Comp. & Maths. with Appls. Vol. 11, No. 12, pp. 1171–1181, 1985
Printed in Great Britain.

0097–4943/85 $3.00 + .00

SOME DYNAMIC COMPUTATIONAL GEOMETRY PROBLEMS

MIKHAIL J. ATALLAH
Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907, U.S.A.

(*Received* 6 *April* 1984)

Communicated by Ervin Y. Rodin

Abstract—We consider some problems in computational geometry when every one of the input points is moving in a prescribed manner.

1. INTRODUCTION

Geometric objects may represent physical entities that do not have a fixed position in space, and therefore it is natural to consider the problems of computational geometry in a framework where every one of the geometric objects considered is moving in a prescribed manner. In this paper we assume that we are dealing with n points $P_1, \ldots, P_n$ such that every coordinate of every P_i is a function of a time variable t. We use the word *dynamic* to refer to the situation when the points are moving and the word *static* for the case when they are fixed (these words are used with a different meaning in other papers, but the context in which we use them should not cause confusion).

After a few preliminaries in Section 2, Section 3 mainly deals with the problem of determining the intervals of time during which one of the points belongs to the convex hull of the collection of points. Section 4 deals with the problem of determining some properties of the points after a long time has elapsed.

2. PRELIMINARIES

Throughout this paper, we assume that the input consists of a description of the motion of every one of the points $P_1, \ldots, P_n$. Motion is assumed to be in Euclidean d-dimensional space. We restrict our attention to the case where every coordinate of every point is a polynomial in the time variable t, and if every such polynomial has degree $\leq k$ then we refer to this motion as *k-motion* (so the static case is that of 0-motion). More specifically, if O is the origin of the coordinate system, then for k-motion we have $\mathbf{OP}_i(t) = \sum_{l=0}^{k} \mathbf{C}_{il} t^l$ $(1 \leq i \leq n)$, where every $\mathbf{C}_{il}$ is a constant d-dimensional vector (throughout the paper, we use boldface for vectors, so that $\mathbf{AB}$ is the vector from A to B). The motion of P_i is entirely described by the vectors $\mathbf{C}_{il} (0 \leq l \leq k)$, so that the input for point P_i is just a list of those vectors. The *initial position* of point P_i is its position at $t = 0$, and the *velocity* of P_i is $(\mathrm{d}/\mathrm{d}t)\mathbf{OP}_i(t)$. Observe that in the case of 1-motion every point is moving on a straight line with a constant velocity. We use $d_{ij}(t)$ to denote the distance between points P_i and P_j as a function of time.

For convenience, we assume that no two points have the same initial position. On the other hand, we do *not* assume that the vectors $\mathbf{C}_{1l}, \ldots, \mathbf{C}_{nl}$ are distinct $(1 \leq l \leq k)$. Such an assumption would be too restrictive since it would even rule out the case when some points are fixed while others are moving.

The arithmetic operations involved in our algorithms are $+$, $-$, $\times$, $/$ and, in the algorithms of Section 3, the $\sqrt{\ }$ operation. If additional operations are needed by an algorithm then this will be explicitly stated. Throughout, we use $\log^* n$ to denote the smallest integer i for which $\exp_i(1) > n$, where $\exp_1(x) = e^x$ and $\exp_i(x) = e^{\exp_{i-1}(x)}$. Whenever we refer to k-motion, we are assuming that $k = O(1)$.

We need to define the following function $\lambda(n, s)$:

DEFINITION 2.1

Let $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$ and define $L_{n,s}$ as the set of strings over the alphabet Σ_n that do not contain any a_ia_i as a substring and do not contain any $\xi_{ij}^s (i \neq j)$ as a subsequence, where ξ_{ij}^s is an alternating string of length $s + 2$, defined as follows: $\xi_{ij}^1 = a_ia_ja_i$, $\xi_{ij}^{2p} = \xi_{ij}^{2p-1}a_j$ and $\xi_{ij}^{2p+1} = \xi_{ij}^{2p}a_i (p \geq 1)$. Then $\lambda(n, s)$ is the maximum length that a string in $L_{n,s}$ may have, i.e.

$$\lambda(n, s) = \text{Max}\{|\sigma| \; |\sigma \in L_{n,s}\}.$$

For example, $\lambda(n, 4)$ is the maximum length that a string over the alphabet $\{a_1, \ldots, a_n\}$ may have without containing any a_ia_i as a substring and without containing any $a_ia_ja_ia_ja_i$ $(i \neq j)$ as a subsequence. It is not hard to see that no string in $L_{n,s}$ is longer than $sn(n - 1)/2 + 1$, and therefore $\lambda(n, s)$ is well defined.

The following lemma is due to Davenport and Schinzel[4].

LEMMA 2.2

$\lambda(n, 1) = n$, and $\lambda(n, 2) = 2n - 1$.

Proof. For a proof, we refer the reader to Ref. [4] (this lemma is Theorem 1 in that reference). ■

Finding the exact value of $\lambda(n, s)$ when $s > 2$ is still an open problem. However, Szemeredi[14] has proved the following.

LEMMA 2.3

$\lambda(n, s) < cn \log^* n$, where c depends only on s.

Proof. See Ref. [14]. ■

Throughout, whenever we refer to $\lambda(n, s)$ we are assuming that s is $O(1)$. Therefore Lemma 2.3 implies that $\lambda(n, s) = O(n \log^* n)$.

Now, suppose we are given n real-valued functions of time $f_1, \ldots, f_n$, where each f_i is continuous for all values of t and has an $O(1)$ storage description, and we are asked to compute a description of the pointwise MIN of these n functions, defined by $h(t) = \text{MIN}_{1 \leq i \leq n}\{f_i(t)\}$. Note that h is continuous, and that it is typically made up of "pieces" each of which is a section of one of the f_i's (Fig. 1 shows three functions whose pointwise MIN has five pieces). More formally, a piece of h is the portion of a function f_i in an interval of time $[t_1, t_2]$ such that (i) h is identical to f_i in that interval of time, and (ii) h is not identical to any $f_j (1 \leq j \leq n)$ over an interval which properly contains $[t_1, t_2]$. The storage representation of such a piece consists of the index i together with the interval $[t_1, t_2]$ (so a piece has an $O(1)$ storage description). (Detail: If f_i and f_j are identical over the interval $[t_1, t_2]$ then we break the tie by taking min (i, j).) The desired description of h is a list of the descriptions of the successive pieces that make it up. The next lemma bounds the number of pieces that make up h if no two distinct functions f_i and f_j intersect more than s times (f_i and f_j *intersect p times* iff the equation $f_i(t) = f_j(t)$ has p real solutions).

LEMMA 2.4

Let $f_1, \ldots, f_n$ be real-valued functions of time, each of which is continuous. If no two distinct functions f_i and f_j intersect more than s times, then $h(t) = \text{MIN}_{1 \leq i \leq n}\{f_i(t)\}$ is made up of no more than $\lambda(n, s)$ pieces, and this bound is the best possible.

Proof. Scan (left to right) the pieces of h, creating as you go along a string σ over the alphabet $\{a_1, \ldots, a_n\}$, in the obvious way: If the piece you are currently looking at belongs to f_i then do $\sigma := \sigma a_i$ (for example, Fig. 1 would result in $\sigma = a_1a_2a_1a_3a_2$). The number of

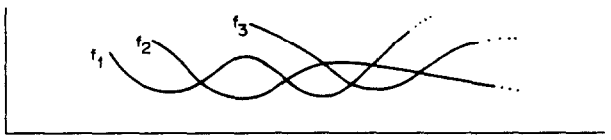

Fig. 1.

pieces that make up h is equal to $|\sigma|$. It is easy to see that $\sigma \in L_{n,s}$. To prove that the bound is tight, it suffices to show that for every string $\sigma \in L_{n,s}$ there exist n functions which satisfy the conditions of the lemma and whose associated string σ belongs to $L_{n,s}$. This can be proved by induction on n, and is omitted for the sake of conciseness (the details can be found in [1]). ■

LEMMA 2.5

Let $f_1, \ldots, f_n$ and h be as in Lemma 2.4 and, in addition, assume that (i) every f_i has an $O(1)$ storage description and can be evaluated at any t in $O(1)$ time, and (ii) for every two distinct functions f_i and f_j, the (at most s) real solutions to the equation $f_i(t) = f_j(t)$ can be computed in $O(1)$ time. Then h has $O(n \log^* n)$ pieces and its description can be computed in time $O(n \log n \log^* n)$. If $s \leq 2$ then h has $O(n)$ pieces and can be computed in time $O(n \log n)$.

Proof. The bounds on the number of pieces of h are an immediate consequence of Lemmas 2.2 and 2.3. To compute the description of h, we recursively compute the description of the pointwise MIN of $f_1, \ldots, f_{n/2}$ and that of the pointwise MIN of $f_{n/2+1}, \ldots, f_n$. Each of these two descriptions has at most $\lambda(n/2, s) \leq \lambda(n, s)$ pieces, and they can be combined to give the description of h in time $c\lambda(n, s)$, in a manner reminiscent of the way two sorted sequences are merged. This implies that, if $T(n)$ is the time this procedure takes, then we have

$$T(n) \leq 2T(n/2) + c\lambda(n, s).$$

The time bound follows from the above recurrence and Lemmas 2.2 and 2.3 ■

Not all the functions whose pointwise MIN we want to compute are continuous for all t. We now give lemmas similar to 2.4 and 2.5 for the pointwise MIN (call it h) of functions $g_1, \ldots, g_n$ which have discontinuities and are not defined for all t. It is understood that $h(t)$ is the smallest of only those g_i's that are actually defined at time t (if they are all undefined at time t then h is also undefined at t). Our formal definition of a piece of h is the same as that we gave earlier (note that in this case a piece of h may have discontinuities in its interval of time and may be undefined over portions of that interval).

Let g be a function of time. We say that g has a *transition* at time t_0 if, at time t_0, it switches between being defined and undefined (i.e. if it is undefined just before t_0 and defined just after t_0, or if it is defined just before t_0 and undefined just after t_0). Figure 2 shows a function which has two transitions (at t_1 and t_2) and two jump discontinuities (at t_3 and t_4).

LEMMA 2.6

Let $g_1, \ldots, g_n$ be real-valued functions of time, such that (i) every g_i is continuous except for at most p jump discontinuities and q transitions, has an $O(1)$ storage description and can be evaluated at any t in its domain in time $O(1)$, and (ii) no two distinct functions g_i and g_j intersect more than s times and these s intersections can be computed in time $O(1)$. Then the pointwise MIN of the g_i's is made up of $\lambda(n, s + 2p + 2q) = O(n \log^* n)$ pieces and its description can be computed in time $O(n \log n \log^* n)$.

Proof. Let h be the pointwise MIN of the g_i's, and let σ be the string obtained from h in the manner outlined in the proof of Lemma 2.4. That σ does not contain any $a_i a_i$ as a substring follows from the fact that no two consecutive pieces of h belong to the same g_i. We now show that σ does not contain any $\xi_{ij}^{s+2p+2q}(i \neq j)$ as a subsequence. We may assume that g_i and g_j are distinct since otherwise the symbol $a_{\max(i,j)}$ does not appear at all in σ (because of the tie-breaking rule previously mentioned). Let m_{ij} be the number of times one of the following takes

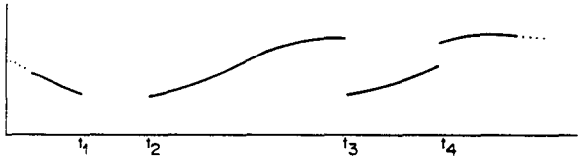

Fig. 2.

place: (i) an intersection between g_i and g_j, (ii) a transition or a jump of g_i, (iii) a transition or a jump of g_j. Note that by hypothesis we have $m_{ij} \leq s + 2p + 2q$. Now, observe that if, for $t_1 < t_2$, we have $h(t_1) = g_i(t_1)$ and $h(t_2) = g_j(t_2)$ then in the interval of time $[t_1, t_2]$ at least one of events (i)–(iii) must have taken place. This implies the following: If $a_i a_j$ is a subsequence of σ then $m_{ij} \geq 1$; if $a_i a_j a_i$ is a subsequence of σ then $m_{ij} \geq 2$; . . . ; if $\xi_{ij}^{s+2p+2q}$ is a subsequence of σ then $m_{ij} \geq s + 2p + 2q + 1$, a contradiction. Therefore $\sigma \in L_{n,s+2p+2q}$, which implies that $|\sigma| \leq \lambda(n, s + 2p + 2q) = O(n \log^* n)$. The time bound on computing the description of h follows from the divide-and-conquer approach outlined in the proof of Lemma 2.5. ■

It is clear that Lemmas 2.4–2.6 still hold if the word MIN is replaced by MAX.

3. KEEPING TRACK OF CHANGES OVER TIME

In this section we consider how some properties of the points vary as t increases from $t = 0$ to $t = \infty$. If $g(t)$ is a polynomial in t of degree $\alpha = O(1)$, none of whose coefficients depends on n, then we count the time needed to find its roots as being $O(1)$ (we make this assumption with the understanding that its practicality may be questionable for $\alpha \geq 5$).

3.1 *The convex hull*

Throughout this subsection, we assume k-motion in the plane. Let $\theta_{ij}(t)$ be the angle that $\mathbf{P}_i\mathbf{P}_j$ makes with the x axis at time t (by convention, we have $-\pi < \theta_{ij}(t) \leq +\pi$). Define $\gamma_{ij}(t)$ to be equal to $\theta_{ij}(t)$ when $\theta_{ij}(t) \geq 0$ and to be undefined otherwise, and define $\beta_{ij}(t)$ to be equal to $\theta_{ij}(t)$ when $\theta_{ij}(t) < 0$ and to be undefined otherwise.

The functions A_i, B_i, C_i, D_i are defined as follows:

$$A_i(t) = \underset{j \neq i}{\text{MIN}} \{\gamma_{ij}(t)\},$$

$$B_i(t) = \underset{j \neq i}{\text{MAX}} \{\gamma_{ij}(t)\},$$

$$C_i(t) = \underset{j \neq i}{\text{MIN}} \{\beta_{ij}(t)\},$$

$$D_i(t) = \underset{j \neq i}{\text{MAX}} \{\beta_{ij}(t)\},$$

where the MINs and MAXs at time t only involve the functions that are defined at t. If at time t every $\gamma_{ij}(1 \leq j \leq n, j \neq i)$ is undefined then A_i and B_i are both undefined at t. Similarly, if at time t every $\beta_{ij}(1 \leq j \leq n, j \neq i)$ is undefined then C_i and D_i are undefined at t.

LEMMA 3.1

Each of the functions A_i, B_i, C_i and D_i has $O(n)$ transitions and jump discontinuities.

Proof. Note that every γ_{ij} is continuous and has at most k transitions, so that the total number of transitions of the $n - 1$ functions $\gamma_{ij}(1 \leq j \leq n, j \neq i)$ is $O(n)$. Since a transition or jump of any of A_i, B_i, C_i, D_i coincides with a transition of one of the γ_{ij}'s, the lemma follows. ■

LEMMA 3.2

Each of the functions A_i, B_i, C_i, and D_i has $O(n \log^* n)$ pieces.

Proof. We give a proof for A_i (the proofs for B_i, C_i, D_i are similar). Because of Lemma 2.3, it suffices to show that A_i has no more than $\lambda(n, 4k)$ pieces. Recall that $A_i(t)$ is simply the MIN of those γ_{ij}'s that are defined at time t. Now observe that two distinct functions γ_{ij} and γ_{il} intersect at most $2k$ times, because θ_{ij} and θ_{il} intersect at most $2k$ times (verify this). In addition, every γ_{ij} is continuous and can have at most k transitions. Therefore it follows from Lemma 2.6 that A_i is made up of no more than $\lambda(n, 4k)$ pieces. ■

Let f and g be real-valued functions of t. We agree that the function $f - g$ is defined at t iff both f and g are defined at t. In this case the value of $f - g$ at t is simply $f(t) - g(t)$.

LEMMA 3.3

At time t, point P_i belongs to the convex hull iff one of the following conditions is true:

(i) $A_i(t) - D_i(t) \geq \pi$,
(ii) $B_i(t) - C_i(t) \leq \pi$,
(iii) A_i and B_i are undefined at t,
(iv) C_i and D_i are undefined at t.

(The proof of the above lemma is left to the reader.)

THEOREM 3.4

A point P_i changes between "belonging to the convex hull" and "not belonging to the convex hull" $O(n \log^* n)$ times.

Proof. Lemma 3.1 implies that conditions (iii) and (iv) of Lemma 3.3 switch between being true and false $O(n)$ times. We now bound the number of times that condition (i) of Lemma 3.3 switches between true and false (a similar argument holds for condition (ii) of that lemma). Let p, q, r be (respectively) the number of jumps, transitions and local minima of $A_i - D_i$. It is easy to see that the number of times $A_i - D_i$ switches between being $<\pi$ and $\geq\pi$ is $O(p + q + r)$. That $p + q = O(n)$ follows from Lemma 3.1. Lemma 3.2 implies that $A_i - D_i$ has $O(n \log^* n)$ pieces. Since every one of these pieces has $O(1)$ local minima, it follows that $r = O(n \log^* n)$. ∎

It is not hard to find a 1-motion example in which a point switches between belonging and not belonging (to the hull) $n - 1$ times.

COROLLARY 3.5

The sequence of hulls has $O(n^2 \log^* n)$ elements.

THEOREM 3.6

The intervals of time during which a given point belongs to the convex hull can be computed in time $O(n \log n \log^* n)$.

Proof. Note that solving $\theta_{ij}(t) = \theta_{il}(t)$ amounts to finding the instants of time at which $\mathbf{P}_i\mathbf{P}_j$ and $\mathbf{P}_i\mathbf{P}_l$ are parallel, which can be considered to take $O(1)$ time in view of the assumption stated before the theorem. This observation and Lemmas 2.3 and 2.6 imply that the representation of each of A_i, B_i, C_i, D_i can be computed in time $O(n \log n \log^* n)$. Getting the representations of $A_i - D_i$ and $B_i - C_i$ takes an additional $O(n \log^* n)$ time, and getting the instants of time at which each of the conditions of Lemma 3.3 is satisfied also takes $O(n \log^* n)$ time, since solving an equation like $\theta_{ij}(t) - \theta_{il}(t) = \pi$ amounts to finding the instants of time at which $\mathbf{P}_i\mathbf{P}_j$ and $\mathbf{P}_i\mathbf{P}_l$ are antiparallel. The theorem follows. ∎

3.2 *Other problems*

In this subsection we briefly make a few observations about the problem of keeping track, over time, of some other properties of the points. We assume k-motion in d-dimensional space.

Let S denote the sequence of points that are closest to some selected point, say P_1. The elements of S are listed in the chronological order in which they occur, so that the first element of S is the point closest to P_1 at time $t = 0$, and the last element of S is the point closest to P_1 at time $t = \infty$. W denotes the sequence of pairs of points that are closest (again, the elements of W are listed in the order in which they occur). S' and W' denote the sequences obtained by replacing the word "closest" by "farthest" in the definitions of S and W, respectively.

OBSERVATION 3.7

Each of S and S' has a length of $O(n \log^* n)$ and can be computed in time $O(n \log n \log^* n)$. If $k \leq 2$ then their length is $O(n)$ and they can be computed in time $O(n \log n)$.

OBSERVATION 3.8

Each of W and W' has a length of $O(n^2 \log^* n)$, and can be computed in time $O(n^2 \log n \log^* n)$. If $k \leq 2$ then their length is $O(n^2)$ and they can be computed in time $O(n^2 \log n)$.

Proof of Observations 3.7 and 3.8. Simply observe that in the case of k-motion every

$d_{ij}^2(t)$ is a polynomial of degree $2k$ in t, and that such functions satisfy the conditions of Lemma 2.5. ■

OBSERVATION 3.9

Computing S requires $\Omega(n \log n)$ time in the worst case.

Proof. We show that an algorithm that computes S can be used to sort n arbitrary numbers with $O(n)$ time additional work. Let $x_2, \ldots, x_{n+1}$ be arbitrary numbers to be sorted. Let $x_1 = \text{Min}\{x_2, \ldots, x_{n+1}\} - 1$, and let the input to the algorithm that computes S be the points $P_1, \ldots, P_{n+1}$ such that every P_i is initially on the x axis, at position x_i, and such that point P_1 has zero velocity, while all the other points are moving leftward on the x axis with the same constant velocity. S then consists of the numbers $x_2, \ldots, x_{n+1}$ in increasing order. ■

OBSERVATION 3.10

Computing W requires $\Omega(n^2)$ time in the worst case.

Proof. Consider 1-motion along the x axis. The trajectory of a point in the $t - x$ plane is a straight line. The n straight lines corresponding to the n points can certainly be chosen so that (i) they have $n(n-1)/2$ distinct points of intersection, and (ii) those $n(n-1)/2$ intersections have distinct projections on the t axis. Since in this case one intersection corresponds to one element of W, the length of W is at least $n(n-1)/2$. ■

Suppose that we want to compute the list I whose elements are the intervals of time during which the points can be enclosed within a rectilinear hyperbox of given dimensions.

OBSERVATION 3.11

I can be computed in time $O(n \log n \log^* n)$ ($O(n \log n)$ if $k \leq 2$).

(The proof uses Lemma 2.5 and is left to the reader.)

let δ_t be the length of the side of the smallest rectilinear hypercube that can enclose the points at time t, and let $\delta = \text{Min}_t\, \delta_t$.

OBSERVATION 3.12

δ can be computed in time $O(n \log n \log^* n)$ if $k > 2$, in time $O(n \log n)$ if $k = 2$, and in time $O(n)$ if $k = 1$.

(The proof for $k \geq 2$ follows from Lemma 2.5, that for $k = 1$ follows from the technique described in Ref. [10], and both are left to the reader.)

4. STEADY-STATE COMPUTATIONS

We use the words *steady state* to refer to conditions at time $t = \infty$. For example, the steady-state closest pair is the closest pair at $t = \infty$, i.e. it is the last element of W. In this section we give algorithms for computing steady-state properties of the moving points. The only arithmetic operations needed by these algorithms are $+$, $-$, $\times$, and $/$.

First we need to introduce some additional terminology. For k-motion and $0 \leq s \leq k$, we define the point P_{is} as being such that $\mathbf{OP}_{is}(t) = \sum_{l=0}^{s} \mathbf{C}_{il} t^l$ (recall that O is the origin of coordinates). Note that $P_{ik} = P_i$, and that P_{i0} is the initial position of P_i. We also define the (static) point V_{is} as being such that $\mathbf{OV}_{is} = \mathbf{C}_{is}$. The points $V_{1s}, \ldots, V_{ns}$ need not be distinct (we already noted that assuming them to be distinct is too restrictive), and by eliminating duplicates from among them we obtain $q_s (1 \leq q_s \leq n)$ distinct points which we call $Q_{1s}, \ldots, Q_{q_s s}$. We use N_i^s to denote the set $\{P_j \mid \mathbf{OV}_{js} = \mathbf{OQ}_{is}\}$. Observe that $N_1^s, \ldots, N_{q_s}^s$ form a partition of $\{P_1, \ldots, P_n\}$, and that $N_i^0 = \{P_i\}$ (since we assumed the initial positions to be distinct).

4.1 *Closest and farthest points*

We now consider the problem of finding the steady-state closest pair(s). We need to take

a closer look at $d_{ij}(t)$. We have

$$d_{ij}^2(t) = \|\mathbf{C}_{jk} - \mathbf{C}_{ik}\|^2 t^{2k} + 2(\mathbf{C}_{jk} - \mathbf{C}_{ik}) \cdot \sum_{\alpha=0}^{k-1} (\mathbf{C}_{j\alpha} - \mathbf{C}_{i\alpha}) t^{k+\alpha} + \| \sum_{\alpha=0}^{k-1} (\mathbf{C}_{j\alpha} - \mathbf{C}_{i\alpha}) t^{\alpha} \|^2$$

$$= \|\mathbf{V}_{ik}\mathbf{V}_{jk}\|^2 t^{2k} + 2\mathbf{V}_{ik}\mathbf{V}_{jk} \cdot \sum_{\alpha=0}^{k-1} \mathbf{V}_{i\alpha}\mathbf{V}_{j\alpha} t^{k+\alpha} + \| \sum_{\alpha=0}^{k-1} \mathbf{V}_{i\alpha}\mathbf{V}_{j\alpha} t^{\alpha} \|^2 \qquad (*)$$

where $\cdot$ stands for the scalar product between vectors and $\| \|$ is the euclidean length of a vector. Note that, for large t, the dominant term in $(*)$ is the first one, and therefore the steady-state closest pair(s) (P_i, P_j) must have smallest $\|\mathbf{V}_{ik}\mathbf{V}_{jk}\|$. If $\mathbf{V}_{ik}\mathbf{V}_{jk} \neq \mathbf{0}$ for every $i \neq j$ then the problem can be solved by enumerating in time $O(n \log n)$[3] the static (at most $O(n)$) closest pairs among $V_{1k}, \ldots, V_{nk}$ and then breaking the tie between the candidate pairs thus obtained in time $O(n)$ by using a brute-force way which is based on the observation that the coefficients of $d_{ij}^2(t)$ and $d_{uv}^2(t)$ indicate which one is smaller at $t = \infty$ (such a "comparison" between $d_{ij}^2(\infty)$ and $d_{uv}^2(\infty)$ takes constant time). Note that we are using the expression "break the tie" somewhat loosely, since even after the tie is broken there may be more than one winner (it is possible that $d_{ij}^2(t)$ and $d_{uv}^2(t)$ have exactly the same coefficients). But if the points $V_{1k}, \ldots, V_{nk}$ are not distinct then there may be $\Theta(n^2)$ pairs (P_i, P_j) which have $\mathbf{V}_{ik}\mathbf{V}_{jk} = \mathbf{0}$, and we cannot afford to use a brute-force way for breaking the tie between them. Instead, we note that for every such candidate pair, the first two terms of $(*)$ are zero, and that minimizing the third term in $(*)$ is just a steady-state closest pair problem for $k - 1$-motion. This leads to the following recursive algorithm, which returns the steady-state closest pair(s) among n input points $P_1, \ldots, P_n$ having k-motion in d-dimensional space.

Step 1. If the points $V_{1k}, \ldots, V_{nk}$ are not distinct (i.e. if $V_{ik} = V_{jk}$ for some $i \neq j$) then go to step 2. Otherwise there are $O(n)$ pairs (P_i, P_j) with smallest $\|\mathbf{V}_{ik}\mathbf{V}_{jk}\|$ and therefore we can enumerate them in time $O(n \log n)$[3] and then break the tie in time $O(n)$ (using the brute-force way already mentioned) and return the surviving pair(s).

Comment: Note that if $k = 0$ then we do not go to step 2, since we assumed that the P_i's have distinct initial positions.

Step 2. Compute $N_1^k, \ldots, N_{q_k}^k$, and for every N_i^k which contains more than one point do the following: After assigning to every $P_j \in N_i^k$ the motion of $P_{j,k-1}$, recursively find the steady-state closest pair(s) among the points in N_i^k (which now have a $k - 1$-motion). Let H_i be the set of pairs returned by this recursive call. The union of the H_i's thus obtained is the set of candidates for the closest-pair position: Break the tie between these candidates in time $O(\Sigma_i |H_i|)$ and return the surviving pair(s).

Comment: It is easy to prove by induction on k that $|H_i| = O(|N_i^k|)$. This implies that $\Sigma_i |H_i| = O(n)$.

Correctness of the above algorithm follows from the discussion preceeding it. If $T(n, k)$ is its running time, then

$$T(n, k) \leq \sum_i T(n_i, k - 1) + cn \log n,$$

where $\Sigma_i n_i \leq n$, and $T(n, 0) \leq c'n \log n$. It easily follows that $T(n, k) = O(n \log n)$, which is optimal since it is well known that $\Omega(n \log n)$ time is a lower bound for this problem in the static case[13]. This completes the proof of the following.

THEOREM 4.1

For k-motion in d-dimensional space, the steady-state closest pair(s) can be found in time $O(n \log n)$, and this is optimal.

We now consider the steady-state farthest-pair(s) problem. We restrict motion to be in the plane. If $V_{ik} \neq V_{jk}$ for every $i \neq j$ then the problem is easy: There are $O(n)$ pairs (P_i, P_j) with largest $\|\mathbf{V}_{ik}\mathbf{V}_{jk}\|$ and therefore we can enumerate them in time $O(n \log n)$[12] and then break the tie in time $O(n)$ using the brute-force method already mentioned. But if $V_{1k}, \ldots, V_{nk}$ are

not distinct then there may be $\Theta(n^2)$ pairs (P_i, P_j) having largest $\|\mathbf{V}_{ik}\mathbf{V}_{jk}\|$. We want to break the tie between these candidates without having to enumerate them. We now show how this can be done for the case of 1-motion. In this case (*) becomes

$$d_{ij}^2(t) = \|\mathbf{V}_{i1}\mathbf{V}_{j1}\|^2 t^2 + 2\mathbf{V}_{i1}\mathbf{V}_{j1} \cdot \mathbf{V}_{i0}\mathbf{V}_{j0} t + \|\mathbf{V}_{i0}\mathbf{V}_{j0}\|^2. \qquad (**)$$

Let v, w be such that $\|\mathbf{Q}_{v1}\mathbf{Q}_{w1}\|$ is largest, and let D_{vw} be an axis parallel to $\mathbf{Q}_{v1}\mathbf{Q}_{w1}$. Since all pairs (P_i, P_j) in $N_v^1 \times N_w^1$ have the same $\mathbf{V}_{i1}\mathbf{V}_{j1} (= \mathbf{Q}_{v1}\mathbf{Q}_{w1})$, the second term in (**) implies that the "best" pair in $N_v^1 \times N_w^1$ (i.e. the one with largest $d_{ij}(\infty)$) must be such that the "shadow" of $\mathbf{V}_{i0}\mathbf{V}_{j0}$ on D_{vw} is largest, i.e. $P_i \in N_v^1$ must be such that V_{i0} has smallest projection on D_{vw}, and $P_j \in N_w^1$ must be such that V_{j0} has largest projection on D_{vw} (i.e. smallest projection on D_{wv}). We use this observation for choosing the "best" pair in $N_v^1 \times N_w^1$.

The following algorithm computes the steady-state farthest pair(s) for 1-motion in the plane.

Step 1. Find $Q_{11}, \ldots, Q_{q_1 1}$ and partition the P_i's into sets $N_1^1, \ldots, N_{q_1}^1$.

Step 2. Find the set F of $O(n)$ farthest pairs among $Q_{11}, \ldots, Q_{q_1 1}$. If there exists some $(Q_{v1}, Q_{w1}) \in F$ such that $|N_v^1| > 1$ or $|N_w^1| > 1$ then go to step 3. Otherwise the set $\cup N_v^1 \times N_w^1$ (where the union is over all $(Q_{v1}, Q_{w1}) \in F$) consists of $|F|$ $(= O(n))$ candidate pairs. Break the tie between the candidate pairs, and then output the surviving pair(s) and halt.

Step 3. For every $(Q_{v1}, Q_{w1}) \in F$, let D_{vw} and D_{wv} be axes that are parallel to $Q_{v1}Q_{w1}$ and $\mathbf{Q}_{w1}\mathbf{Q}_{v1}$, respectively. Let DIR be the set of all such axes, and note that $|\mathrm{DIR}| = 2|F|$. Now, for every Q_{v1} which appears in some pair of F, let $\mathrm{DIR}_v = \{D_{vw} \mid D_{vw} \in \mathrm{DIR}\}$, and then find for every direction in DIR_v the point in N_v^1 which corresponds to it, where a point of N_v^1 is said to *correspond* to direction D_{vw} iff no other point in N_v^1 has a V_{i0} with a smaller projection on D_{vw} (we have assumed that to a given D_{vw} corresponds only one point of N_v^1, but the algorithm can easily be modified to handle the general case).

Step 4. Let $F' = \{(P_i, P_j) \in N_v^1 \times N_w^1 | (Q_{v1}, Q_{w1}) \in F; P_i \text{ corresponds to } D_{vw}, P_j \text{ corresponds to } D_{wv}\}$ (note that $|F'| = O(n)$). Break the tie among the pairs in F' and then output the surviving pair(s) and halt.

THEOREM 4.2

For 1-motion in the plane, the above algorithm finds the steady-state farthest pair(s) in time $O(n \log n)$.

Proof. Correctness of the algorithm follows from the discussion preceeding it. The only step of the algorithm where it is not obvious that the time needed is $O(n \log n)$ is that part of step 3 which has to do with computing the correspondance between points of N_v^1 and directions in DIR_v. Lemma 4.3 (which follows) implies that this can be done in $O(|N_v^1| \log |N_v^1| + |\mathrm{DIR}_v| \log |\mathrm{DIR}_v|)$, and since $\Sigma_v |N_v^1| = n$ and $\Sigma_v |\mathrm{DIR}_v| = O(n)$ it follows that the time for step 3 is $O(n \log n)$. ∎

LEMMA 4.3

Let A be a set of (static) points, and DR be a set of oriented axes ($|A| = m$, $|\mathrm{DR}| = \delta$). For every $D \in \mathrm{DR}$, let

$$S_D = \{P \in A \mid P \text{ has smallest projection on } D\}.$$

All the S_D's can be computed in time $O(m \log m + \delta \log \delta)$.

Proof. Let $\mathrm{HA} = (A_1, \ldots, A_q)$ $(q \leq m)$ be the points of the convex hull of A listed in counterclockwise cyclic order, and let $\mathrm{SDR} = (D_1, \ldots, D_\delta)$ be the axes of DR listed by increasing value of their slope. HA can be found in time $O(m \log m)$, and SDR in time $O(\delta \log \delta)$. We now show that we can find $S_{D_1}, \ldots, S_{D_\delta}$ with an additional $O(q + \delta)$ time. For the rest of this proof, by "checking A_i against D_j" we mean comparing the projections of A_{i-1}, A_i and A_{i+1} on D_j in order to find out whether $A_i \in S_{D_j}$ or not (it is clear that knowledge of these three projections is all we need to make such a decision). For D_1, find in time $O(q)$ a point of HA that belongs to S_{D_1}; say it is A_1. From this point on, we proceed in the manner which we outline next (and which is reminescent of the way two sorted sequences are merged).

We check A_1 against D_2, D_3, . . . until we hit a D_j to which it does not correspond (possibly $j = 2$), in which case we move to A_2 and check it against D_{j-1}, D_j, . . . until we find a D_l to which it does not correspond (possibly $l = j - 1$), in which case we move to A_3 . . . etc. In this way we "scan" each of HA and SDR only once, and this implies that we spent time $O(q + \delta)$ doing so. Correctness is an immediate consequence of the following two observations: (i) The D_i's to whose S_{D_i} a given A_j belongs are consecutive in SDR (with the convention that D_1 and D_δ are consecutive), and (ii) A_i and A_{i+1} have at most one D_j to whose S_{D_j} they both belong, and in this case $A_i \notin S_{D_{j+1}}$ and $A_{i+1} \notin S_{D_{j-1}}$. ■

The steady-state farthest-pair algorithm for 1-motion can be generalized to k-motion. The details are cumbersome, but the main idea is essentially the same as that for 1-motion: First we find the set F of farthest pairs among $Q_{1k}, \ldots, Q_{q_k k}$ and then for every pair $(Q_{vk}, Q_{wk}) \in F$ we try to find the "best" pair $(P_i, P_j) \in N_v^k \times N_w^k$. We use the coefficient of t^{2k-1} in (*) to decide which pair in $N_v^k \times N_w^k$ is best and, if there is still ambiguity, we use successively the coefficients of t^{2k-2}, t^{2k-3}, . . . etc (the details, which we omit, involve repeated use of Lemma 4.3 in order to maintain the time $O(n \log n)$ performance).

THEOREM 4.4

For k-motion in the plane, the steady-state farthest pair(s) can be found in time $O(n \log n)$.

4.2 *The convex hull*

Let CH be the steady-state convex hull of the P_i's.

THEOREM 4.5

For k-motion in d-dimensional space ($d \leq 3$), CH can be computed in time $O(n \log n)$, and this is optimal.

Proof. We give a proof only for the case $d = 2$ (it illustrates the main idea). The representation we use for CH is a list of those P_i's that belong to the hull at $t = \infty$, in counterclockwise cyclic order.

If $k = 0$ then use Graham's algorithm[8] to find CH in time $O(n \log n)$. Otherwise, find (in time $O(n \log n)$) $Q_{1k}, \ldots, Q_{q_k k}$ and $N_1^k, \ldots, N_{q_k}^k$. Then, compute the (static) convex hull HQ of $Q_{1k}, \ldots, Q_{q_k k}$ (this also takes time $O(n \log n)$[8]). Now, for every $Q_{vk} \in$ HQ, recursively compute the steady-state convex hull (call it K_v) of the points $\{P_{i,k-1} \mid P_i \in N_v^k\}$, and then from K_v get the steady-state hull (call it H_v) of the points in N_v^k. Getting H_v from K_v takes time $O(|N_v^k|)$ since it suffices to replace every $P_{i,k-1}$ by P_i in the list representing K_v. A point $P_i \in N_v^k$ belongs to CH iff (i) $Q_{vk} \in$ HQ, (ii) $P_i \in H_v$, and (iii) there is a line L passing through Q_{vk} and a line L' passing through P_i such that L and L' are parallel, L is a supporting line of HQ and L' is a supporting line of H_v, and if HQ is to the right (left) of L then H_v is to the right (left) of L'. These observations imply that, once we have HQ and the H_v's, CH can be computed in time $O(n)$, in a manner which we now outline. Scan the elements of the list representing HQ and for every such element (say, Q_{vk}), go through the corresponding H_v and for every P_i on H_v check in time $O(1)$ whether it belongs to CH or not, as follows: Let Q_{uk} and Q_{wk} be (respectively) the predecessor and successor of Q_{vk} in HQ, and let P_r and P_s be (respectively) the predecessor and successor of P_i in H_v. Compute (in time $O(1)$) the steady-state direction of the vector $\mathbf{P}_i\mathbf{P}_r$ and let $\mathbf{OD}_{ir}$ be parallel to that direction. Similarly, $\mathbf{OD}_{is}$ is parallel to the steady-state direction of $\mathbf{P}_i\mathbf{P}_s$. Let $\mathbf{OE}_{vu}$ and $\mathbf{OE}_{vw}$ be parallel to $\mathbf{Q}_{vk}\mathbf{Q}_{uk}$ and $\mathbf{Q}_{vk}\mathbf{Q}_{wk}$, respectively. Now, $P_i \in$ CH iff O is not inside the convex hull of the four points D_{ir}, D_{is}, E_{vu}, E_{vw}, which can easily be verified in time $O(1)$. This implies that the time needed to compute CH after computing HQ and the H_v's is $O(\Sigma_v |N_v^k|) = O(n)$. If $T(n, k)$ is the running time of this algorithm, then

$$T(n, k) \leq \sum_i T(n_i, k - 1) + cn \log n,$$

where $\Sigma_i n_i = n$, and $T(n, 0) = c'n \log n$. It easily follows that $T(n, k) = O(n \log n)$. This is optimal because there is a well-known $\Omega(n \log n)$ lower bound for this problem in the static

case[7,12,15]. We omit the proof for the case $d = 3$ (the main idea for $d = 3$ is quite similar to the one for $d = 2$, and uses the results in[11].) ■

4.3 *Other problems*

THEOREM 4.6

For k-motion in the plane, a steady-state euclidean minimum spanning tree can be found in time $O(n \log n)$, and this is optimal.

(The proof, which we omit, uses techniques similar in flavor to those we used for Theorems 4.1 and 4.2, and depends on the fact that a static euclidean minimum spanning tree in the plane can be found in time $O(n \log n)$[13].)

THEOREM 4.7

For k-motion in the plane, the (two or three) points which determine the steady-state smallest enclosing circle can be found in time $O(n)$.

(The proof, which we omit, makes use of the $O(n)$ time algorithm for finding such a circle in the static case[10].)

5. OPEN PROBLEMS

1. Do remarks similar to 3.7 and 3.8 hold if, in the definitions of S, W, S', W' the words "closest" and "farthest" are replaced by (respectively) "p^{th} closest" and "p^{th} farthest"? This leads to the question of how many pieces make up the pointwise MIN^p of n functions, where the MIN^p of $f_1, \ldots, f_n$ at time t is the p^{th} smallest number among $f_1(t), \ldots, f_n(t)$.

2. Do remarks similar to 3.9 and 3.10 hold for S' and W', or can one compute S' and W' faster than S and W?

3. *When* do steady-state conditions settle in? Assume that $k = 1$, that CH is the steady-state hull of the moving points, and let t' be the smallest instant of time such that CH is the hull of the points for all time $t \geq t'$. Is there an $o(n^2)$ algorithm for computing t'? Similar questions can be asked for the closest- and farthest-pair problems, the minimum spanning tree problem, etc.

4. Given n red points and m blue points having 1-motion in the plane, is there a "fast" algorithm for deciding whether there is an instant of time at which the red and blue points are separable? (The obvious brute-force approach gives an $O(mn(m + n) \log (m + n))$ time solution.)

5. Given n red points and m blue points having 1-motion in the plane, is there an $o(mn)$ time algorithm for deciding whether there will ever be a collision between a red point and a blue point? If all blue points are moving on the same line, starting from the same initial position, then an $O(\max (m, n) \log \min (m, n))$ time solution is quite easy: Compute the median velocity of the blue points and let B_1 be the set of blue points whose velocity is less than the median, B_2 those whose velocity is more. Let P be the blue point having median velocity. If P collides with a red point then we are done; otherwise let A_1 be the red points that are "too fast" for a collision with P, A_2 those that are "too slow," and observe that no point in B_1 can collide with one in A_1, and that no point in B_2 can collide with one in A_2. This observation leads to a recursive algorithm whose running time $T(m, n)$ satisfies the recurrence

$$T(m, n) \leq \operatorname*{Max}_{\alpha+\beta=n} \{T(m/2, \alpha) + T(m/2, \beta)\} + cm + c'n,$$

with $T(1, n) = c'n$. From this recurrence it easily follows that $T(m, n) \leq cm \log n + c'n \log m$.

6. Let ST be the sequence of euclidean minimum-cost spanning trees of the moving points. A crude upper bound on the number of elements in ST is $O(n^4)$ (this follows from the fact that every change in the minimum spanning tree is the result of one edge becoming cheaper than another edge). Can this bound be improved? (Similar questions can be asked for many other problems.)

6. SUMMARY

We considered some problems in computational geometry when every coordinate of every point is a polynomial of constant degree in a time variable t. The problems we considered fall into two categories: (i) those dealing with the changes undergone by some properties of the points as t continuously increases from 0 to ∞, and (ii) those having to do with where some properties of the points will eventually "stabilize" and stop changing (i.e. the steady-state condition of the points).

Acknowledgements—The author is grateful to Micha Sharir for pointing out Refs. [4], [5] and [14], and to Rao Kosaraju for many helpful comments.

REFERENCES

1. M. J. Atallah, Dynamic computational geometry. TR-450, Computer Science Dept., Purdue University, West-Lafayette, IN.
2. M. Ben-Or, Lower bounds for algebraic computation trees, in *Proc. of 15th Annual Symposium on Theory of Computing*, 1983, pp. 80–86. ACM, New York, 1983.
3. J. L. Bentley and M. I. Shamos, Divide-and-conquer in multidimensional space, in *Proc. 8th Annual Symposium on Theory of Computing*, 1976, pp. 220–230. ACM, New York, 1976.
4. H. Davenport and A. Schinzel, A combinatorial problem connected with differential equations. *Amer. J. Math.* **87**, 684–694 (1965).
5. H. Davenport, A combinatorial problem connected with differential equations. *Acta Arith.* **17**, 363–372 (1965).
6. D. P. Dobkin and R. L. Lipton, On the complexity of computations under varying sets of primitives. *J. Comput. System Sci.* **18**, 86–91 (1979).
7. P. V. Emde Boas, "On the $\Omega(n \log n)$ lower bound for convex hull and maximal vector determination. *Info. Proc. Letters* **10**, 132–136 (1980).
8. R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set. *Info. Proc. Letters* **1**, 132–133 (1972).
9. H. T. Kung, F. Luccio and F. P. Preparata, On finding the maxima of a set of vectors. *J. ACM* **22**, 469–476 (1975).
10. N. Megiddo, Linear-time algorithms for linear programming in R^3 and related problems, in *Proc. 23rd Annual Symposium on Theory of Computing*, 1982, pp. 329–338. ACM, New York, 1982.
11. F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions. *Comm. ACM* **20**, 87–93 (1977).
12. M. I. Shamos, Geometric complexity, in *Proc. 7th Annual Symposium on Theory of Computing*, 1975, pp. 224–233. ACM, New York, 1975.
13. M. I. Shamos and D. Hoey, Closest-point problems, in *Proc. 16th Annual Symposium on Foundations of Computer Science*, 1975, pp. 151–162. IEEE, New York, 1975.
14. E. Szemeredi, On a problem of Davenport and Schinzel. *Acta Arith.* **25**, 213–224 (1974).
15. A. C. Yao, A lower bound to finding convex hulls. *J. ACM* **28**, 780–787 (1981).