ELSEVIER

Computational Geometry 6 (1996) 315–327

Computational Geometry
Theory and Applications

Queries on Voronoi diagrams of moving points

O. Devillers[a,*], M. Golin[b,1], K. Kedem[c,2], S. Schirra[d,3]

[a] *INRIA, B.P.93, F-06902 Sophia-Antipolis cedex, France*
[b] *Hong Kong UST, Kowloon, Hong Kong*
[c] *Ben-Gurion University, Beer-Sheva, Israel*
[d] *Max Planck Institut für Informatik, 66123 Saarbrücken, Germany*

Communicated by Mark Keil; accepted 7 November 1995

Abstract

Suppose we are given n moving postmen described by their motion equations $p_i(t) = s_i + v_i t$, $i = 1, \ldots, n$, where $s_i \in \mathbb{R}^2$ is the position of the ith postman at time $t = 0$, and $v_i \in \mathbb{R}^2$ is his velocity. The problem we address is how to preprocess the postmen data so as to be able to efficiently answer two types of nearest-neighbor queries. The first one asks "who is the nearest postman at time t_q to a dog located at point s_q. In the second type a query dog is located at point s_q at time t_q, its speed is $v_q > |v_i|$ (for all $i = 1, \ldots, n$), and we want to know which postman the dog can catch first. The first type of query is relatively simple to address, the second type at first seems much more complicated. We show that the problems are very closely related, with efficient solutions to the first type of query leading to efficient solutions to the second. We then present two solutions to these problems, with tradeoff between preprocessing time and query time. Both solutions use deterministic data structures.

Keywords: Dynamic computational geometry; Voronoi diagram; Post-office problem; Parametric search; Persistent data structures

1. Introduction

The classic *post-office* problem is to preprocess the locations of n post-offices in the plane so as to permit efficient solutions to queries of the type "where is the closest post-office to a customer located at (x, y)". The standard solution to this problem is to preprocess the post-offices by constructing their *Voronoi-diagram*; a query is answered by performing a planar point location in the Voronoi-diagram.

* Corresponding author. E-mail: Olivier.Devillers@sophia.inria.fr. Partially supported by ESPRIT Basic Research Action r. 7141 (ALCOM II).

[1] E-mail: golin@cs.ust.hk. Partially supported by HK-RGC grant HKUST 181/93E.

[2] E-mail: klara@ivory.bgu.ac.il. Supported by a grant from the U.S.-Israeli Binational Science Foundation.

[3] E-mail: stschirr@mpi-sb.mpg.de. Supported by BMFT (ITS 9103).

We discuss the variant of the post-office problem that arises when the post-offices become postmen, i.e., they are allowed to move. A recent paper [4] introduced the problem and demonstrated a data structure for solving it. 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 [4] we assume that the motion of each postman is described by the equation

$$p_i(t) = s_i + v_i t, \quad i = 1, \ldots, n,$$

where t stands for time, s_i is the location in the plane of the ith postman at time $t = 0$, and $v_i \in \mathbb{R}^2$ is his velocity vector. Thus $p_i(t)$ is the location of the ith postman at time t. 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 s_q at time t_q who is the postman *closest* to s_q?" In the static case the meaning of "closest" was clearly closest in terms of distance. When the postmen are moving we must distinguish between two very different problems: the closest postman at a given time (see query (1) below) and the postman that can be reached first (see query (2) below).

More formally, denoting the Euclidean distance between points $p, s \in \mathbb{R}^2$ by $|p - s|$, we define the two types of queries:

(1) *Moving-Voronoi* query: Given a point (dog) query, q, by its location $s_q \in \mathbb{R}^2$, at time t_q, find the postman nearest to it. Let

$$M(s_q, t_q) = \{p_i\colon\ |p_i(t_q) - s_q| \leqslant |p_j(t_q) - s_q|,\ \forall j = 1, \ldots, n\}$$

be the set of nearest postmen to s_q at time t_q. The query returns a postman from $M(s_q, t_q)$. (Throughout the paper we abuse notation slightly by having p_i denote both the ith postman and its motion parametrized by t.)

(2) *Dog-Bites-Postman* query: We define the query, q, to be a triple (s_q, t_q, v_q) where $s_q \in \mathbb{R}^2$ is the initial location of the dog at time t_q, and $v_q > 0$, $v_q \in \mathbb{R}$, is the dog's speed. The problem is to find the postman that the dog can reach quickest. For the dog, only the magnitude of its speed is known; the direction of its velocity is chosen by the dog to minimize the time before reaching a postman. For $j = 1, \ldots, n$, set

$$t_j = \min\{t \geqslant t_q\colon\ (t - t_q)v_q = |p_j(t) - s_q|\},$$

to be the earliest time that the dog can catch postman j if it starts running at time t_q, and

$$D(s_q, t_q, v_q) = \{p_i\colon\ t_i \leqslant t_j,\ \forall j = 1, \ldots, n\}$$

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

As in [4] we assume that $v_q > |v_i|$, for all $i = 1, \ldots, n$, 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 briefly discuss what happens if the dog is slower than the postman in Section 5.

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

At first glance Moving-Voronoi queries look much easier to solve than Dog-Bites-Postmen ones. This is because the first type of query can be thought of as freezing time at $t = t_q$; if the positions

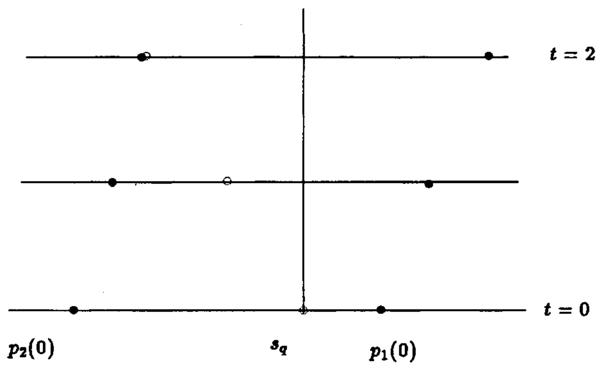

Fig. 1. Two types of queries.

of the postmen at time t_q are all known the problem can be solved. The second problem looks much more difficult because any solution not only needs information about where the postmen are at time $t = t_q$ but also needs further information about where they are going to be in the future. One of the main tools developed in this paper is a geometric method of reducing Dog queries to a sequence of Moving-Voronoi queries.

In this paper we describe two deterministic techniques for solving these queries, with tradeoff between preprocessing time and query time. For each technique we construct a data structure which permits a fast solution of both types of queries. In essence both solutions reduce to the problem of answering several point location queries in arrangements in (x, y, t)-space. For both techniques the solution we propose for solving Moving-Voronoi queries is a straightforward cookbook application of known tools. The complications arise in modifying these solutions to solve Dog queries as well.

The first solution starts by locating t_q in some data structure and next locates $s_q = (x_q, y_q)$ in the plane parallel to the xy-plane at $t = t_q$. The Dog-Bites-Postman query time is $O(\log^4 n)$ and its space complexity is $O(T(n) \log n)$, where $T(n)$ is the number of topological changes in the Voronoi-diagram of linearly moving points (cf. [6,8]). $T(n)$ is described in more detail in the next section. For Moving-Voronoi queries the query time is $O(\log^2 n)$ with the same space complexity as above.

The second solution first locates (x_q, y_q), and then locates t_q in a data structure on a line parallel to the t-axis through (x_q, y_q). It has query time $O(\log n)$ for both type of queries but space complexity $O((T(n)^2)$.

The time and space requirements for both solutions as well as the query times they support are presented in Fig. 2.

The paper is organized as follows. In the next section we discuss the geometric structure of the problem. In Section 3 we briefly review some previous related work. Our first deterministic approach

Method	Space	Preprocessing	M.V. Query	Dog Query
1	$O(T(n)\log n)$	$O(T(n)\log n)$	$O(\log^2 n)$	$O(\log^4 n)$
2	$O(T(n)^2)$	$O(T(n)^2\log n)$	$O(\log n)$	$O(\log n)$

Fig. 2. The algorithms presented in this paper for solving moving-Voronoi and dog queries. $T(n)$ is the number of topological changes in the Voronoi diagram of n moving points.

is given in Subsection 4.1. Our second, time-optimal solution, is presented in Subsection 4.2. We conclude and discuss open problems in Section 5.

2. The geometric structure of the problem

We start by considering the Voronoi-diagram of n moving points

$$p_i(t) = s_i + v_i t, \quad i = 1, \ldots, n.$$

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 this plane. At any given time t_0 the set of points (*sites*) $p_i(t_0)$, $i = 1, \ldots, n$, define a *planar* Voronoi-diagram, $V(t_0)$, which partitions the plane $t = t_0$. As the points move with t, their corresponding planar Voronoi diagram, $V(t)$, changes continuously and sweeps the 3-dimensional space (x, y, t). The sweep creates a partition, $\mathcal{M}$, of this space in the following way. The vertices of $V(t)$ sweep along edges of $\mathcal{M}$, edges of $V(t)$ sweep faces of $\mathcal{M}$ and Voronoi regions of $V(t)$ sweep three-dimensional cells of $\mathcal{M}$. Thus, $\mathcal{M}$ is a cell complex, which we call the *moving Voronoi diagram* of the moving points.

During the sweep along t the Voronoi diagram $V(t)$ may undergo two types of changes. The first type is a *continuous deformation*, in which the topology of the Voronoi diagram remains the same; in this type of change, Voronoi proximity relations do not change so no Voronoi edges and/or vertices are either created or deleted. Only the location of the Voronoi vertices in the plane and the location and lengths of the Voronoi edges change. The second type of change is the addition and deletion of Voronoi edges. A Voronoi vertex is the center of an empty circle containing three sites on its boundary. A Voronoi edge can disappear only if its two Voronoi vertex endpoints merge. This in turn occurs only when all of the sites defining the two circles—there are four of them—become co-circular. See [8] for more details. Due to new proximity relations, an old Voronoi edge contracts to a vertex (effectively merging its two endpoints) and then expands to become a new Voronoi edge. When this type of change occurs the topological structure of $V(t)$ is modified and these changes are therefore called *topological changes* in $V(t)$. Note that when this type of change occurs the Voronoi vertex defined by the four sites at time t creates a vertex in $\mathcal{M}$.

In order to estimate the complexity of $\mathcal{M}$ we first need a bound on the number of vertices in $\mathcal{M}$, which is also the number of topological changes in the moving Voronoi diagram. We denote this number by $T(n)$. The value of $T(n)$ has been extensively studied; it is known that $T(n) = O(n^3)$ [6,8], 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 $\mathcal{M}$, as measured by the total number of its cells, edges, faces and vertices, is $O(n + T(n)) = O(T(n))$.

In [8], Guibas, Mitchell and Roos describe an algorithm that in $O(T(n) \log n)$ time starts at $t = -\infty$ and sweeps towards $t = \infty$, stopping at each topological change in the Voronoi-diagram and reporting it. Suppose then that their algorithm has been run and the times $\tau_1 < \tau_2 < \cdots < \tau_k$, $k \leqslant T(n)$, at which the topological changes occur have been found.

Another way to view the cell complex $\mathcal{M}$ is to describe the motion of the points as line segments in 3-space, thus 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 $p_i(t) = s_i + v_i t$ for $i = 1, \ldots, n$, where each point $s_i = (x_i, y_i)$ 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 n sleeves in $\mathcal{M}$, one for each point.

3. Previous work

The combinatorics of moving-Voronoi queries have already been addressed in [2,8,11–13]; these papers actually treat the evolution of *changes* in the Delaunay triangulation and the Voronoi diagram and not point location in them. A special case of dog type queries—one in which all of the postmen move with the same velocity—was dealt with in [15], in that simple case, the diagram has linear size and is computed in $O(n \log n)$ time. The general dog type query and algorithms for both types of queries were introduced in [4].

The approach to solving Moving-Voronoi queries followed in [4] uses the fact that $\mathcal{M}$ subdivides three-space into cells such that all points 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_q, y_q, t_q). In [4], a three-dimensional point location structure for $\mathcal{M}$ is built incrementally by adding the postmen in a random order, one at a time, into the structure and by saving the changes that the addition of the new postman caused to the old structure. (This method is an extension of the Guibas, Knuth and Sharir [9] randomized data structure for point location in static Voronoi diagrams.) It was shown in [4] 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 P is the set of n postmen being stored then the expected size of the data structure was shown to be

$$O\Biggl(\sum_{r \leqslant n} \frac{T^E(r)}{r}\Biggr)$$

where $T^E(r)$ is the expected number of topological changes in the moving-Voronoi diagram of a random sample of r postmen from P. This implies that the expected size of the data structure is $O(n^3)$.

4. Our solutions

We now describe two approaches that will each allow us to solve both types of queries deterministically. One approach is more economical in space requirements than the other approach, while having greater query times. In both approaches we first solve the Moving-Voronoi query, which is actually a point location problem in a Voronoi diagram of moving points. Based on the point location solution we build an algorithm for the Dog-Bites-Postman query.

4.1. Space-efficient solution

One approach to solving a Moving-Voronoi query would be to store the topology of the graph of each Voronoi-diagram between two consecutive topological changes in a way that permits point location. Recall that we denoted by $\tau_1, \ldots, \tau_k$ the times at which the topological changes occured in the moving Voronoi diagram. We denote by V_i the topological structure of the Voronoi diagram of the postmen in time interval $[\tau_i, \tau_{i+1})$. (In this structure a Voronoi edge, e.g., is stored as the sites that are equidistant from it, and a Voronoi vertex as a triple of the sites equidistant from it, together with the cyclic ordering of the edges incident to it.) An obvious improvement to this approach takes advantage of the fact that two consecutive Voronoi diagrams have one topological change between them, which, as described above, causes just a constant number of local changes to the edges and vertices of these Voronoi diagrams. So a data structure for dynamic planar point location that uses only the topology and can be made partially persistent would be very useful.

We are not aware of an efficient planar point location structure that uses only the topology of the planar map. However, Goodrich and Tamassia [7] present a method for dynamic planar point location and a dynamic data structure which maintains a dynamically changing monotone subdivision, its graph theoretic dual and spanning trees for both, which nearly uses only the topology. (A *monotone subdivision* with respect to the y-axis is a planar graph in which each face has the property that its boundary is intersected at most twice by any horizontal line.) This point location structure can be used for $V(t')$ and $V(t'')$ as long as both have the same topology and the directed graphs, obtained by directing all edges of both Voronoi diagrams downwards, with respect to the y-axis, are isomorphic (cf. Lemma 4.2 in [7]). Clearly, a planar graph in which each face is convex, such as the Voronoi diagram, is a monotone subdivision. The Voronoi diagram V_i is kept in a topological representation. In order to maintain a y-monotone representation of the moving Voronoi diagram, we notice that the directed graph associated with the Voronoi diagram $V(t)$, changes only when, at time t', a Voronoi edge in $V(t')$ becomes parallel to the x-axis (horizontal). Since all the faces of the moving Voronoi diagram are ruled surfaces of constant degree, each edge can become horizontal at most a constant number of times, so, in total, the number of changes in the monotone representation of the Voronoi diagrams is also $O(T(n))$. We refine each interval $[\tau_i, \tau_{i+1})$ into a sequence of sub-intervals, such that in each sub-interval the y-monotone representation corresponding to the Voronoi diagram, does not change. Let the set $\mathcal{T} = \{\tau'_1, \ldots, \tau'_l\}$, $l = O(T(n))$, denote now the thus refined set of times.

Since the successive Voronoi diagrams are monotone subdivisions, a persistent data structure can be used for point location [5,7]. Given the list of times and the corresponding changes at these times in the monotone subdivisions, the data structure can be constructed in time $O(T(n)\log n)$, and its size is $O(T(n)\log n)$. Applying this to the Moving-Voronoi query $q = (s_q, t_q)$, we first perform an $O(\log n)$ binary search on the set $\mathcal{T}$ to find i such that $t_q \in [\tau'_i, \tau'_{i+1})$, and then follow by performing the point location algorithm of Goodrich and Tamassia [7] in time $O(\log^2 n)$. This brings the total time for solving a Moving-Voronoi query to $O(\log^2 n)$.

Turning to the Dog-Bites-Postman queries, we will now describe how access to moving-Voronoi solutions and an approach very similar to parametric searching (cf. [10]) will permit us to solve dog

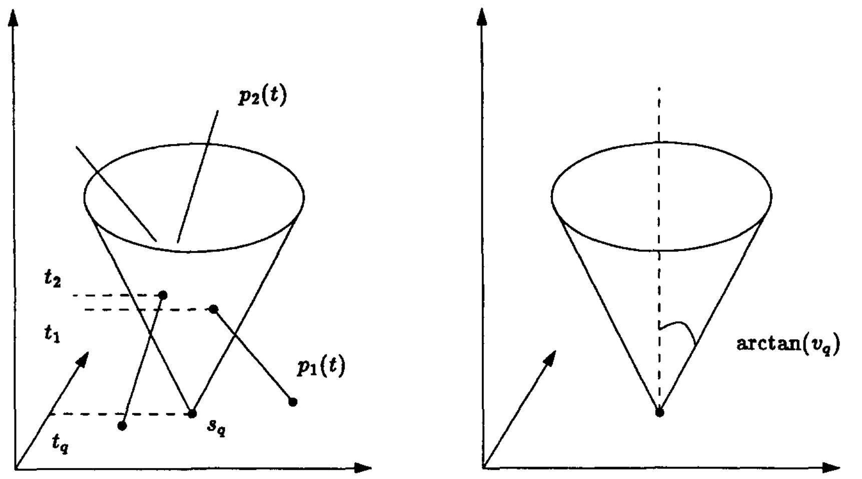

Fig. 3. The postmen in (x, y, t) space and the cone C_q of the dog's motion options.

queries. Recall that a dog query q is specified by the dog's starting location s_q at starting time t_q, and its speed v_q, and that t_i is the first time that a dog can reach postman p_i. Let $t^* = \min_i t_i$ be the first time that a dog can reach some postman. The crucial observation is the following lemma.

Lemma 1. *Let $t > t_q$ be an arbitrary time and let $p_i(t)$ be the nearest postman to s_q at time t. If the dog is faster than all of the postmen then*

$$t^* \leqslant t \quad \textit{if and only if} \quad t_i \leqslant t.$$

Proof. Since $t^* \leqslant t_i$ the *if* direction is obvious.

To prove *only if* we introduce a geometric construct associated with the Dog-Bites-Postman query. We can view the motion of the postmen as straight line segments in (x, y, t)-space (see Fig. 3). A query dog at (s_q, t_q) with speed v_q can choose to run in any one direction, which corresponds to choosing a generating line on the boundary of a circular cone C_q in (x, y, t)-space, with an apex at (s_q, t_q), that grows upwards with angle $\arctan v_q$. The motion (direction) chosen by the dog is, therefore, a ray from the apex of C_q on the boundary of it. Finding the postman that can be reached quickest is equivalent to finding the line segment of postman p_j which intersects the cone C_q at the lowest t value. Denote by $C_q(t)$ the circle which is the 'horizontal' cross section of C_q at time $t \geqslant t_q$. Clearly the radius of $C_q(t)$ is $v_q(t - t_q)$ for $t \geqslant t_q$.

Assume by contradiction that $t^* \leqslant t < t_i$. Then there must be some postman p_j such that $t^* = t_j \leqslant t$. Since the dog is faster than all postmen, thus faster than postman p_j, then, once the line of postman p_j in (x, y, t)-space enters the cone C_q, at time t_j, it will never leave the cone again, i.e., $C_q(t') \cap p_j(t') \neq \emptyset$ for all $t' \geqslant t_j$. This implies that $C_q(t) \cap p_j(t) \neq \emptyset$ because it entered the cone at time $t_j \leqslant t$. On the other hand, the point $p_i(t)$ must be outside of circle $C_q(t)$ because $t < t_i$, and t_i is the first time the line segment of postman p_i entered the cone. The radius of $C_q(t)$ is $v_q(t - t_q)$, so

$$|p_i(t) - s_q| > v_q(t - t_q) \geqslant |p_j(t) - s_q|,$$

contradicting the fact that $p_i(t)$ was the nearest postman to s_q at time t. □

Let p_i be a nearest postman to s_q at time t^*. The lemma implies that $t_i \leqslant t^*$. By definition $t^* \leqslant t_i$ so $t^* = t_i$ and p_i is a postman that the dog can catch quickest. This suggests an algorithm for solving Dog-Bites-Postman queries; perform a Moving-Voronoi query at location s_q at time t^*. The difficulty with this approach is that knowing t^* requires having already solved the dog query. We work around this difficulty by using parametric search [10] and applying Lemma 1.

The algorithm for the Dog-Bites-Postman query has two phases. The first phase performs a modified binary search on τ to find the interval $[\tau'_i, \tau'_{i+1})$ such that $t^* \in [\tau'_i, \tau'_{i+1})$, and the associated point location data structure for the topology of $V(t^*)$. The second phase uses a variation of parametric search [10] to find t^* in this interval. It runs the point location procedure implicitly for t^*, and on any branching point of the procedure it makes a local decision on the branching options, as we describe below, while, at the same time, it truncates the time interval where t^* can be found.

For the first phase note that questions of the form "is $t^* \leqslant t$?" can be answered by performing a Moving-Voronoi query at time t, taking the answer p_i and checking whether $t_i \leqslant t$. If the answer is "yes" then the lemma implies $t^* \leqslant t$, otherwise it implies $t^* > t$. Using binary search on τ, we can find, by asking $\log(T(n)) = O(\log n)$ such queries, the interval $[\tau'_i, \tau'_{i+1})$ such that $t^* \in [\tau'_i, \tau'_{i+1})$. The total amount of time for the binary search is the number of Moving-Voronoi queries made multiplied by the amount of time required for answering a Moving-Voronoi query, i.e., $O(\log n) * O(\log^2 n) = O(\log^3 n)$.

Along with the interval $[\tau'_i, \tau'_{i+1})$ we also find the associated point location structure for searching in the Voronoi diagram $V(t)$, $t \in [\tau'_i, \tau'_{i+1})$. How can this be used to search in $V(t^*)$? Consider the algorithm of [7] for simple point location at a fixed time t in the interval $[\tau'_i, \tau'_{i+1})$. It asks $O(\log^2 n)$ questions of the form "is $s_q = (x_q, y_q)$ above or below bisector line $L(t)$ at time t?", where $L(t)$ is the line through an edge of $V(t)$, and of the form "is y_q greater or smaller than the y-coordinate of Voronoi vertex $v(t)$ at time t?"

Even though we do not know the exact value of t^* we will be able to use Lemma 1 to *parametrically* answer questions of the two types. This will enable us to make the proper branching choices in the point location procedure and find the region that contains s_q at time t^* and its associated nearest neighbor postman, as we describe below.

The lines $L(t)$ to which s_q is compared in the procedure, are extensions of edges of $V(t)$, and are therefore the bisectors of two postmen. Suppose then that line $L(t)$ is the bisector of postmen p' and p''. The crucial observation here is that s_q lies on $L(t)$ only when $|p''(t) - s_q|^2 - |p'(t) - s_q|^2 = 0$; since the points move with linear motion this is a quadratic equation in t so, if the equation is not identically 0—corresponding to s_q always lying on the bisector—then s_q may lie on $L(t)$ at most twice. Thus, s_q switches from being above or below $L(t)$ to below or above $L(t)$ at most twice.

Answering the question "is s_q above or below line $L(t)$ at time t^*?" is therefore a matter of calculating the times that s_q lies on $L(t)$. Suppose these are times t' and t'', and assume $t' \leqslant t''$. If both t' and t'' are outside time interval $[\tau'_i, \tau'_{i+1})$ then s_q is either always above or always below line $L(t)$ for all times in the interval and specifically for time t^*. We calculate which it is, above or below, and then proceed with the search. If either t' or t'' or both are in this interval we perform at most two calls (one for each of these times) to the Moving-Voronoi query procedure to calculate if t^* is less than t', between t' and t'' or greater than t'', using Lemma 1. As before, locating s_q in $V(t')$ gives the postman p_j closest to s_q at time t', and checking if $t_j \leqslant t'$ will tell us, using Lemma 1, if $t^* \leqslant t'$. Similarly for t''. The answers to these two questions allow us to perform the correct branching, as well as to truncate the time interval where t^* can be found.

Similarly, the Voronoi vertex $v(t)$, to which s_q is compared in the second type of question, is the center of the circumcircle through three specified linearly moving postmen and therefore follows an algebraic curve of constant degree and can only pass through the horizontal line $y = y_q$ a constant number of times. As with the case of the $L(t)$ we can calculate these times and, using Lemma 1, decide using a constant number of Moving-Voronoi queries whether s_q is above or below Voronoi vertex $v(t)$ at time $t = t^*$.

Thus, the point location algorithm of [7] provides us with $O(\log^2 n)$ questions as described above, at each question we get at most a constant number of time values for which we answer a constant number of Moving-Voronoi queries in $O(\log^2 n)$ time to determine the next branching in the point location algorithm. At the end of the parametric point location algorithm we have located s_q at time $t = t^*$. The full parametric point location procedure uses $O(\log^4 n)$ time. The initial binary search used only $O(\log^3 n)$ time so the total cost of performing a Dog-Bites-Postman query is $O(\log^4 n)$.

Theorem 1. *A Moving-Voronoi query for n postmen can be answered in time $O(\log^2 n)$ time using space $O(T(n) \log n)$. A Dog-Bites-Postman query for n postmen slower than the dog can be answered in time $O(\log^4 n)$ using space $O(T(n) \log n)$ where $T(n)$ denotes the number of topological changes in the moving Voronoi diagram.*

4.2. Time-efficient solution

Consider a fixed infinite vertical line l perpendicular to the horizontal plane at point $s = (x, y)$ and its intersections with the faces of the cell complex $\mathcal{M}$. These intersections subdivide l into intervals such that in each interval only one postman is nearest to all points $(s, t) = (x, y, t)$ for all t in this interval. Label the interval with the index of the nearest postman. If l is tangent to a face of $\mathcal{M}$ then it is equidistant to two postmen, in which case we break ties by labeling the interval by the nearest postman with the smaller index. The labels change only at the times $t_1^s < t_2^s < \cdots < t_m^s$ when l intersects $\mathcal{M}$. We set i_j to be the index of the nearest postman to s between times t_j^s and t_{j+1}^s, $j = 1, \ldots, m$. To make our definitions consistent we set $t_0^s = -\infty$ and $t_{m+1}^s = \infty$. We call the times t_j^s the *stabbing times* and the sequence i_j, $j = 1, \ldots, m$, the *stabbing sequence* associated with s. Let us denote $T^s = \{t_1^s, \ldots, t_m^s\}$. The number of different labelings of lines can be bounded by the number of faces, edges and vertices of the projection of $\mathcal{M}$ on the (x, y) plane.

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

Lemma 2. *Fix a point s and let the stabbing sequence $i_1, \ldots, i_m$ be defined as above. Then $m \leqslant 2n$.*

Proof. An (n, s) Davenport–Schinzel sequence is a sequence composed over an alphabet of n characters, where each pair of alphabet characters is allowed to alternate in the sequence at most s times. We show that a stabbing sequence is an $(n, 2)$ Davenport–Schinzel sequence. An $(n, 2)$ Davenport–Schinzel sequence has length at most $2n$, see [1].

Suppose the stabbing sequence did contain some 2-repeating subsequence. Between each subsequence $i \ldots j$ or $j \ldots i$ there must be a time t such that

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

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 $|p_i(t) - s|^2 - |p_j(t) - s|^2$

is a quadratic equation and only has two roots, leading to a contradiction. Therefore the stabbing sequence is an $(n, 2)$ Davenport–Schinzel sequence and hence has length $m \leqslant 2n$. □

We can now propose a different approach to answering a Moving-Voronoi query. Note that between any two stabbing times t_j^s and t_{j+1}^s the vertical line through s is wholly contained within the region associated with postman p_{i_j}. If, for any query point s_q, we could access the stabbing times associated with s_q in a way that permits binary search on T^s, then, in logarithmic time, we could solve a Moving-Voronoi query (s_q, t_q) by performing a binary search on the stabbing times to find the interval that contains t_q, which will immediately give us p_{i_j} as the nearest postman to s_q at time t_q. We show below a data structure that allows us to access the stabbing times in this way, so that the Moving-Voronoi query can be performed in time $O(\log n)$.

The nice fact is that using the same data structure we can also answer the Dog-Bites-Postman query in time $O(\log n)$. This will follow from the next lemma (which also follows from Theorem 4 in [4]).

Lemma 3. *Let s be a fixed point in $\mathbb{R}^2$, and let v be a fixed speed of a query dog, such that $v > |v_i|$, $i = 1, \ldots, n$. We define a function $\rho(t)$ as follows. Let p be a postman nearest to s at time t. Set $d(t) = |p(t) - s|$ to be the distance between s and its nearest postman. Define the function $\rho : \mathbb{R} \to \mathbb{R}$,*

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

Then

(a) *ρ is a 1–1 continuous mapping from $\mathbb{R}$ to $\mathbb{R}$ such that if $t > t'$ then $\rho(t) > \rho(t')$. Furthermore $\rho(-\infty) = -\infty$ and $\rho(\infty) = \infty$.*

(b) *The quickest reachable postman for a dog starting from point s at time $\rho(t)$ with speed v, is the postman nearest to s at time t: $D(s, t, \infty) = D(s, \rho(t), v)$.*

Proof. (a) Let $t > t' \in \mathbb{R}$, $s \in \mathbb{R}^2$ and let $p = M(s,t)$ and $p' = M(s,t')$ the nearest postman of s at time t and t' (p may be equal to p').

$$\begin{aligned} d(t) &\leqslant |p'(t) - s| \quad \text{since at time } t \text{ postman } p \text{ is closer to } s \text{ than } p' \\ &\leqslant |p'(t') - s| + |p'(t) - p'(t')| \quad \text{by triangular inequality} \\ &< d(t') + v(t - t') \quad \text{since the speed of } p' \text{ is less than } v. \end{aligned}$$

Hence

$$\rho(t) = t - \frac{d(t)}{v} > t - \frac{d(t') + v(t - t')}{v} = \rho(t').$$

$\rho(-\infty) = -\infty$ and $\rho(\infty) = \infty$ follows from the fact that the nearest neighbor of s when t goes to $\pm\infty$ has a speed $< v$.

(b) From the definitions above, to reach p at time t, the dog must leave s at time less than $\rho(t)$. Now using this remark and Statement (a), a dog starting at time $\rho(t)$ from s cannot reach a postman p' at a time $t' < t$, since to do that, the dog must leave s before $\rho(t') < \rho(t)$. □

Taken together the two statements of Lemma 3 provide us with a way of answering a Dog-Bites-Postman query: Given a dog query (s, t_q, v) we locate the unique interval I_j^s such that $t_q \in J_j^s$. The index of the postman assigned to interval I_j^s immediately gives us the postman that the dog can reach quickest.

We will now show how we find the interval I_j^s such that $t_q \in J_j^s$. Recall that we assume that for a fixed s we have a sequence of stabbing times with the assigned indices of the closest postman in each interval. We can do this by performing a binary search on the m values

$$\rho(t_1^s) < \rho(t_2^s) < \cdots < \rho(t_m^s).$$

Since we do not know these values in advance we perform binary search on the set T^s. For each $t_j^s \in T^s$ we compute $d(t_j^s) = |p_{i_j}(t_j^s) - s|$ and from there $\rho(t_j^s)$. Consequently, given any t_j^s we can, in constant time, decide whether $t_q > \rho(t_j^s)$ or not. We can therefore perform an $O(\log n)$ binary search to find the interval J_j^s which contains t_q without explicitly computing the whole sequence J_j^s, $j = 1, \ldots, m$.

To review, we have just seen that if we have a data structure which returns the stabbing times T^s, in a form suitable for binary search, for any given point s, then we can solve both Moving-Voronoi queries and dog queries in $O(\log n)$ time.

We now describe such a data structure. The 3-dimensional diagram $\mathcal{M}$ is orthogonally projected on the horizontal plane ((x, y)-plane) to get a planar subdivision $\mathcal{M}^\perp$. A vertex of $\mathcal{M}$ is projected to a vertex of $\mathcal{M}^\perp$. An edge of $\mathcal{M}$ is projected to some curve in the plane. A face of $\mathcal{M}$ is projected to some planar region whose boundary is defined by edges of $\mathcal{M}$, and possibly by the silhouette of the face, which is the locus of points of vertical tangency of the projected face. All these projected features define edges and vertices of $\mathcal{M}^\perp$. Additional vertices of M_p are the intersection points of the projected curves. It is known that the silhouette of an algebraic surface patch of a constant degree consists of a constant number of connected components (the boundaries of which are also algebraic of constant degree), and that it has a constant number of extremal points in a given direction and a constant number of points of self intersection. Thus, the total number of curve segments defining $\mathcal{M}^\perp$ (projection of edges and silhouettes of the faces of $\mathcal{M}$) is $O(T(n))$. Any two curve segments in $\mathcal{M}^\perp$ intersect at most a constant number of times. Thus the number of cells in the planar subdivision $\mathcal{M}^\perp$ is $O(T(n)^2)$. The projection $\mathcal{M}^\perp$ consists of vertices, edges which are algebraic curves and regions, which are maximally connected planar cells. It is easy to see that for all points in one region there is a unique stabbing sequence.

Assume we have constructed $\mathcal{M}$ by one of the standard methods, see, e.g., [8]. After construction of the defining curve segments we construct the planar subdivision $\mathcal{M}^\perp$ by a plane sweep. The sweep stops at vertices of $\mathcal{M}$ and at intersections and cusps of the projection of the edges and the silhouettes of the faces of $\mathcal{M}$. Under the assumption that intersections and cusps of the curve segments can be computed in constant time, the sweep takes time $O(N \log N) = O(N \log n)$ where $N = O(T(n)^2)$.

During the sweep we can build a point location structure for $\mathcal{M}^\perp$ as described, e.g., by Sarnak and Tarjan [14] or Cole [3]. This point location data structure has space complexity $O(T(n)^2)$, and a point location query takes time $O(\log(T(n))) = O(\log n)$ [5,14].

Assume we are given a Moving-Voronoi query $q = (s_q, t_q)$. We first locate the region in $\mathcal{M}^\perp$ that contains the point s_q. Next we have to locate t_q in the stabbing sequence corresponding to this region. We use binary search trees to store the stabbing sequences. Since the stabbing sequences of neighboring regions are similar, the persistence-technique can be used again. Given a search tree for a connected region in $\mathcal{M}^\perp$, a constant number of updates is sufficient to build a search tree for a neighboring region. Here we can use full persistence, which allows to modify all versions. We choose a region

r_0 in $\mathcal{M}^{\perp}$ and construct a binary search tree for its stabbing sequence. For all other regions r we take a neighboring region whose binary search tree has already been constructed and can be modified according to the full-persistence-technique to get a binary search tree for the stabbing sequence of r. We can use any rooted spanning tree of the dual of the graph defined by $\mathcal{M}^{\perp}$, which has root r_0, to fix the order of search tree constructions. Since $O(1)$ updates suffice, the search tree for a region r can be constructed from the search tree of the predecessor of r in the rooted spanning in time $O(\log n)$ with $O(1)$ additional storage, cf. [5]. Alternatively partial persistence can be used as well. By traversing the rooted spanning tree, we can get a linear list (with duplicates) of neighboring regions. This list defines a sequence of updates of our search structure where only the latest version of the search structure has to be modified, i.e., partial persistence is sufficient. Since the length of the list is less than twice the number of regions this solution has the same asymptotic behavior as the solution based on full persistence.

With each region we store a pointer to (one of) the search tree(s) for its stabbing sequence. No matter whether partial or full persistence is used construction time of the whole structure is $O(n + N \log n)$ and it requires $O(n + N)$ space, where $N = O(T(n)^2)$ is the number of regions. Once a region is known we can locate t with the fully (or partially) persistent binary search tree associated to the region in time $O(\log n)$. Altogether we get Theorem 2.

Theorem 2. *A Moving-Voronoi query for n postmen can be answered in time $O(\log n)$ time using space $O(T(n)^2)$. A Dog-Bites-Postman query for n postmen slower than the dog can be answered in time $O(\log n)$ using space $O(T(n)^2)$ where $T(n)$ denotes the number of topological changes in the moving Voronoi diagram.*

5. Open problems

The major problem left open in this paper is how to solve Dog-Bites-Postman queries if the dog is slower than some of the postmen. If the dog is slower than the postmen then Lemma 1 and the correspondence between Moving-Voronoi and Dog-Bites-Postman queries described above are no longer true and it is not obvious how to construct a data structure that permits the solution of both types of queries.

It would also be nice to be able to introduce some type of systematic trade off between query time and storage requirement for this problem. In our first solution we used $O(T(n) \log n)$ space, but had time complexity $O(\log^4 n)$ to answer Dog-Bites-Postman queries while in our second solution we achieved logarithmic search time for Dog-Bites-Postman queries at the expense of squaring the storage requirements to $O(T(n)^2)$. Are there intermediate techniques that balance storage requirements and search times?

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 [7] to solving moving-Voronoi queries. Furthermore they would like to thank an anonymous referee for pointing out the use of partial persistence in the time-efficient approach to Dog-Bites-Postman queries.

References

[1] P.K. Agarwal and M. Sharir, Davenport–Schinzel Sequences and Their Geometric Applications (Cambridge Univ. Press, Cambridge, UK, 1995).
[2] G. Albers and T. Roos, Voronoi diagrams of moving points in higher dimensional spaces, in: Proc. 3rd Scand. Workshop Algorithm Theory, Lecture Notes in Computer Science 621 (Springer, Berlin, 1992) 399–409.
[3] R. Cole, Searching and storing similar lists, J. Algorithms 7 (1986) 202–220.
[4] O. Devillers and M. Golin, Dog bites postman: Point location in the moving Voronoi diagram and related problems, in: Proc. 1st Annu. European Sympos. Algorithms (ESA '93), Lecture Notes in Computer Science 726 (Springer, Berlin, 1993) 133–144.
[5] J.R. Driscoll, N. Sarnak, D.D. Sleator and R.E. Tarjan, Making data structures persistent, J. Comput. Syst. Sci. 38 (1989) 86–124.
[6] J.-J. Fu and R.C.T. Lee, Voronoi diagrams of moving points in the plane, Internat. J. Comput. Geom. Appl. 1(1) (1991) 23–32.
[7] M. Goodrich and R. Tamassia, Dynamic trees and dynamic point location, in: Proc. 23rd Annu. ACM Sympos. Theory Comput. (1991) 523–533.
[8] 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., Lecture Notes in Computer Science 570 (Springer, Berlin, 1991) 113–125.
[9] L.J. Guibas, D.E. Knuth and M. Sharir, Randomized incremental construction of Delaunay and Voronoi diagrams, Algorithmica 7 (1992) 381–413.
[10] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, J. ACM 30 (1983) 852–865.
[11] T. Roos, Voronoi diagrams over dynamic scenes, in: Proc. 2nd Canad. Conf. Comput. Geom. (1990) 209–213.
[12] T. Roos, Dynamic Voronoi diagrams, Ph.D. thesis, Bayerische Julius-Maximilians-Univ. Würzburg, Germany (1991).
[13] T. Roos and H. Noltemeier, Dynamic Voronoi diagrams in motion planning, in: Computational Geometry—Methods, Algorithms and Applications: Proc. Internat. Workshop Comput. Geom. CG '91, Lecture Notes in Computer Science 553 (Springer, Berlin, 1991) 227–236.
[14] N. Sarnak and R.E. Tarjan, Planar point location using persistent search trees, Commun. ACM 29 (1986) 669–679.
[15] K. Sugihara, Voronoi diagrams in a river, Internat. J. Comput. Geom. Appl. 2(1) (1992) 29–48.