Revenge of the Dog: Queries on Voronoi Diagrams of Moving Points

O. Devillers* **M. Golin†** **K. Kedem‡** **S. Schirra§**

Suppose we are given n moving postmen $p_i(t) = q_i + v_i t$, $i = 1, \ldots, n$ where $q_i, v_i \in \mathbb{R}^2$. The problem we address is how to preprocess the postmen so as to be able to efficiently answer two types of nearest neighbor queries. The first type of query asks "who is the nearest postman at time t_0 to a customer located at point q_0?" The second type of query also assumes a query point q_0 at time t_0 but in addition specifies a query speed v_0 with which the query point can move towards a postman. It then asks for the postman that the query point can reach quickest. We provide the first deterministic data structure that permits solving both types of queries.

1 Introduction

We discuss two variations of the classic post-office problem that arise when the post-offices are allowed to move. A recent paper [DG93] introduced the problems and demonstrated a data structure for solving them. The data structure and techniques used there were inherently randomized; the existence of efficient deterministic solutions was posed as an open question. In this paper we provide such solutions. Following [DG93] we assume that each postman moves with constant velocity:

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

where $p_i(t)$ is the location of the i^{th}postman at time t, $q_i \in \mathbb{R}^2$ its location at time 0, and $v_i \in \mathbb{R}^2$ its velocity. By analogy with the static post-office problem we would like to preprocess the postmen so as to easily answer the question "given a query point q_0 at time t_0 who is the closest postman?"

In the static case the meaning of "closest" was quite clear. The closest post-office was also the one that could be reached quickest. In the moving postmen problem we have to distinguish between the two different types of closeness; if the query point q_0 at time t_0 is not moving then we search for the postman whose distance from q_0 at time t_0 is shortest. But when the query point is allowed to chase the postmen in a given speed $v_0 \in \mathbb{R}$, then the postman it can reach first is not necessarily the one closest to it at time t_0.

More formally, we define the types of queries we deal with as follows. Let us denote by $|p - q|$ the Euclidean distance between points p and q in the plane.

(1) Moving-Voronoi queries: Given a customer at location $q_0 \in \mathbb{R}^2$, at time $t_0 \in \mathbb{R}$, find the postman nearest to it. Let

$$M(q_0, t_0) = \{p_i : |p_i(t_0) - q_0| \leq |p_j(t_0) - q_0|, j \neq i, i = 1, \ldots n\}$$

be the set of nearest postmen. The query returns a postman from $M(q_0, t_0)$. (Throughout the paper we abuse notation a little by having p_i denote both the i'th postman and its motion parametrized by t.)

(2) Dog-Bites-Postman queries: The inputs of a query Q are $q_0 \in \mathbb{R}^2$, $t_0 \in \mathbb{R}$, and $v_0 > 0$, $v_0 \in \mathbb{R}$ (only the size of speed is known, direction is chosen by the dog). They specify a dog located at q_0 at time t_0 capable of running at maximum speed v_0. The dog is mean; it wants to catch and bite a postman. It's also 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

$$t_j(Q) = \min\{t \geq t_0 : (t - t_0)v_0 = |p_j(t) - q_0|\},$$

$j = 1, \ldots, n$, to be the first time that the dog can catch postman j, and

$$D(Q) = \{p_i : t_i(Q) \leq t_j(Q), \quad j \neq i, \quad i = 1, \ldots, n\}$$

to be the set of postmen that the dog can reach quickest. The query returns a postman from $D(Q)$.

For the second type of query we assume that $v_0 > |v_i|$, $i = 1, \ldots, n$, i.e., the dog is faster than all of the postmen. This guarantees that every query has an answer and also simplifies the underlying geometry of the problem. We discuss what happens if the dog is slower than the postman in Section 6.

As an example, see Figure 1, suppose that $n = 2$ with $p_1(t) = (2 + t/4, 0)$, $p_2(t) = (-3 + t/2, 0)$. The query point is $q_0 = (0, 0)$, the query time $t_0 = 0$ and the query speed $v_0 = 1$. The nearest neighbor to q_0 at time t_0 is postman p_1. The postman that the dog can reach quickest though is p_2 (this will happen at $t = 2$). The reason

[1]INRIA, B.P.93, F-06902 Sophia-Antipolis cedex, France. email:Olivier.Devillers@sophia.inria.fr. Partially supported by ESPRIT Basic Research Action r. 7141 (ALCOM II).

[2]Hong Kong UST, Kowloon, Hong Kong. email:golin@cs.ust.hk. Partially supported by HK-RGC grant HKUST 181/93E

[3]Ben-Gurion University, Beer-Sheva, Israel. email:klara@math.tau.ac.il. Part of this work was done at Cornell University supported by AFOSR-91-0328.

[4]Max Planck Institut für Informatik, 66123 Saarbrücken, Germany. email:stschirr@mpi-sb.mpg.de. Supported by BMFT (ITS 9103)

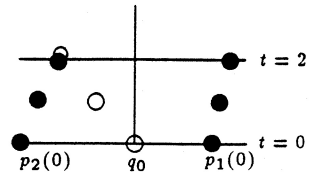

Figure 1: Two types of queries.

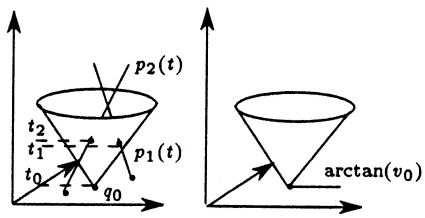

Figure 2: The postmen in (x, y, t) space and the cone of the dog

that the two answers are different is that p_1, the nearest postman, is moving away from the dog while p_2, the further one, is moving towards it.

In the moving-Voronoi query, once t_0 is specified no further information about the postmen's movements is needed. This type of query is equivalent to doing point location in the planar Voronoi diagram $V(t_0)$ of the sites $p_1(t_0), \ldots, p_n(t_0)$.

As illustrated by the example above, for a Dog-type query, we must also know how the points move since it is possible that a postman further away from the dog might be moving towards the dog and can therefore be reached quicker than a nearer postman moving away. One can think of the postmen as straight lines in (x, y, t) space (see Figure 2). A query dog for a query $Q = (q_0, t_0, v_0)$ can be thought of as a circular cone $C(Q)$ with apex (q_0, t_0) that grows upwards with angle $\arctan v_0$. Finding the postman that can be reached quickest is equivalent to finding the postman-line $p(t)$ which intersects $C(Q)$ at the lowest t value. Solving a postman query is therefore equivalent to doing point location in the three dimensional Voronoi diagram of the lines defined above, using the given cone $C(Q)$ as a convex-distance function.

In [DG93] a randomized data structure is described which permits the solution of both moving-Voronoi and fast Dog-type queries. The data structure is constructed incrementally by adding postmen one at a time in a random order. Given this structure both types of queries can be answered in $O(\log^2 n)$ expected time. The expected size of the data-structure is $O(T(n))$ (more precisely it depends on the number of topological changes, $T(n)$, in the moving Voronoi diagram which we discuss below), and the expected preprocessing time is $O(T(n) \log n)$. The development of a deterministic solution was left as an open problem.

In this paper we show two ways to answer these queries deterministically. In the first one we use $O(T(n))$ space and the dog's query time is $O(\log^4 n)$, and in the second we show how to build a deterministic data structure which answers both types of queries in $O(\log n)$ time. The price paid for the latter is an increase in the storage requirement and the preprocessing time, to $O(T(n)^2)$ and $O(T(n)^2 \log n)$, respectively.

The paper is organized as follows. In the next section we discuss our approach to the problem, we review the randomized solution of [DG93] to this problem, and outline an alternative deterministic algorithm based on a technique in [GT91].

In Section 3 we analyze the cell complex $\mathcal{M}$. In Section 4 we show the correspondence between the data structure required for moving-Voronoi queries and the data structure required for Dog-type queries. In Section 5 we show how, using persistent data structures, we answer the two types of queries. We discuss open problems in Section 6.

2 The Generic Approach

We start by considering the Voronoi-diagram of n moving sites (postmen)

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

Consider the three-dimensional space (x, y, t) where the x and y axes span the horizontal plane and the t axis is vertical to the horizontal plane. At any given time $\bar{t}$ the sites $p_i(\bar{t})$ induce a planar Voronoi-diagram, $V_{\bar{t}}$, which partitions the plane $t = \bar{t}$. As we sweep V_t upwards in time, the edges of the swept Voronoi diagram form a partition, $\mathcal{M}$, of 3-space in the following way. The edges of V_t become faces of $\mathcal{M}$, vertices of V_t become edges of $\mathcal{M}$ and Voronoi regions of V_t become three-dimensional cells of $\mathcal{M}$. Thus, $\mathcal{M}$ is a cell complex, defined by the *moving Voronoi diagram* of the moving points. We will sometimes refer to $\mathcal{M}$ as the *Moving-Voronoi-Diagram.*

During the sweep along t the Voronoi diagram V_t undergoes two types of changes. The first type is a *continuous deformation*, where the topology of the diagram remains the same; Voronoi proximity relations do not change. What changes are the locations of the Voronoi vertices in the plane and the locations and lengths of the Voronoi edges. The second type of changes is the addition or deletion of Voronoi edges. This occurs when, due to new proximity relations, a new Voronoi edge appears, first as a new vertex, which then expands (as t grows) to become an edge; or when a Voronoi edge shrinks to a Voronoi vertex and disappears. At these times of appearance and disappearance the topological structure of V_t is modified; these changes are therefore called *topological changes* in V_t. The (x, y, t) points at which these topological changes occur will be the vertices of $\mathcal{M}$. We use $T(n)$ to denote the number of topological changes.

We call the times t at which topological changes occur *critical times.*

The value of $T(n)$ has been well studied; it is known that $T(n) = O(n^3)$ (see, e.g., [FL91], [GMR91]), (this can be shown, e.g., using the linearization method) and that there are sets of n moving points for which $T(n) = \Omega(n^2)$. The problem of whether there are sets of n moving points for which $T(n) = \omega(n^2)$, i.e. asymptotically bigger than n^2, is still open.

Since a two dimensional Voronoi-diagram has space complexity $O(n)$ and each topological change can cause only a constant number of changes to V_t, the space complexity of $\mathcal{M}$ is $O(n + T(n)) = O(T(n))$.

One approach to solving a moving-Voronoi query is to construct a data structure which, given t_0, implicitly reconstructs the Voronoi diagram V_{t_0} and then allows efficient planar point location of q_0 in V_{t_0}. This can be done by slightly modifying a data structure described by Goodrich and Tamassia in [GT91]. This modified structure uses $O(T(n))$ space to store all of the $T(n)$ topologically different Voronoi diagrams and, given a specified diagram, permits planar point location in it using only $O(\log^2(n))$ time. For us the important feature of this structure is that it stores the topological structure of the Voronoi diagrams and not the actual location of their vertices and edges. Given a query (q_0, t_0) one would first use an $O(\log(T(n))) = O(\log n)$ binary search on the topological changes of the Voronoi diagrams to find the topology of the Voronoi diagram at time t_0, and then use Goodrich and Tamassia's structure to locate, in $O(\log^2 n)$ time, the point q_0 within the Voronoi diagram at t_0.

This approach can be used also for dog-queries, increasing the query-time by using the parametric search technique [Me83]. Let Q be a query with location q_0 at time t_0 and speed v_0. The main observation is due to the hypothesis that the dog is faster than the postmen which implies $C(Q) \cap p_i(t) \neq \emptyset$ for all $t \geq t_i(Q)$. Let $t^*(Q)$ be the smallest time where the dog can reach a postman. Given a time t with $M(q_0, t) = p_i$ we can compare $t^*(Q)$ and t as follows: If $t_i(Q) > t$ then the dog cannot reach a postman before t, and $t^*(Q) \in [t, t_i(Q)]$. Otherwise the dog can reach a postman before t, (at least p_i and possibly others) and $t^*(Q) < t_i(Q) < t$. Since $D(Q) = M(q_0, t^*(Q))$, the idea is to run the algorithm above using a query $M(q_0, t^*(Q))$, where $t^*(Q)$ is unknown and to maintain an interval $[t_{inf}, t_{sup}]$ containing $t^*(Q)$. The algorithm by Goodrich and Tamassia begins by a binary search on the critical times. Whenever $t^*(Q)$ is compared to a critical time t, if $t \notin [t_{inf}, t_{sup}]$ then the answer to the comparison is known, otherwise the query $M(q_0, t) = p_i$ is solved and the interval of $t^*(Q)$ is changed, either to $[t_{inf}, t_i(Q)]$ (if $t_i(Q) < t$), or to $[t, \min(t_{sup}, t_i(Q))]$ (if $t_i(Q) > t$), and the binary search continues. After the binary search we know the topology of $V_{t^*(Q)}$. During the location of q_0 in $V_{t^*(Q)}$, we have to compare q_0 against edges of the Voronoi diagram. let $e(t^*(Q))$ be such an edge and let $L(t)$ be the bisector at time t of the postman defining $e(t^*(Q))$. Either (q_0, t) is for all t on the same side of $L(t)$ (wrt the plane parallel to the x, y-plane at height t) or there is exactly one time t_L where (q_0, t_L) is on $L(t_L)$. By solving the query $M(q_0, t_L)$ we can compare t_L and $t^*(Q)$ and thereby decide on which side of $e(t^*(Q))$ the point q_0 resides. Running the parametrized query $M(q_0, t^*(Q))$ performs $O(\log^2 n)$ tests and each such test is answered in $O(\log^2 n)$ time, thus a dog query can be answered in $O(\log^4 n)$ time. Hence we have

Theorem 1 *A moving-Voronoi query for n postmen can be decided in time $O(\log^2 n)$ time using space $O(T(n))$. A dog query for n postmen slower than the dog can be decided in time $O(\log^4 n)$ using space $O(T(n))$.*

Another approach to solving moving-Voronoi queries uses the fact that $\mathcal{M}$ subdivides three-space into cells such that all (q, t) points $(q = (x, y))$ in a given cell have the same nearest postman. Solving a moving-Voronoi query can therefore be done by locating the cell in $\mathcal{M}$ which contains (x, y, t). This approach is used in [DG93], where a three-dimensional point location structure for $\mathcal{M}$ is built incrementally by adding the postmen at a random order, one at a time, to the structure and by saving the changes that the addition of the new postman caused to the old structure. (This method is similar to the Guibas, Knuth and Sharir [GKS92] randomized data structure for point location in static Voronoi diagrams.) It was shown in [DG93] that the expected time for a moving-Voronoi query in this data structure is $O(\log^2 n)$ where the expectation is taken over all possible orders in which the postmen can be inserted into the data structure. It was also shown that, if the dog is faster than all of the postmen, then this same data structure also answers Dog-type queries in $O(\log^2 n)$ expected time. If S is the set of n postmen being stored then the expected size of the data structure was shown to be $O\left(\sum_{r \leq n} \frac{T(r)}{r}\right)$ where $T(r)$ is the expected number of topological changes in the moving-Voronoi diagram of a random sample of r postmen from S. This implies that the expected size of the data structure is $O(n^3)$.

In the following sections we sketch our second deterministic algorithm. First we analyze the geometry of $\mathcal{M}$.

3 The cell complex $\mathcal{M}$

Consider the Voronoi diagram $V(t)$ of the moving sites $p_1(t), \ldots, p_n(t)$. As described above, the cell complex $\mathcal{M}$ is constructed as we sweep the changing diagram $V(t)$ up along t. $\mathcal{M}$ consists of vertices, edges, surface patches and cells; the vertices are points (x, y, t) where there is some topological change in the moving Voronoi diagram, the edges are the vertices of the planar Voronoi diagram which are swept along t, the surface patches

are the edges of the planar Voronoi diagram swept along t, and the cells are the planar Voronoi regions swept along t. Observe now the moving points $p_i(t)$ as line segments in (x, y, t) space; the cells of $\mathcal{M}$ can be viewed as *sleeves* around these line segments. The boundaries of the sleeves consist of algebraic surface patches (ruled surfaces), which in turn intersect in algebraic curves, called edges, and the edges intersect in the vertices of the cell complex $\mathcal{M}$.

More explicitly, let $p_i(t) = q_i + v_i t$ for $i = 1, \ldots, n$, where each point $q_i = (x_i, y_i)$, and $v_i = (v_{xi}, v_{yi})$. Then the surface between two moving points $p_i(t)$ and $p_j(t)$ is described by

$$(x - x_i - v_{xi}t)^2 + (y - y_i - v_{yi}t)^2 = (x - x_j - v_{xj}t)^2 + (y - y_j - v_{yj}t)^2,$$

which is a quadratic algebraic surface. The edges, which are intersections of these surfaces, can be quartic curves in (x, y, t).

Clearly there are exactly $O(n)$ sleeves in $\mathcal{M}$. As to the total complexity of $\mathcal{M}$. We have said that the number of vertices in $\mathcal{M}$ is $O(T(n))$, it is well known that the number of changes to the Voronoi diagram at a topological change is bounded by a constant, therefore the total complexity of $\mathcal{M}$ (vertices, edges, surface patches) is $O(T(n))$.

Now consider $l_{x,y}$, the fixed infinite vertical line perpendicular to the horizontal plane at point (x, y) and its intersections with the surfaces of the cell complex $\mathcal{M}$. These intersections subdivide $l_{x,y}$ into intervals in which a particular postmen is always the nearest. Label the interval with the name of the corresponding nearest postman. The labels change at the times that $l_{x,y}$ intersects $\mathcal{M}$. The enumeration of the different labelings along $l_{x,y}$ as t changes from $t = -\infty$ to $t = \infty$ defines the *labeling* of $l_{x,y}$. The (infinite) set of all vertical lines is divided into equivalence classes, each class containing all the lines with equal labeling.

The number of different labelings of these lines can be bounded by the number of faces, edges and vertices of the projection of $\mathcal{M}$ on the (x, y) plane. To obtain the projection of $\mathcal{M}$ on the plane we project the edges and vertices of $\mathcal{M}$ on the plane. Projecting surfaces on the plane is the projection of all points on the surface patch which are tangent to a vertical line (the *silhouette* of the surface patch). It is known that the silhouette of an algebraic surface patch of a constant degree consists of a constant number of connected components (each of which is also algebraic of constant degree), and that it has a constant number of extremal points in a given direction and points of self intersection.

Thus, the total number of edges on the projection of $\mathcal{M}$ on the plane (x, y) is $O(T(n))$. Since every edge (or surface) is an algebraic of constant degree, so is its projection on the plane. Hence two edges on the plane intersect at most a constant number of times. Given that there can be $O(\binom{T(n)}{2})$ intersecting edges, we get that the number of equivalence classes (cells on the plane) is $O(T(n)^2)$.

4 The correspondence between the two types of queries

Consider a customer sitting at some fixed point $q = (x, y)$ forever observing the postmen whizzing by. As time passes, the nearest postman or postmen to the customer will only change at certain discrete times, $t_1 < t_2 < \cdots < t_m$; between two discrete times the nearest postman (postmen) will remain the same. We set i_j to be the index of a nearest postman to q between times t_j, and t_{j+1}, $j = 1, \ldots, m$. To make our definitions consistent we set $t_0 = -\infty$ and $t_{m+1} = \infty$. Formally, the t_j and i_j are defined so that for all $t \in [t_j, t_{j+1})$ we have $p_{i_j} \in M(x, y, t)$ and for small enough ϵ we have that $M(x, y, t_j - \epsilon) \neq M(x, y, t_j + \epsilon)$.

To visualize the situation geometrically consider a vertical line starting at $(q, -\infty)$ and traveling upwards to (q, ∞). The time t_j is the j'th time that the line passes through a face of $\mathcal{M}$. The face that the line stabs at time t_j is the one that separates the region associated with postman $p_{i_{j-1}}$ from the one associated with postman p_{i_j}. We call the times t_j, the *stabbing times* and the sequence i_j, $j = 1, \ldots, m$, the *stabbing sequence* associated with q.

Because the postmen are moving linearly, m, the size of the stabbing sequence must be small.

Lemma 1 *Fix a point q and let the stabbing sequence $i_1, \ldots, i_m$ be defined as above. This sequence is a $(n, 2)$ Davenport-Schinzel sequence and hence $m \leq 2n$.*

Proof.
A sequence is a $(n, 2)$ Davenport-Schinzel sequence if it does not contain a 2-repeating subsequence of the form

$$i \ldots j \ldots i \ldots j.$$

Suppose the stabbing sequence did contain some 2-repeating subsequence. If $i \ldots j$ or $j \ldots i$ then there is some time t between the time when p_i is the nearest postman to q and the time when p_j is the nearest postman such that

$$|p_i(t) - q|^2 = |p_j(t) - q|^2.$$

The existence of a 2-repeating sequence therefore implies the existence of at least three distinct times t when this equation is satisfied. The points move with constant speed though so $|p_i(t) - q|^2 - |p_j(t) - q|^2$ is a quadratic equation and only has two roots leading to a contradiction. By contradiction the stabbing sequence is a $(n, 2)$ Davenport-Schinzel sequence and has hence length $m \leq 2n$. □

We can now propose a different approach to answering a moving-Voronoi query. Note that between any two

stabbing times t_j, t_{j+1} the vertical line through q will be totally contained within the region associated with postman p_{i_j}.

Suppose that, given any point q we can access the stabbing times associated with q in a way that permits binary search on t. Then we can solve a moving-Voronoi query (q,t) as follows: in $O(\log m) = O(\log n)$ time perform a binary search on the stabbing times to find the interval $I_j = [t_j, t_{j+1})$ that contains t. Then the nearest postman to q at time t is postman p_{i_j}.

The surprising fact is that if we can access the stabbing sequence in this way then we can also answer dog queries in $O(\log n)$. This will follow from the next lemma which is a consequence of Theorem 4 in [DG93].

Lemma 2 *Let q be a fixed point in $\mathbb{R}^2$. Let v be a fixed dog speed faster than those of all of the postmen; $v > |v_i|$, $i = 1, \ldots, n$. Given a time t we define a function $\rho(t)$ as follows. Let $p \in M(q,t)$ be a postman nearest to q at time t. Set $d(t) = |p(t) - q|$ to be the distance between q and its nearest neighbor. Define the function $\rho_v : \mathbb{R} \to \mathbb{R}$*

$$\rho_v(t) = t - \frac{d(t)}{v}.$$

Then
(a) ρ_v is a 1-1 continuous mapping from $\mathbb{R}$ to $\mathbb{R}$ such that if $t > t'$ then $\rho_v(t) > \rho_v(t')$. Furthermore $\rho_v(-\infty) = -\infty$ and $\rho_v(\infty) = \infty$.
(b) $M(q,t) = D(q, \rho_v(t), v)$.

Statement (a) tells us that ρ_v maps the interval $I_j = [t_j, t_{j+1})$ continuously into the interval $\rho_v(I_j) = [\rho_v(t_j), \rho_v(t_{j+1}))$ and that

$$\rho_v(t_1) < \rho_v(t_2) < \cdots < \rho_v(t_m).$$

Because the intervals I_j, $j = 1, \ldots, m$, partition $\mathbb{R}$ the intervals $\rho_v(I_j)$, $j = 1, \ldots, m$ also partition $\mathbb{R}$. Statement (b) says that if p is a nearest neighbor to q at time t then p is a postman that a dog starting at q at time $\rho_v(t)$ can reach quickest if the dog travels with speed v. Taken together the two statements provide us with a way of answering a Dog-type query. Given a dog located at q at time t' that can travel with speed v we locate the unique interval I_j such that $t' \in \rho_v(I_j)$. We then know that postman p_{i_j} is the one that the dog can reach quickest.

There remains one difficulty. How do we find the interval I_j such that $t' \in \rho_v(I_j)$. We can do this by performing a binary search on the m values

$$\rho_v(t_1) < \rho_v(t_2) < \cdots < \rho_v(t_m).$$

Although we do not know these values in advance the values

$$t_1 < t_2 < \cdots < t_m$$

are available in a fashion that permits us to do binary search on them. Given any t_j we also know the postman p_{i_j} which is a nearest postman to q at time t_j. We can therefore, in constant time, compute $d(t) = |p_{i_j}(t_j) - q|$ and from there $\rho_v(t_j)$. Consequentially, given any t_j we can, in constant time, decide whether $t' > \rho_v(t_j)$ or not. We can therefore perform an $O(\log n)$ binary search to find the interval $\rho_v(I_j)$ which contains t'.

To review, we have just seen that if we have a data structure which returns the stabbing times $\{t_j\}$, in a form suitable for binary search, for any given point q, then we can solve both moving-Voronoi queries and dog queries in $O(\log n)$ time. In the next section we will describe such a data structure.

5 Point location using persistent binary search trees

Assume we have constructed $\mathcal{M}$ by one of the standard methods, see, e.g., [GMR91]. Let $\mathcal{M}^\perp$ denote the projection of $\mathcal{M}$ onto the (x,y) plane. We construct the planar subdivision defined by $\mathcal{M}^\perp$ by a plane sweep. The sweep stops at intersections and cusps of the projection of the edges and the silhouettes of the facets of $\mathcal{M}$. Under the assumption that intersections and cusps of the curve segments can be computed in constant time, the sweep takes time $O(r \log r) = O(r \log n)$ where r is the number of regions of the subdivision. Since two algebraic curves of bounded degree have at most a constant number of intersections, we have $r = O(T(n)^2)$.

During the sweep we can build a point location structure according to Sarnak and Tarjan [ST86] and Cole [Co86]. The sweep defines a subdivision of the plane into vertical slabs such that curve segments in each slab are totally ordered. Since the order of the curves is similar for contiguous slabs, the curve segments are stored in versions of a partially persistent binary search tree. For locating a point (x,y) we first determine the slab containing x and then locate y between the curve segments intersecting the slab using the version of the partially persistent binary search tree that corresponds to the slab. All versions together have space complexity $O(r) = O(T(n)^2)$, point location takes time $O(\log r) = O(\log n)$ [ST86, DSST89].

After locating the region that contains the projection (x,y) of the query point (x,y,t), we have to locate t in the stabbing sequence corresponding to this region. Since the stabbing sequences of neighboring regions are similar we can use persistent search trees again. However, we don't have a natural linear order on the versions of the binary search trees, so partially persistence is not sufficient here. Hence we use fully persistent binary search trees [DSST89] to store the stabbing sequences. Given a search tree for a region, a constant number of updates is sufficient to build a search tree for a neighboring region. Any rooted spanning tree of the dual of the graph defined by the planar subdivision gives us a rooted version tree (a partial order on the versions).

Starting with the empty version we build the version of the fully persistent binary search tree corresponding to the root region of our spanning tree. For the other regions we update the version corresponding to the region which precedes it in the spanning tree. Since $O(1)$ updates suffice the fully persistent search tree for a region can be constructed in time $O(\log n)$ with $O(1)$ additional storage, cf. [DSST89]. With a region we store a pointer to the search tree for its stabbing sequence. We get construction time $O(n + r \log n)$ and $O(n + r)$ space, where r is the number of regions. Once a region is known we can locate t with the fully persistent binary search tree in time $O(\log n)$. Altogether we get

Theorem 2 *A moving-Voronoi query for n postmen can be decided in time $O(\log n)$ time using space $O(T(n)^2)$. A dog query for n postmen slower than the dog can be decided in time $O(\log n)$ using space $O(T(n)^2)$.*

6 Open problems

The major problem left open in this paper is how to solve Dog-type queries if the dog is slower than some of the postmen. If the dog is slower than the postmen then the correspondence between moving-Voronoi and Dog-type queries described in Section 4 no longer holds and it is not obvious how to construct a data structure that permits the solution of both types of queries.

It will be good to introduce a trade off between query time and storage requirement for this problem. In Section 2 we use an optimal $O(n^3)$ space, but the Dog-type query is answered in time $O(\log^4 n)$, while in Section 5 we achieve a logarithmic time for answering a query, while the space needed is quite large – $O(n^6)$.

7 Acknowledgements

Klara Kedem and Mordecai Golin would like to thank the Max-Planck-Institut, for providing an excellent working environment.

The authors would like to thank Mike Goodrich for directing our attention to the applicability of [GT91] to solving moving-Voronoi queries.

References

[Co86] R. Cole. Searching and storing similar lists. *J. Algorithms*, 7:202–220, 1986.

[DG93] O. Devillers and M.J. Golin. Dog Bites Postman: Point Location in the Moving Voronoi Diagram and Related Problems. In *Proceedings of the First European Symposium on Algorithms.* 1993

[DSST89] J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38:86–124, 1989.

[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.

[Me83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms In *Journal of the ACM*, volume 30, pages 852–865, 1983.

[ST86] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.