summarize, a region of the 3D moving radial Voronoi diagram is defined by at most 6 moving sites (see Figure 5). Fewer than 6 sites define a region when the moving triangle is unbounded or when a site which caused the triangle to appear or disappear is one of the four sites defining the moving triangle.

The third fact is that the complexity of $\mathcal{M}$, as measured by the number of its vertices, edges, faces, and regions is $O(T(n))$ where $T(n)$ is $O(n)$ – the complexity of $V(0)$ – plus the number of topological changes that occur in $V(t)$ as t increases. This follows from the basic properties of the static Voronoi-diagram.

This section modifies the GKS algorithm to work on the moving Voronoi diagram and proves the following theorem.

Theorem 2 *The moving radial Voronoi diagram of a set $\mathcal{S}$ of n moving sites inserted in random order can be maintained using $O(f_{\mathcal{S}}(k)\log k/k)$ expected insertion time and $O(f_{\mathcal{S}}(n))$ expected total space for the insertion of the k^{th} point where $f_{\mathcal{S}}(k)$*

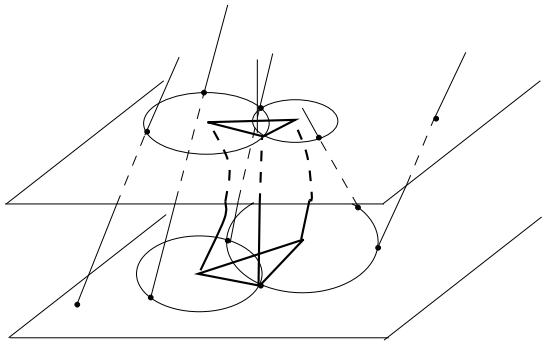

Figure 5: A region of the 3D moving diagram

is the expected size of the moving diagram for a random sample of size k of $\mathcal{S}$. Any fixed query point can be located in $O(\log^2 n)$ expected time (all expectations are taken over the order of insertions of the sites into the data structure).

The complexity depends on $f_{\mathcal{S}}$ since the size of the moving Voronoi diagram can vary between $\Omega(n)$ and $O(n^3)$. So, our algorithm is partially output sensitive. More precisely, it is not sensitive to the size of the output but to the expected size of intermediate results, a phenomenon which occurs in the analysis of other semi-dynamic randomized algorithms [BDS+92]. Furthermore, no tight upper bound for the size of the moving Voronoi diagram are known. It is not even known if it can be larger than $O(n^2)$ [GMR91]. If the worst-case size of the moving Voronoi diagram is $\Theta(n^3)$ then our algorithm will take $O(n^3 \log n)$ expected time and use $O(n^3)$ space to construct the data structure.

3.1 Algorithm

The location structure

This section describes the structure used for locating query point $q = (x_q, y_q, t_q)$ in the 3D space. This solves the question "at given time t_q, what is the nearest site $p_i(t_q)$ to the point (x_q, y_q)?".

Following the notation developed above we use $\mathcal{M}_k$ to denote the cell-structure of the 3D moving-Voronoi diagram of postmen $p_1, \ldots, p_k$. The algorithm works in the same way as does the GKS algorithm described in Section 2. It successively

constructs all of the $\mathcal{M}_k$, $k = 1, \ldots, n$ storing at "level" k the new regions created by the insertion of the corresponding moving site.

Suppose that region (cell) R which contains point q in $\mathcal{M}_{k-1}$ no longer exists in $\mathcal{M}_k$. Then we say that R is *destroyed* by the insertion of site k. In such a case R is contained in the union of some new regions stored at level k. As in the 2D case, the regions created by the insertion of site p_k are divided into interior and exterior regions.

Interior regions are the ones which are swept-up interior triangles. Equivalently, these are the regions which have a segment of the line $p_k(t)$ as a supporting edge. Exterior regions are the swept-up parts of exterior triangles, i.e. the regions which are not interior. Both types of regions will be stored in a way which will enable us to easily determine which of them contains q.

Exterior regions Fix time t and consider what happens when going from the radial Voronoi diagram of sites $p_1(t), \ldots, p_{k-1}(t)$ to the radial Voronoi diagram of sites $p_1(t), \ldots, p_k(t)$. Suppose $p_i(t)$'s Voronoi cell is a neighbor of $p_k(t)$'s in the new diagram. We have already seen that $p_i(t)$'s cell contains at most 3 external triangles. Now sweep these triangles up in time; they become external regions in $\mathcal{M}_k$ (associated with postman p_i). Note, though, that there may be many more than 3 exterior regions associated with p_i on level k. It is just that at any given time, t, only 3 of them may exist.

For each p_i that has external regions on level k do the following: store the entire list of its external regions sorted by the times t that they start existing and the times t that they cease to exist. These times correspond to topological changes in the moving-Voronoi diagram so there may be at most $O(n^3)$ items in each list. Therefore, given any time t we can search the list in $O(\log n)$ time to find the 3 external triangles that exist in $p_i(t)$'s Voronoi region.

Suppose region R containing points closest to postman p_i is destroyed by the insertion of site p_k. Create a pointer from R to the sorted list of exterior regions associated with p_i that are created by the insertion of p_k.

Let $R(t)$ denote a cut at specific time t of region R. $R(t)$ is a triangle in the two-dimensional radial Voronoi diagram of $p_1(t), \ldots, p_{k-1}(t)$. $R(t)$ will overlap at most two of the exterior triangles in the Voronoi cell associated with $p_i(t)$ in two-dimensional radial Voronoi-diagram of $p_1(t), \ldots, p_k(t)$.

Given query point $q = (x_q, y_q, t_q) \in R$ it is now simple to discover, in $O(\log n)$ time, whether a new exterior region created by the insertion of p_k contains q. Let p_i be the postman that R is associated with. Simply search the sorted list of p_i's

exterior regions for t_q to find the three exterior regions that exist at time t_q and check if one of them contains q.

Interior regions The set of all interior regions created by the insertion of site k is exactly the set of all the regions which have the line $p_k(t)$ as a supporting edge. Note that a region can not be both interior and exterior.

As in the non-moving case, if the query q does not belong to an exterior region, we must examine the whole set of interior regions to determine the one that contains q.

In the non-moving case, the interior triangles were sorted according to the angle θ that their supporting edges form with the horizontal around the newly inserted site.

In this case, a two dimensional structure is used: the first dimension is the time t and the second is the polar angle θ_t around $p(t)$. The whole set of interior triangles created by the insertion of $p(t)$ defines a monotone subdivision of this 2D space (t, θ_t). To find a query point q we map it to (t_q, θ_q) (where θ_q is the angle between $\overline{p(t)q}$ with the x-axis), and this mapped point is located in the above subdivision using any static point location algorithm [EGS86, CS89, Pre90]. A vertex of this subdivision corresponds to a topological change in the moving-Voronoi diagram. Since there are $O(n^3)$ of these and point location can be performed in time proportional to the logarithm of the subdivision size this implies that the internal region containing q can be found in $O(\log n)$ time.

Point Location Location of a query point q in the final moving-Voronoi diagram $\mathcal{M}_n$ follows the paradigm laid down by the GKS algorithm. The algorithm starts by finding the region $R \in \mathcal{M}_3$ that contains q.

The generic step of the algorithm assumes that we know the region R that contains q on some level k, i.e., in $\mathcal{M}_k$. Either $R \in \mathcal{M}_n$ and we are done or R is destroyed by the insertion of some site p_i. In this second case we must find the region on level i containing q. Following the procedures described in the previous subsections we first check in $O(\log n)$ time to see if q is contained in an exterior region. If it is not we use $O(\log n)$ time to search the monotone subdivision to find the proper internal region containing q.

In both cases the new region on level i containing q has been found and the search can continue.

Update algorithm

To insert a new moving site $p(t)$, it is necessary to determine all the regions destroyed by the insertion of the new site, and to make the necessary updates. The regions to be destroyed are found in the following way:

First use the data structure to locate the region containing $p(0)$. This can be thought of as mimicking the operation of the two-dimensional GKS algorithm. Next, use the adjacency relationships to find all the regions that will be destroyed. This is clearly possible, since the set of regions to be destroyed is connected (in any plane $t =$ constant, it is star shaped). The idea is to start from the initial region and work our way "out" stopping at each region that is not destroyed.

After the destroyed regions are found, a new level corresponding to the new site is created in the structure. All new regions are computed from the destroyed ones; they are organized in the location structure described above comprising some sorted lists and a planar monotone subdivision. Notice that these structures are static and any location structure can be used, randomized [CS89], or not [EGS86]. The adjacency relations must then be updated by modifying appropriate pointers. Notice that all new pointers will be between regions visited during the update because all the regions neighboring the destroyed area were visited during the search for the destroyed regions.

3.2 Analysis

Location of a given point

The location algorithm follows the sequence of regions containing the query point q through the n levels. Given a region containing q the next region is determined in $O(\log n)$ time by first checking the sorted list to determine if q is in an exterior region, and, if it isn't, by performing a planar point location in a monotone subdivision to find the proper interior region.

The number of regions in the sequence is expected $O(\log n)$ as in Section 2. This result follows from the fact that a region is defined by at most 6 of sites so a given region, the one containing q, will be constructed at the k^{th} level only if the k^{th} site is one of these 6, which occurs with probability $O(\frac{1}{k})$. Therefore the expected number of regions in the sequence is $O\left(\sum_{k=1}^{n} \frac{1}{k}\right) = O(\log n)$.

The expected time complexity of the location of any query point is therefore $O(\log^2 n)$, where the expectation is taken over the insertion order of the sites.

Insertion of a site

The insertion procedure is divided into two parts. The first finds some region destroyed by the insertion of p_k. The second works its way out from this initial region to find all the destroyed regions and update the data structure.

We find the first region destroyed by looking for a two-dimensional region in the radial Voronoi diagram of $p_1(0), \ldots, p_{k-1}(0)$ that is destroyed by the insertion of $p_k(0)$. This, in fact, is done exactly as in the two-dimensional GKS algorithm and the analysis performed there applies here. We can find such a region in $O(\log n)$ expected time. The search for an initial destroyed region is in fact the problem of locating $p(0)$ in the two-dimensional Voronoi diagram, so the analysis performed in the two-dimensional case applies and the expected time for the search is $O(\log n)$.

The cost of the update step, is output-sensitive; it is necessary to compute the expected number of new regions, but also the expected number of adjacency relations that must be updated. Standard techniques of analyzing randomized algorithms can be used to count the expected number of new regions. After the insertion of the k^{th} moving site, the moving Voronoi diagram has $f_{\mathcal{S}}(k)$ regions (expected) and, since a region is determined by at most 6 sites, the expected number of regions created by the insertion of the k^{th} site is less than $6\frac{f_{\mathcal{S}}(k)}{k}$. Since the number of neighbors of a regions is not bounded, the analysis of the number of adjacency relations involved in the update algorithm requires a hint first used by Boissonnat and Teillaud [BT93, BDS+92] (the so called bicycle hint). The idea is to associate adjacency relations with regions. Two adjacent regions correspond to two 2D adjacent moving triangles, these triangles are determined by 5 sites (4 sites each, but 3 are common) and the appearance and disappearance of the adjacency at two critical instants are determined by two other sites. In fact an adjacency is determined by at most 7 sites; if the expected number of adjacency relations in the moving Voronoi diagram is $f_{\mathcal{S}}(k)$ (it is the number of faces in the diagram which is less than $f_{\mathcal{S}}$, the total complexity of the diagram) the number of adjacency relations modified by the insertion of the k^{th} site is less than $7\frac{f_{\mathcal{S}}(k)}{k}$.

We must also build the point location data structure for the monotone subdivision and the sorted lists of exterior regions. The size of the monotone subdivision is exactly the number of internal regions created plus the number of adjacencies/vertices between them. As shown above this is $O\left(\frac{f_{\mathcal{S}}(k)}{k}\right)$. The location structure for the monotone subdivision can be constructed in $O\left(\frac{f_{\mathcal{S}}(k)\log k}{k}\right)$ using any classical technique, randomized or not.

There are at most $O\left(\frac{f_S(k)}{k}\right)$ exterior regions created by p_k (since they are a subset of the total set of regions created) and each region appears in exactly one sorted list so the total size of the lists of exterior regions is also $O\left(\frac{f_S(k)}{k}\right)$. The collection of these lists can therefore be built in $O\left(\frac{f_S(k)\log k}{k}\right)$ time .

The algorithm therefore updates the diagram in $O\left(\frac{f_S(k)\log k}{k}\right)$ time using $O\left(\frac{f_S(k)}{k}\right)$ space.

If the postmen travel with non-constant velocity the algorithm can, in many cases, be generalized without difficulty. The subdivision in space (t, θ_t) is still a monotone subdivision (the degree of the edges may change) and the function f_S can be different, but the general ideas still work.

4 The dog and postmen problem

We now consider dog queries. Imagine that the moving sites are postmen. A query asks "a dog capable of running at maximum speed v_d is put outside at time t_d and location (x_d, y_d). It will run and bite a postman. Which postman can it reach in a minimum time?"

These types of queries differ from the "moving Voronoi" type addressed in the previous section. It is possible that the dog might be able to reach a postman further away (that is travelling towards it) quicker than it can reach a nearby one (that is travelling away from it). The answer to the query depends strongly on v_d. For example, in Figure 6, although postman **A** is closer to the dog than postman **B**, the dog will be able to catch **B** before being able to catch **A** because **A** is moving away from the dog.

In what follows we will assume that $v_d > |v_i|$, $i = 1, \ldots, n$, i.e., the dog is faster than all of the postman. This ensures that the dog will always be able to catch at least one postman. Later, we will discuss what happens if the dog is not as fast as some of the postmen. Note that as we let $v_d \to \infty$ the postman that can be reached quickest becomes the nearest postman. In this sense at least, the moving Voronoi problem can be thought of as the limiting behavior of the dog problem.

We will prove the following theorem:

Theorem 3 *The postman that any fixed query dog can reach quickest can be found in $O(\log^2 n)$ expected time, with the same space and preprocessing time as in Theorem 2.*

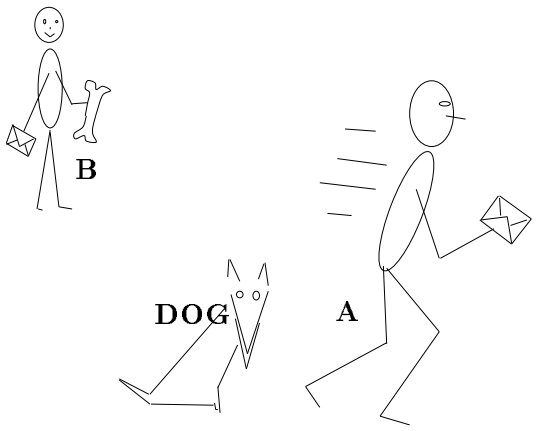

Figure 6: A dog query not reaching the nearest site

We do this by showing that if the dog is faster than all of the postmen then there is an efficient way to transform a dog query into a moving-Voronoi query.

In the previous section we described the Moving-Voronoi diagram of n points which we denoted by $\mathcal{M}$. Now consider its topology: $\mathcal{M}$ can be considered as the stacking, one on top of another, of an infinite number of static Voronoi diagrams, $V(t)$, $t \in [0, \infty)$, where $V(t)$ is the Voronoi diagram of points $p_1(t), \ldots, p_n(t)$.

First recall the following facts about a static Voronoi diagram, such as $V(t)$: a point is a Voronoi vertex if and only if it is the center of an empty (horizontal) circle that has three sites on its circumference and a point is on a Voronoi edge if and only if it is the center of an empty circle that passes through two sites.

Now consider the structure of $\mathcal{M}$. A Voronoi vertex in the static diagram $V(t)$ will be swept-up in time to become an edge in $\mathcal{M}$; an edge in $V(t)$ will be swept up to become a face of $\mathcal{M}$; a two-dimensional region in $V(t)$ will be swept up to become a three-dimensional region of $\mathcal{M}$. Finally, a point is a vertex of $\mathcal{M}$ if and only if it is a point where some Voronoi vertices of $V(t)$ appear and disappear. Thus, a point is a vertex of $\mathcal{M}$ if and only if it is the center of an empty horizontal circle that contains four points on its circumference.

We now return to the dog problem. Given a query (x_d, y_d, t_d, v_d), it is clear that the dog can reach any point in space contained in a vertical cone of apex (x_d, y_d, t_d) and angle arctan v_d which we denote by C_d. The postmen travel along the lines $p_i(t)$. Therefore the answer to the query is the lowest intersection of one of the lines $p_i(t)$

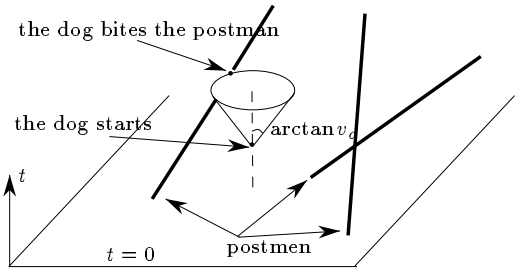

Figure 7: A query dog

with C_d. (see Figure 7). The different answers to this query split space into different regions in each of which the answer remain the same. Denote the three dimensional cell-structure that this splitting induces by $\mathcal{D}$. It is in fact the Voronoi diagram of the lines $p_i(t)$ with the convex distance function associated to an upward vertical cone of angle $\arctan v_d$ (bounded below by a horizontal plane)[2].

Although the "moving-Voronoi" and "dog" problems seem so different we will soon see that, under the hypothesis that the dog is faster than the postmen, i.e., $v_d > |v_i|$, $i = 1, 2, \ldots, n$, they are actually quite similar[3]

First of all, under this hypothesis, verifying the solution to a dog query is easy. Suppose that $d = (x_d, y_d, t_d, v_d)$ is a dog query such that the quickest reachable postman is p_i. Let $t_{i,d}$ denote the time when the dog reaches p_i (which depends only upon d and p_i): $t_{i,d}$ is the unique positive solution of the equation $|p_i(t_{i,d}) - (x_d, y_d)| = v_d(t_{i,d} - t_d)$ and is also the unique time that the line $p_i(t)$ intersects cone C_d.[4]

Lemma 1 *The quickest reachable postman for a dog query $d = (x_d, y_d, t_d, v_d)$ is p_i if and only if the nearest neighbor of (x_d, y_d) at time $t_{i,d}$ is p_i.*

Proof.

[2] See [Kle89] for more on convex distance functions. Unfortunately there are no general purpose efficient algorithms for finding Voronoi diagrams for convex distance functions.

[3] The two problems are even more closely related than we prove here. It is possible to show that $\mathcal{M}$ and $\mathcal{D}$ are topologically equivalent: there is a continuous one-one function from 3-space onto itself that maps the vertices, edges, faces and regions of $\mathcal{M}$ into those of $\mathcal{D}$. We do not go into the details here because they are not required for the algorithm.

[4] Each line $p_i(t)$ intersects cone C_d exactly once because the cone is expanding with speed $v_d > |v_i|$.

Suppose the dog can catch p_i quickest but the nearest neighbor of (x_d, y_d) at time $t_{i,d}$ is p_j. Then the point $p_j(t_{i,d})$ is *inside* the cone C_d so the line $p_j(t)$ is inside C_d at time $t_{i,d}$. This line must therefore intersect the cone at some time $t_{j,d} < t_{i,d}$, i.e., the dog can catch p_j before catching p_i, contradicting our assumption.

Conversely, suppose that p_i is the nearest site to (x_d, y_d) at time $t_{i,d}$ but the dog can catch p_j before catching p_i, i.e. $t_{j,d} < t_{i,d}$. By definition $|p_i(t_{i,d}) - (x_d, y_d)| = v_d(t_{i,d} - t_d)$. Since the speed $|v_j| < v_d$ this implies

$$\begin{aligned} |p_j(t_{i,d}) - (x_d, y_d)| &\leq |p_j(t_{i,d}) - p_j(t_{j,d})| + |p_j(t_{j,d}) - (x_d, y_d)| \\ &= |v_j|(t_{i,d} - t_{j,d}) + v_d(t_{j,d} - t_d) \\ &< v_d(t_{i,d} - t_d) = |p_i(t_{i,d}) - (x_d, y_d)|. \end{aligned}$$

Thus, at time $t_{i,d}$, p_j is nearer to (x_d, y_d) than p_i, leading to a contradiction. □

Recall now that our algorithm for locating a query point in $\mathcal{M}$ actually found the location of the query point in all of the diagrams $\mathcal{M}_k$, $k \leq n$. We use this fact together with the previous lemma to develop an algorithm for solving dog queries. It is just a variation of our algorithm for solving moving Voronoi queries and uses the same data structure.

Let $S_k = \{p_i : i \leq k\}$ be the set of the first k postmen. The idea of our new algorithm is that we will walk down the levels $\mathcal{M}_k$, $k \leq n$, keeping track at each level of the postman $p_i \in S_k$ that the dog can reach quickest. Lemma 1 tells us that this is the postman that is nearest to the point (x_d, y_d) at time $t_{i,d}$. At each level we also keep track of the region $R \in \mathcal{M}_k$ such that $(x_d, y_d, t_{i,d}) \in R$; R is a region associated with postman p_i. To derive an efficient algorithm we will also need the following lemma:

Lemma 2 *Given a a dog query $d = (x_d, y_d, t_d, v_d)$ let p_i be the postman in the set S_{k-1} that can be reached quickest. and $R \in \mathcal{M}_{k-1}$ be the region such that $(x_d, y_d, t_{i,d}) \in R$. Suppose that the dog can reach postman p_k faster than it can reach postman p_i. Then the insertion of p_k destroys R in $\mathcal{M}_k$.*

Proof.

We are given that $t_{k,d} < t_{i,d}$. By definition R is a region such that if $(x, y, t) \in R$ then the closest postman to (x, y) at time t is p_i. To show that the insertion of p_k destroys R it therefore suffices to show that for the particular point $(x_d, y_d, t_{i,d}) \in R$ the point (x_d, y_d) is closer to p_k at time $t_{i,d}$ than to p_i or

$$|p_k(t_{i,d}) - (x_d, y_d)| < |p_i(t_{i,d}) - (x_d, y_d)|.$$

The proof is similar to that of the previous lemma:

$$\begin{aligned} |p_k(t_{i,d}) - (x_d, y_d)| &\leq |p_k(t_{i,d}) - p_k(t_{k,d})| + |p_k(t_{k,d}) - (x_d, y_d)| \\ &= |v_k|(t_{i,d} - t_{k,d}) + v_d(t_{k,d} - t_d) \\ &< v_d(t_{i,d} - t_d) = |p_i(t_{i,d}) - (x_d, y_d)|. \end{aligned}$$

□

Suppose now that we know that p_i is the postman in S_{k-1} that the dog can reach quickest and know the region $R \in \mathcal{M}_{k-1}$ such that $(x_d, y_d, t_{i,d}) \in R$ Lemma 1 says that p_i is a nearest neighbor to (x_d, y_d) at time $t_{i,d}$ among S_{k-1} so R is a region associated with p_i.

If R is not destroyed by the insertion of p_k then R remains a region of $\mathcal{M}_k$ and Lemma 2 implies that the dog can not reach p_k faster than it can reach p_i. Therefore p_i is still the postman in S_k that the dog can reach quickest and we do not have to do anything.

If R is destroyed by the insertion of p_k, then we first compare $t_{i,d}$ and $t_{k,d}$; if $t_{i,d} < t_{k,d}$ then p_i remains the answer to the dog query and we must find the new region in $\mathcal{M}_k$ containing $(x_d, y_d, t_{i,d})$. This region is associated with postman p_i so it is one of the external regions that R points to. Otherwise, $t_{i,d} < t_{k,d}$, p_k is the new answer to the dog query, and we must find the region in $\mathcal{M}_k$ containing $(x_d, y_d, t_{k,d})$ which will be an internal region associated with p_k.

We now describe the algorithm. It starts by finding which of the postman p_1, p_2, p_3 the dog can reach quickest, calculates the time t' at which the dog reaches the postman and then finds the region $R \in \mathcal{M}_3$ that contains (x_d, y_d, t').

In the general step we suppose that at the previous step of the algorithm we had junped to level k of the data structure and found that in S_k the dog can reach postman p_k quickest and also found the region $R \in \mathcal{M}_k$ such that $(x_d, y_d, t_{k,d}) \in R$. We now follow the pointers from R to level i where the insertion of p_i destroys R. If there is no such pointer then we are finished and p_k is the postman the dog can reach quickest.

Now, in $O(1)$ time, compare $t_{k,d}$ and $t_{i,d}$; if $t_{k,d} < t_{i,d}$ then p_k remains the answer to the dog query and we find the new region containing $(x_d, y_d, t_{k,d})$ in $O(\log n)$ time by searching the list of external regions that R points to. Otherwise, p_i is the new answer to the dog query and we must find the region containing $(x_d, y_d, t_{i,d})$. We do this in $O(\log n)$ time by searching the monotone subdivision describing the interior regions.

This algorithm takes $O(\log n)$ per level examined and the same analysis as used in the previous section shows that it examines $O(\log n)$ levels on average. Therefore the algorithm uses $O(\log^2 n)$ time on average.

5 Dynamization

The algorithms presented above are semi-dynamic in that sites/postmen can be added to, but not removed from the data structure. In what follows we sketch how to modify them to become fully dynamic in the randomized model. In a fully dynamic scheme random sites/postmen can be both added to and deleted from the structure.

In situations in which a randomized algorithm stores the history of the incremental construction of its data structure two different approaches have been developed for making the algorithm fully dynamic. One, proposed by Schwarzkopf [Sch91], stores the history of the structure for both insertions and deletions; the other used by Devillers *et al.* [DMT92, DTY92] and also by Clarkson *et al.* [CMS92] constructs a new history taking into account only the insertions of remaining sites. (Mulmuley [Mul91, MS91] describes yet another method which does not use the history of the construction.)

The reconstruction of a new history requires sophisticated algorithms to update the structure and seems to be too complicated for our needs. Instead, we describe how to use Schwarzkopf's method to devise a fully dynamic algorithm with $O(\log^3 n)$ query time. We briefly sketch the technique below.

In our algorithm, when a new site p_{i+1} was inserted, all regions in M_{i+1} that overlapped any destroyed region $R \in \mathcal{M}_i$ were organized into a location structure, namely a sorted list and a monotone subdivision.

To implement Schwarzkopf's method, it is necessary to to design a similar location structure for the case in which R is destroyed by the deletion of a site p_j instead of by the insertion of a new site. Unfortunately, the set of regions overlapping R in the new diagram $\mathcal{M}'$ after the deletion of p_j could be fairly complicated, in fact as complicated as $\mathcal{M}'$ itself. The idea, therefore, is to recursively use the algorithm to design the location structure. More precisely if p_j is deleted, the moving Voronoi diagram of all sites that are neighbors of p_j is computed and all the destroyed regions store pointers to this structure.

Following Schwarzkopf's ideas, we can bound the number of regions in history visited during a location query by $O(\log n)$. Since the transition between two successive regions can be done in $O(\log^2 n)$ time by a call to Theorem 2 one finds that a location query can be performed in $O(\log^3 n)$ time.

The following theorem summarizes the results concerning dynamization:

Theorem 4 *The moving radial Voronoi diagram of a set $\mathcal{S}$ of n moving sites can be maintained in $O(f_{\mathcal{S}}(n)\log n/n)$ expected update time (insertion or deletion) and $O(f_{\mathcal{S}}(n)/n)$ expected space per update. where $f_{\mathcal{S}}(r)$ is the expected size of the moving diagram for a random sample of size r of $\mathcal{S}$. Any fixed query point is located in $O(\log^3 n)$ expected time (all expectations refer only to the random order of insertion).*

The dog and postmen problem can be solved with the same time and complexity if the dog is faster than all postmen.

6 Conclusion

We have presented a randomized dynamic structure for the moving Voronoi diagram able to answer queries in the Voronoi diagram, at any time t, in $O(\log^2 n)$ expected time.

As described, the data structure assumes that the sites are moving with constant velocities. The structure can be extended to handle postman moving with certain other types of speeds, e.g., polynomial motion (where each component of a postman's location vector is a polynomial of fixed degree in t).

In the case of sites moving with constant velocities, the structure can handle more complicated queries, "dog" queries. In this type of query we assume that the customer/dog is also moving, with constant speed which is faster than the speed of all of the postmen. Since the speed of the dog is part of the query this type of query is in fact a 4-dimensional one as opposed to the 3-dimensional moving-Voronoi ones.

If the dog is slower or the same speed as some of the postman, the problem becomes more complicated: a postman moving with velocity v can be viewed as a line in three-dimensional space of slope $\frac{1}{|v|}$ where $|v|$ is the speed of the postman, and a dog can be viewed as the upper half of a cone of slope $\frac{1}{v_d}$. For a fast dog we have $v_d > |v|$ and there is exactly one intersection point between the line and the half-cone. For a slow dog there may be zero or two intersection points meaning that the dog might have two ways to reach the postman running at full speed with one way being quicker than the other.

For these reasons it is still an open problem to solve dog queries efficiently when some of the postmen move faster than the dog. For the same reasons the dog and postmen problem is more difficult to generalize to other kind of motions for postmen, because it is not possible to guarantee that there is only one point of intersection

between the half cone (dog) and the postman trajectory. It is still an open problem to devise satisfactory algorithms for these motions.

The algorithms as originally described were semi-dynamic: postman could be added but not deleted. We also sketched how to utilize a technique originally due to Schwarzkopf [Sch91] to derive a fully dynamic algorithm in the randomized setting that permits $O(\log^3 n)$ expected location time for the queries.

We conclude by noting that the dog diagram can be viewed as a Voronoi diagram of lines using a convex distance function; developing bounds for these kind of diagrams is an open problem in computational geometry.

Acknowledgements

The authors would like to thank M. Goodrich for pointing out the applicability of [GT91] to the moving-Voronoi problem. They would also like to thank Jean-Pierre Merlet for supplying us with his interactive drawing preparation system JPdraw .

References

[AR92] G. Albers and T. Roos. Voronoi diagrams of moving points in higher dimensional spaces. In *Proc. 3rd Scand. Workshop Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 399–409. Springer-Verlag, 1992.

[Aur91] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[BT93] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoret. Comput. Sci.*, 112:339–354, 1993.

[Che85] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Report, Dept. Math. Comput. Sci., Dartmouth College, Hanover, NH, 1985.

[CMS92] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. In *Proc. 9th Sympos. Theoret. Aspects*

Comput. Sci., volume 577 of *Lecture Notes in Computer Science*, pages 463–474. Springer-Verlag, 1992.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[Dev92] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Internat. J. Comput. Geom. Appl.*, 2(1):97–111, 1992.

[DMT92] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Comput. Geom. Theory Appl.*, 2(2):55–80, 1992.

[DTY92] O. Devillers, M. Teillaud, and M. Yvinec. Dynamic location in an arrangement of line segments in the plane. *Algorithms Rev.*, 2(3):89–103, 1992.

[EGS86] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[FL91] J.-J. Fu and R. C. T. Lee. Voronoi diagrams of moving points in the plane. *Internat. J. Comput. Geom. Appl.*, 1(1):23–32, 1991.

[GKS92] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[GMR91] L. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *Proc. 17th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, volume 570 of *Lecture Notes in Computer Science*, pages 113–125. Springer-Verlag, 1991.

[GT91] M. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 523–533, 1991.

[Kle89] R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[MS91] K. Mulmuley and S. Sen. Dynamic point location in arrangements of hyperplanes. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 132–141, 1991.

[Mul91] K. Mulmuley. Randomized multidimensional search trees: dynamic sampling. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 121–131, 1991.

[Pre90] F. P. Preparata. Planar point location revisited. *Internat. J. Found. Comput. Sci.*, 1:71–86, 1990.

[RN91] Thomas Roos and Hartmut Noltemeier. Dynamic Voronoi diagrams in motion planning. In *Computational Geometry — Methods, Algorithms and Applications: Proc. Internat. Workshop Comput. Geom. CG '91*, volume 553 of *Lecture Notes in Computer Science*, pages 227–236. Springer-Verlag, 1991.

[Roo90] T. Roos. Voronoi diagrams over dynamic scenes. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 209–213, 1990.

[Roo91] T. Roos. *Dynamic Voronoi diagrams.* Ph.D. thesis, Bayerische Julius-Maximilians-Univ., Würzburg, Germany, 1991.

[Sch91] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 197–206, 1991.

[Sei91] R. Seidel. Backwards analysis of randomized geometric algorithms. Manuscript, ALCOM Summerschool on efficient algorithms design, Århus, Denmark, 1991.

[Sug92] K. Sugihara. Voronoi diagrams in a river. *Internat. J. Comput. Geom. Appl.*, 2(1):29–48, 1992.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399