Dynamic Computational Geometry

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Report Number:
83-450

TR 450

DYNAMIC COMPUTATIONAL GEOMETRY

Mikhail J. Atallah

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907.

Abstract

We consider problems in computational geometry when every one of the input points is moving in a prescribed manner. We present and analyze efficient algorithms for a number of problems and prove lower bounds for some of them.

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 $P_1, \cdots, P_n$ 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).

Some of the techniques for solving static computational geometry problems do not seem to help in the dynamic case, especially when we are trying to continuously update over time the information we have about the moving points (for example, as the points move their voronoi diagram [10,11] seems too expensive to maintain). On the other hand, some issues which were irrelevant in the static case are very important in the dynamic one. An example of this is the problem of computing the pointwise *MIN* of n functions of time (this problem leads to interesting questions about how long a string can be without containing any of a number of forbidden patterns). It also turns out that a number of dynamic problems can be solved by considering some suitably defined static problems.

After introducing our notation in Section 2 and making a few preliminary observations in Section 3, the algorithms and lower bound results are presented in Sections 4 and 5. Section 6 suggests problems for further research, and Section 7 concludes.

2. Terminology

Throughout this paper, we assume that the input consists of a description of the motion of every one of the points $P_1, \cdots, P_n$. 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 $\overline{OP_i}(t)=\sum_{l=0}^{k} \overline{C}_{il}\, t^l$ $(1 \leq i \leq n)$, where every $\overline{C}_{il}$ is a constant d-dimensional vector. The motion of P_i is entirely described by the vectors $\overline{C}_{il}$ $(0 \leq l \leq k)$, 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 $\frac{d}{dt}\overline{OP_i}(t)$. 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 $\overline{C}_{1l}, \cdots, \overline{C}_{nl}$ 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 4, the $\surd$ operation. If additional operations are needed by an algorithm then this will be explicitly stated.

We now define the function $\lambda(n,s)$ which will play an important role in the paper.

Definition 2.1 Let $\Sigma_n = \{a_1, a_2, \cdots, a_n\}$ and define $L_{n,s}$ as the set of strings over the alphabet Σ_n that do not contain any $a_i a_i$ as a substring and do not contain any ξ_{ij}^s $(i \neq j)$ as a subsequence, where ξ_{ij}^s is defined as follows: $\xi_{ij}^1 = a_i a_j a_i$, $\xi_{ij}^{2p} = \xi_{ij}^{2p-1} a_j$ 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.

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

For example, $\lambda(n,4)$ is the maximum length that a string over the alphabet $\{a_1, \cdots, a_n\}$ may have without containing any $a_i a_i$ as a substring and without containing any $a_i a_j a_i a_j a_i a_j$ $(i \neq j)$ 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. (We will soon show that the bound just given is not tight.)

3. Preliminaries

This section presents a few preliminary observations which will be needed later in the paper. The first such observation has to do with the function $\lambda(n,s)$.

Lemma 3.1 $\lambda(n,1)=n$, and $\lambda(n,2)=2n-1$.

Proof: That $\lambda(n,1)=n$ is trivial. We now prove that $\lambda(n,2)=2n-1$. Let $\sigma \in L_{n,2}$, i.e. σ is a string over the alphabet $\{a_1, \cdots, a_n\}$, does not contain any $a_i a_i$ as a substring and does not contain any $a_i a_j a_i a_j$ $(i \neq j)$ as a subsequence. We prove that $|\sigma| \leq 2n-1$ by induction on n. The basis $(n=1)$ is trivial. For the induction step, assume $n>1$ and let a_i be the first symbol in σ, so that $\sigma = a_i \beta$. If a_i does not appear in β then $\beta \in L_{n-1,2}$ and therefore by the induction hypothesis $|\beta| \leq 2n-3$, and therefore $|\sigma| \leq 2n-2$. If a_i appears in β then $\sigma = a_i \eta a_i \gamma$ where a_i does not appear in η and $|\eta| \neq 0$. Observe that no a_j $(j \neq i)$ appears in both η and γ since otherwise σ would contain the "forbidden" subsequence $a_i a_j a_i a_j$. Therefore $\eta \in L_{p,2}$ and $a_i \gamma \in L_{q,2}$ where $q+p=n$ and $1 \leq p,q < n$. The induction hypothesis gives

$$|\sigma|-1 = |\eta| + |a_i \gamma| \leq 2p-1+2q-1 = 2n-2.$$

This completes the proof that every $\sigma \in L_{n,2}$ must have $|\sigma| \leq 2n-1$. Since the string $a_1a_2a_1a_3a_1 \cdots a_1a_na_1$ belongs to $L_{n,2}$ and has length $2n-1$ it follows that $\lambda(n,2)=2n-1$. ■

Now, suppose we are given n real-valued functions of time $f_1, \cdots, f_n$, 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 $h(t) = \underset{1 \le i \le n}{MIN}\{f_i(t)\}$. Note that h is continuous for all values of t, and that it is typically made up of "pieces" each of which is a section of one of the f_i's (figure 1 shows 3 functions whose pointwise *MIN* has 5 pieces).

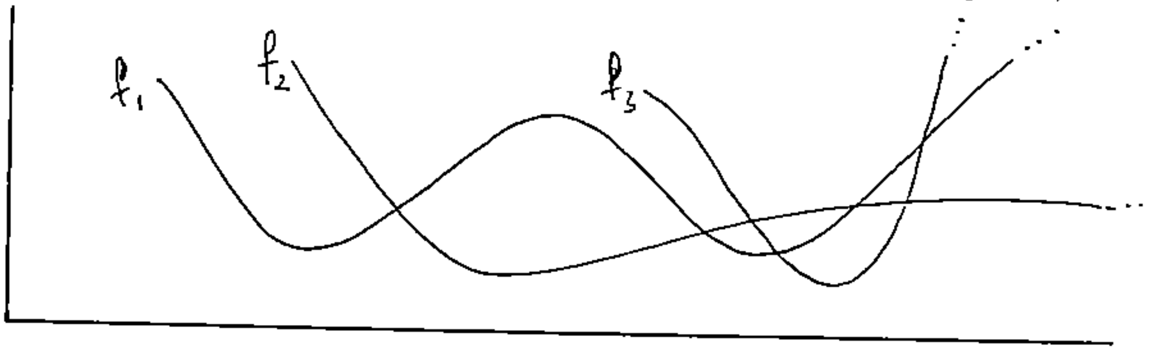

Figure 1.

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 $(1 \le j \le n)$ 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 3.2 Let $f_1, \cdots, f_n$ be real-valued functions of time, each of which is continuous for all values of t. If no two distinct functions f_i and f_j intersect more than s times, then $h(t)=\underset{1\le i\le n}{MIN}\{f_i(t)\}$ 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 $\{a_1, \cdots, a_n\}$, in the obvious way: If the piece you are currently looking at belongs to f_i then do $\sigma := \sigma a_i$ (for example, figure 1 would result in $\sigma=a_1a_2a_1a_3a_2$). The number of pieces that make up h is equal to $|\sigma|$. We now prove that $\sigma \in L_{n,s}$. That σ does not contain any a_ia_i as a substring follows from the fact that no two consecutive pieces of h belong to the same f_i. That σ does not contain the subsequence "forbidden" in $L_{n,s}$ can be seen from the following observations: If $a_ia_ja_i$ is a subsequence of σ then f_i and f_j must have intersected at least twice, if $a_ia_ja_ia_j$ is a subsequence of σ then f_i and f_j must have intersected at least 3 times, ... , if ξ_{ij}^s is a subsequence of σ then f_i and f_j must have intersected at least $s+1$ times, a contradiction. This shows that $\sigma \in L_{n,s}$, and therefore h is made up of no more than $\lambda(n,s)$ pieces. We now prove that the bound is tight. We say that n functions are *valid* if they satisfy the conditions of the lemma, and we say that they *give rise to* σ if σ is the string obtained from their pointwise MIN in the manner we outlined above. It clearly suffices to show that for every string $\sigma \in L_{n,k}$ there exist n valid functions which give rise to σ (because then the choice $\sigma \in L_{n,k}$, $|\sigma|=\lambda(n,k)$ would imply that there are n valid functions whose pointwise MIN has exactly $\lambda(n,k)$ pieces). We prove this by induction on n. The claim is trivially true if $n=1$. For $n>1$, let σ' be the string obtained from σ by first removing from σ every occurrence of a_n (call $\hat{\sigma}$ the string resulting from this operation), and then replacing in $\hat{\sigma}$ every substring $(a_i)^r$ $(r\ge 2)$ by a_i. Since $\sigma' \in L_{n-1,k}$, the induction hypothesis implies that there are valid functions $f_1, \cdots, f_{n-1}$ which give rise to σ'. If we do not insist on the resulting $f_1, \cdots, f_n$ being valid, then finding an additional function f_n such that $f_1, \cdots, f_n$ give rise to σ is a trivial matter. Let f_n be one such function. We now sketch how f_n can be "modified" in such a way that $f_1, \cdots, f_n$ become valid while still giving rise to σ. Let h be the pointwise *MIN* of $f_1, \cdots f_n$, and call x be the leftmost piece that f_n contributes to h. Now, modify f_n so that every function f_i which contributes a piece to the left of x intersects f_n only once in the interval before x. This can be done by changing that portion of f_n which is to the left of x into a very steep decreasing straight line (it can always be made steep enough that the desired property holds). Next, make a symmetric modification to the portion of f_n which is to the right of the rightmost piece that f_n contributes to h. Now, between every two consecutive pieces (call them y and z) that f_n contributes to h, modify f_n so that it consists of a very steep increasing straight line followed (after a local maximum) by a very steep decreasing straight line: These two lines can be made steep enough that every f_i which contributes to h a piece between y and z intersects f_n only twice in that interval. These modifications result in a "modified" f_n which intersects no other f_i more than k times (because if it did then σ would have ξ_{in}^k or ξ_{ni}^k as a subsequence, a contradiction). Therefore $f_1, \cdots, f_n$ are now valid. They still give rise to σ, since the portions of f_n that are pieces of h were untouched by the modifications. ▪

Lemma 3.3 Let $f_1, \cdots, f_n$ and h be as in Lemma 3.2 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 the description of h can be computed in time $T(n)$, where

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

Proof: Recursively compute the description of the pointwise *MIN* of

$f_1, \cdots, f_{n/2}$ and that of the pointwise *MIN* of $f_{n/2+1}, \cdots, f_n$. 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 (the details are easy and are left to the reader). ▪

Lemma 3.4 If $s \leq 2$ in Lemma 3.3, then h has at most $2n-1$ pieces and its description can be computed in $O(n \log n)$ time.

Proof: An immediate consequence of Lemmas 3.1 and 3.3. ▪

Unfortunately not all the functions whose pointwise *MIN* we want to compute are continuous for all t. We now give lemmas similar to 3.2 and 3.3 for the pointwise *MIN* (call it h) of functions $g_1, \cdots, g_n$ 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).

Definition 3.5 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 3.6 Let $g_1, \cdots, g_n$ be real-valued functions of time, such that (i) every g_i is continuous except for at most p jump discontinuities, (ii) every g_i has at most q transitions, and (iii) no two distinct functions g_i and g_j intersect more than s times. Then the pointwise *MIN* of the g_i's is made up of no more than

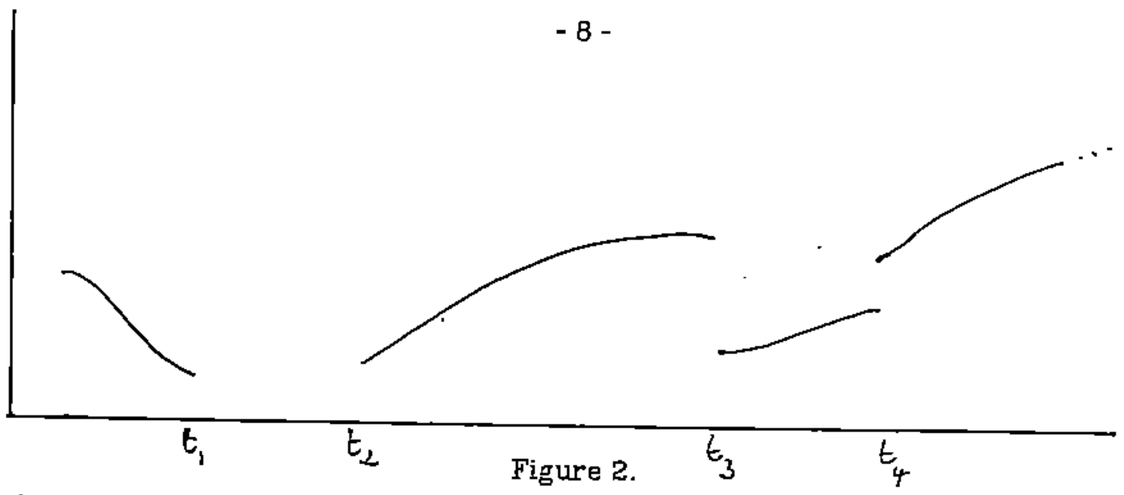

Figure 2.

$\lambda(n,s+2p+2q)$ pieces.

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 3.2. 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 (if they did then this would contradict our definition of what constitutes a piece of h). 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 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 $|\sigma| \leq \lambda(n,s+2p+2q)$. ▪

Lemma 3.7 Let $g_1, \cdots, g_n$ be as in Lemma 3.6 and, in addition, assume that (i) every g_i has an $O(1)$ storage description and can be evaluated at any t in its domain in $O(1)$ time, and (ii) for every two distinct functions g_i and g_j, the (at most s) real solutions to the equation $g_i(t)=g_j(t)$ can be computed in $O(1)$ time.

Then the description of the pointwise *MIN* of the g_i's can be computed in time $T(n)$, where

$$T(n) \le 2T(n/2) + c\lambda(n, s+2p+2q).$$

Proof: An immediate consequence of Lemma 3.6 and the divide-and-conquer approach outlined in the proof of Lemma 3.3. ▪

It is clear that Lemmas 3.2-3.7 still hold if the word *MIN* is replaced by *MAX*.

4. Transient Behavior Computations

In this section we consider how some properties of the points vary as t increases from $t=0$ to $t=\infty$ (we call this the "transient behavior" of the points because for large enough t many properties of the points "stabilize" and stop changing).

1. *Closest and Farthest Points*

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.

Theorem 4.1 For 1-motion in d-dimensional space, each of S and S' has a length of at most $2n-3$ and can be computed in $O(n \log n)$ time.

Theorem 4.2 For 1-motion in d-dimensional space, each of W and W' has a length of at most n^2-n-1 and can be computed in $O(n^2 \log n)$ time.

Proof of Theorems 4.1 and 4.2: Simply observe that in the case of 1-motion every $d_{ij}^2(t)$ is a quadratic function of time, and that quadratic functions of time satisfy the conditions of Lemma 3.4. ▪

Theorem 4.3 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 $x_2, \cdots, x_{n+1}$ be arbitrary numbers to be sorted. Let $x_1 = Min\{x_2, \cdots, x_{n+1}\} - 1$, and let the input to the algorithm that computes S be the points $P_1, \cdots, P_{n+1}$ 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 $x_2, \cdots, x_{n+1}$ in increasing order. ▪

Theorem 4.4 Computing W requires $\Omega(n^2)$ time in the worst case.

Proof: We construct an instance of the problem for which the length of W is $\Theta(n^2)$. Let every P_i be initially on the x-axis, at position $x_i = i$, and assume that every P_i moves rightward on the x-axis, with a velocity of magnitude $w_i = (n+1)^{n-i}$. Verify that P_i goes past $P_{i+1}, \cdots, P_n$ before any P_k $(i<k<n)$ catches up with P_{k+1}. Therefore every P_i will appear in $n-1$ pairs of W, so that the length of W is $n(n-1)/2$. ▪

2. *The Convex Hull*

Assume k-motion in the plane, and let $\vartheta_{ij}(t)$ be the angle that $\overline{P_i P_j}$ makes with the x-axis at time t (by convention, we have $-\pi < \vartheta_{ij}(t) \leq +\pi$). Define $\gamma_{ij}(t)$ to be equal to $\vartheta_{ij}(t)$ when $\vartheta_{ij}(t) \geq 0$ and to be undefined otherwise, and define $\beta_{ij}(t)$ to be equal to $\vartheta_{ij}(t)$ when $\vartheta_{ij}(t) < 0$ and to be undefined otherwise.

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

$$A_i(t) = \underset{j \neq i}{MIN}\{\gamma_{ij}(t)\},$$
$$B_i(t) = \underset{j \neq i}{MAX}\{\gamma_{ij}(t)\},$$
$$C_i(t) = \underset{j \neq i}{MIN}\{\beta_{ij}(t)\},$$
$$D_i(t) = \underset{j \neq i}{MAX}\{\beta_{ij}(t)\},$$

where the MIN's and MAX's at time t only involve the functions that are defined at t. If at time t every γ_{ij} $(1 \le j \le n,\ j \ne i)$ is undefined then A_i and B_i are both undefined at t. Similarly, if at time t every β_{ij} $(1 \le j \le n,\ j \ne i)$ is undefined then C_i and D_i are undefined at t.

Lemma 4.5 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 γ_{ij} $(1 \le j \le n,\ j \ne i)$ 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 4.6 Each of the functions A_i, B_i, C_i, and D_i has no more than $\lambda(n,4k)$ pieces ($\lambda(n,3)$ pieces if $k=1$).

Proof: We give a proof for A_i (the proofs for B_i, C_i, D_i are similar). 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 3.6 that A_i is made up of no more than $\lambda(n,4k)$ pieces. If $k=1$ then the bound can be improved to $\lambda(n,3)$, as follows. Let σ be the string obtained from A_i as outlined in the proof of Lemma 3.2. It suffices to show that σ cannot contain any $a_p a_j a_p a_j a_p$ $(p \ne j)$ as a subsequence. Suppose to the contrary that it does contain such a subsequence. Then γ_{ip} and γ_{ij} must be distinct and must have intersected twice, and each of them must have one transition. But for 1-motion all the γ_{ij}'s are monotonic, and therefore the fact that γ_{ip} and γ_{ij} intersect twice implies that either they are both increasing or both decreasing. We continue the proof assuming they are both increasing (a similar argument holds if they're both decreasing). In this case the graphs of these two functions are as shown in Figure 3, and it is easily seen from that figure that $a_p a_j a_p a_j a_p$ cannot be a subsequence of σ. ▪

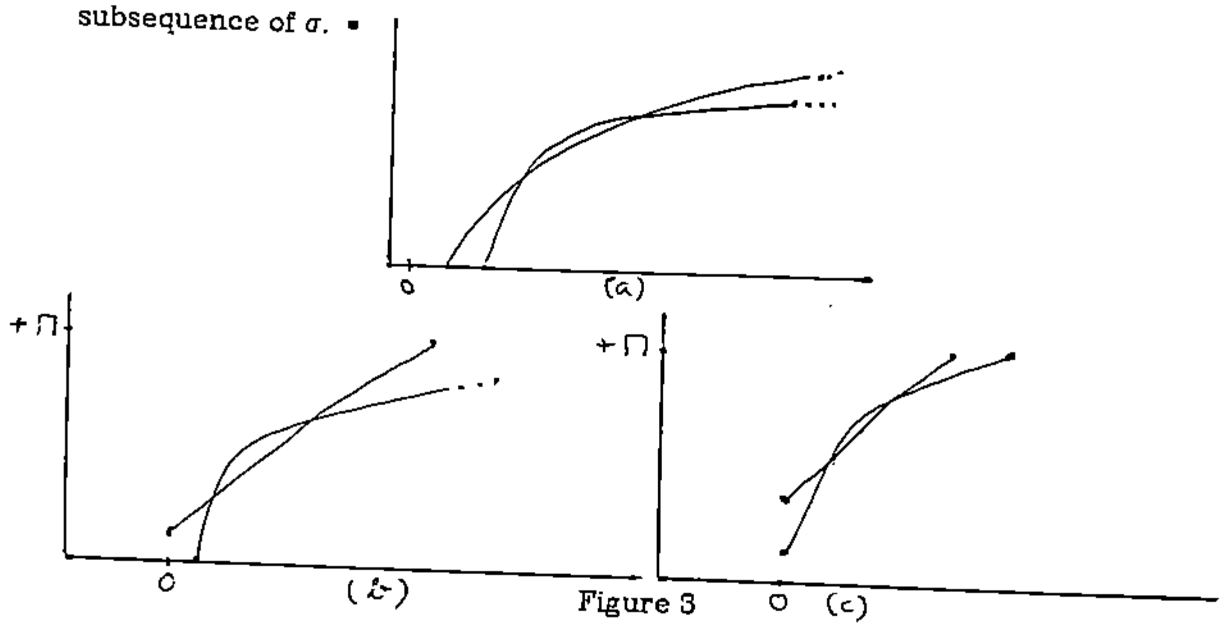

Figure 3

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 4.7 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 easy and is omitted.)

Theorem 4.8 For k-motion in the plane, a point P_i changes between "belonging to the convex hull" and "not belonging to the convex hull" $O(\lambda(n,4k))$ times ($O(\lambda(n,3))$ times if $k=1$).

Proof: Lemma 4.5 implies that conditions (iii) and (iv) of Lemma 4.7 switch between being true and false $O(n)$ times. We now bound the number of times that condition (i) of Lemma 4.7 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 4.5. Lemma 4.6 implies that, if $A_i - D_i$ has m pieces, then $m=O(\lambda(n,4k))$ ($=O(\lambda(n,3))$ if $k=1$). Since every one of these m pieces has $O(1)$ local minima, it follows that $r=O(m)$. ▪

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 4.9 For k-motion in the plane, the sequence of hulls has $O(n\lambda(n,4k))$ elements ($O(n\lambda(n,3))$ if $k=1$).

The next theorem assumes the following: If $g(t)$ is a polynomial (in t) of degree $\leq 2k$, 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 just for the sake of stating the next theorem in terms of k, and with the understanding that the practicality of such an assumption may be questionable for $k \geq 3$).

Theorem 4.10 For k-motion in the plane, let $T(n)$ be the time needed to compute the intervals of time during which a given point belongs to the convex hull. Then

$$T(n) \leq 2T(n/2) + cf(n,k),$$

where $f(n,k)$ equals $\lambda(n,4k)$ if $k \geq 2$, $\lambda(n,3)$ if $k=1$.

Proof: Note that solving $\vartheta_{ij}(t)=\vartheta_{il}(t)$ amounts to finding the instants of time at which $\overline{P_iP_j}$ and $\overline{P_iP_l}$ are parallel, which can be considered to take $O(1)$ time in view of the assumption stated before the theorem. This observation and Lemmas 4.6 and 3.7 imply that the representation of each of A_i, B_i, C_i, D_i can be computed in time $T'(n)$ where $T'(n) \le 2T'(n/2)+c'f(n,k)$. Getting the representations of A_i-D_i and B_i-C_i takes an additional $O(f(n,k))$ time, and getting the instants of time at which each of the conditions of Lemma 4.7 is satisfied also takes $O(f(n,k))$ time, since solving an equation like $\vartheta_{ij}(t)-\vartheta_{il}(t)=\pi$ amounts to finding the instants of time at which $\overline{P_iP_j}$ and $\overline{P_iP_l}$ are antiparallel. Therefore $T(n)=4T'(n)+c_1f(n,k) \le 8T'(n/2)+4c'f(n,k)+c_1f(n,k) \le 2T(n/2)+cf(n,k)$. ▪

Since $\lambda(n,4k)=O(n^2)$, the above theorem implies that $T(n)=O(n^2)$. We conjecture that $\lambda(n,4k)=O(n)$, in which case the theorem implies that $T(n)=O(n \log n)$.

3. *Other Problems*

Assume that motion is in the plane, and suppose that the points are initially distinct (i.e. no two points occupy the same position at $t=0$). The question to be answered is: Will the points remain distinct for all $t>0$? If not, then some pair of points will eventually "collide" (presumably a collision is a bad thing to happen and in a real-world situation something will be done to avoid it). Let A be an algorithm which answers "no" if no two points ever collide and "yes" otherwise, and let $T_A(n)$ be the worst-case running time of A. We do not know if there is an A such that $T_A(n)=o(n^2)$, even for 1-motion (but if all points are moving on the same straight line then there is a trivial $O(n \log n)$ time solution).

Theorem 4.11 $T_A(n)=\Omega(n \log n)$.

Proof 1: We can in $O(n)$ time reduce the element uniqueness problem [4] to the collision problem, as follows: Suppose that we want to test whether two of the numbers $c_1, \cdots, c_n$ are equal. Let $\beta = Max\{|c_1|, \cdots, |c_n|\}+1$, and let $w_i = c_i + \beta$, $1 \leq i \leq n$. Note that the w_i's are positive, that $w_i = w_j$ iff $c_i = c_j$, and that finding the w_i's from the c_i's takes $O(n)$ time. Now create, in $O(n)$ time, the following instance of the collision problem: Choose the points $P_1, \cdots, P_n$ such that (i) the initial position of P_i is on the unit circle centered at the origin, at coordinates $(x_i, y_i) = (\frac{i^2-1}{i^2+1}, \frac{2i}{i^2+1})$, and (ii) every P_i is moving toward the origin O of coordinates on a straight line, with a velocity of magnitude w_i. Observe that two of the w_i's are equal iff two of the moving points collide, which completes the reduction (note that the arithmetic operations used in the reduction are $+, -, \times, /$). Since testing whether n arbitrary numbers are distinct requires $\Omega(n \log n)$ time in the worst case [2,4], the theorem follows.

Proof 2: Consider instances of the problem where motion is restricted to be on the x-axis, and let x_i and v_i denote (respectively) the initial position and the velocity of point P_i. An algorithm for solving such instances of the problem can answer "no collision occurs" only after making sure that $x_i < x_j$ implies $v_i \leq v_j$, which is equivalent to verifying that in the set of two-dimensional vectors $(x_1, -v_1), \cdots, (x_n, -v_n)$, no vector dominates another one (we say that (a,b) *dominates* (c,d) iff $a>c$ and $b>d$). It is well known that verifying this requires $\Omega(n \log n)$ time in the worst case [5,7]. ▪

Assume k-motion in d-dimensional space, and 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 hyperrectangle of given dimensions.

Theorem 4.12 For $k \leq 2$, I can be computed in $O(n \log n)$ time.

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

Theorem 4.13 For $k \leq 2$, δ can be computed in $O(n \log n)$ time.

The last two theorems are easy to prove, using Lemma 3.4 (we leave the details to the reader).

5. 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 T. 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 $\overline{OP_{is}}(t) = \sum_{l=0}^{s} \overline{C}_{il} t^l$ (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 $\overline{OV_{is}} = \overline{C}_{is}$. The points $V_{1s}, \cdots, V_{ns}$ 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 $Q_{1s}, \cdots, Q_{q_s s}$. We use N_i^s to denote the set $\{P_j \mid \overline{OV_{js}} = \overline{OQ_{is}}\}$. Observe that $N_1^s, \cdots, N_{q_s}^s$ form a partition of $\{P_1, \cdots, P_n\}$, and that $N_i^0 = \{P_i\}$ (since we assumed the initial positions to be distinct).

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 $d_{ij}^2(t)$. We have

$$d_{ij}^2(t) = \|\overline{C}_{jk} - \overline{C}_{ik}\|^2 \, t^{2k} + 2(\overline{C}_{jk} - \overline{C}_{ik}) . \sum_{a=0}^{k-1} (\overline{C}_{ja} - \overline{C}_{ia}) t^{k+a}$$

$$+\|\sum_{\alpha=0}^{k-1}(\bar{C}_{j\alpha}-\bar{C}_{i\alpha})t^{\alpha}\|^2$$

$$=\|\overline{V_{ik}V_{jk}}\|^2 t^{2k} + 2\,\overline{V_{ik}V_{jk}}\,.\sum_{\alpha=0}^{k-1}\overline{V_{i\alpha}V_{j\alpha}}\,t^{k+\alpha}$$

$$+\|\sum_{\alpha=0}^{k-1}\overline{V_{i\alpha}V_{j\alpha}}\,t^{\alpha}\|^2 \qquad (*)$$

where "." 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 $\|\overline{V_{ik}V_{jk}}\|$. If $\overline{V_{ik}V_{jk}}\neq\bar{0}$ for every $i\neq j$ then the problem can be solved by enumerating in $O(n\log n)$ time [3] the static (at most $O(n)$) closest pairs among $V_{1k},\cdots,V_{nk}$ and then breaking the tie between the candidate pairs thus obtained in $O(n)$ time 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 $V_{1k},\cdots,V_{nk}$ are not distinct then there may be $\Theta(n^2)$ pairs (P_i,P_j) which have $\overline{V_{ik}V_{jk}}=\bar{0}$, 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 $P_1,\cdots,P_n$ having k-motion in d-dimensional space:

Step 1 : If the points $V_{1k},\cdots,V_{nk}$ 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 $\|\overline{V_{ik}V_{jk}}\|$

and therefore we can enumerate them in $O(n \log n)$ time [3] and then break the tie in $O(n)$ time (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 $N_1^k, \cdots, N_{q_k}^k$, 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 $O(\sum_i |H_i|)$ time 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 $\sum_i |H_i| = O(n)$.

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

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

where $\sum_i n_i \le n$, and $T(n,0) \le c'n \log n$. 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 [11]. This completes the proof of the following

Theorem 5.1 For k-motion in d-dimensional space, the steady-state closest pair(s) can be found in $O(n \log n)$ time, and this is optimal.

We now consider the steady-state farthest pair(s) problem. We restrict motion to be in the plane. If $V_{ik} \ne V_{jk}$ for every $i \ne j$ then the problem is easy:

There are $O(n)$ pairs (P_i,P_j) with largest $\|\overline{V_{ik}V_{jk}}\|$ and therefore we can enumerate them in $O(n \log n)$ time [10] and then break the tie in $O(n)$ time using the brute force way already mentioned. But if $V_{1k}, \cdots, V_{nk}$ are not distinct then there may be $\Theta(n^2)$ pairs (P_i,P_j) having largest $\|\overline{V_{ik}V_{jk}}\|$. 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

$$d_{ij}^2(t) = \|\overline{V_{i1}V_{j1}}\|^2\, t^2 + 2\, \overline{V_{i1}V_{j1}} \cdot \overline{V_{i0}V_{j0}}\, t + \|\overline{V_{i0}V_{j0}}\|^2 \qquad (**)$$

Let v,w be such that $\|\overline{Q_{v1}Q_{w1}}\|$ is largest, and let D_{vw} be an axis parallel to $\overline{Q_{v1}Q_{w1}}$. Since all pairs (P_i,P_j) in $N_v^1 \times N_w^1$ have the same $\overline{V_{i1}V_{j1}}$ ($=\overline{Q_{v1}Q_{w1}}$), 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 $\overline{V_{i0}V_{j0}}$ 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 $Q_{11}, \cdots, Q_{q_1 1}$ and partition the P_i's into sets $N_1^1, \cdots, N_{q_1}^1$.

Step 2 : Find the set F of $O(n)$ farthest pairs among $Q_{11}, \cdots, Q_{q_1 1}$.

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 $\bigcup N_v^1 \times N_w^1$ (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 $\overline{Q_{v1}Q_{w1}}$ and $\overline{Q_{w1}Q_{v1}}$, respectively. Let DIR be the set of all such axes, and note that $|DIR|=2|F|$. Now, for every Q_{v1} which appears in some pair of F, let $DIR_v=\{D_{vw}\,|\,D_{vw}\in DIR\}$, 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 $F'=\{(P_i,P_j)\in N_v^1\times N_w^1\,|\,(Q_{v1},Q_{w1})\in F,\ P_i$ corresponds to D_{vw}, P_j 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 5.2 For 1-motion in the plane, the above algorithm finds the steady-state farthest pair(s) in $O(n\ \log n)$ time.

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 5.3 (which follows) implies that this can be done in $O(|N_v^1|\ \log|N_v^1|\ +\ |DIR_v|\ \log|DIR_v|)$, and since $\sum_v |N_v^1|=n$ and $\sum_v |DIR_v|=O(n)$ it follows that the time for step 3 is $O(n\ \log n)$. ▪

Lemma 5.3 Let A be a set of (static) points, DR be a set of oriented axes ($|A|=m$, $|DR|=\delta$). For every $D\in DR$, let

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

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

Proof: Let $HA=(A_1,\cdots,A_q)$ $(q\le m)$ be the points of the convex hull of A listed in counterclockwise cyclic order, and let $SDR=(D_1,\cdots,D_\delta)$ be the axes of DR listed by increasing value of their slope. HA can be found in $O(m\ \log m)$ time, and SDR in $O(\delta\ \log\delta)$ time. We now show that we can find $S_{D_1},\cdots,S_{D_\delta}$ 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 $O(q)$ time 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 $D_2, D_3, \cdots$ 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 $D_{j-1}, D_j, \cdots$ 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 $O(q+\delta)$ time 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 $Q_{1k}, \cdots, Q_{q_k k}$ 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 $t^{2k-2}, t^{2k-3}, \cdots$ etc (the implementation details, which we omit, involve repeated use of Lemma 5.3 in order to maintain the $O(n \log n)$ time performance).

Theorem 5.4 For k-motion in the plane, the steady-state farthest pair(s) can be found in $O(n \log n)$ time.

2. *The Convex Hull*

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

Theorem 5.5 For k-motion in d-dimensional space ($d \leq 3$), CH can be computed in $O(n \log n)$ time, and this is optimal.

Proof: We give an algorithm 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 [6] to find CH in $O(n \log n)$ time.

Otherwise, find (in $O(n \log n)$ time) $Q_{1k}, \cdots, Q_{q_k k}$ and $N_1^k, \cdots, N_{q_k}^k$. Then, compute the (static) convex hull HQ of $Q_{1k}, \cdots, Q_{q_k k}$ (this also takes $O(n \log n)$ time [6]). Now, for every $Q_{vk} \in HQ$, recursively compute the steady-state convex hull (call it K_v) of the points $\{P_{i,k-1} | P_i \in N_v^k\}$, 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 $O(|N_v^k|)$ time 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 $O(n)$ time, 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 $O(1)$ time 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 $O(1)$ time) the steady-state direction of the vector $\overline{P_i P_r}$ and let $\overline{OD_{ir}}$ be parallel to that direction. Similarly, $\overline{OD_{is}}$ is parallel to the steady-state direction of $\overline{P_i P_s}$. Let $\overline{OE_{vu}}$ and $\overline{OE_{vw}}$ be parallel to $\overline{Q_{vk} Q_{uk}}$ and $\overline{Q_{vk} Q_{wk}}$, 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 $O(1)$ time. This implies that the time needed to compute CH after computing HQ and the H_v's is $O(\sum_v |N_v^k|) = O(n)$. If $T(n,k)$ is the running time of this algorithm, then

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

where $\sum_i n_i = n$, 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 [5,10,12]. (The proof for $d=3$ is similar, and uses the results in [9].) ▪

3. *Other Problems*

Theorem 5.6 For k-motion in the plane, a steady-state euclidean minimum spanning tree can be found in $O(n \log n)$ time, and this is optimal.

(The proof, which we omit, uses techniques similar in flavor to those we used for Theorems 5.1 and 5.2, and depends on the fact that a static euclidean minimum spanning tree in the plane can be found in $O(n \log n)$ time [11].)

Theorem 5.7 For k-motion in the plane, the (two or three) points which determine the steady-state smallest enclosing circle can be found in $O(n)$ time.

(The proof, which we omit, makes use of the $O(n)$ time algorithm for finding such a circle in the static case [8].)

6. Open Problems

1. Do theorems similar to 4.1 and 4.2 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 $f_1, \cdots, f_n$ at time t is the p^{th} smallest number among $f_1(t), \cdots, f_n(t)$. We conjecture that, for functions satisfying the conditions of Lemma 3.2, the maximum number of pieces of their pointwise MIN^p is $O(n)$ for every p.

2. Do theorems similar to 4.3 and 4.4 hold for S' and W', or can one compute S' and W' faster than S and W?

3. What is the exact form of $\lambda(n,s)$ for $s \geq 3$? We conjecture that $\lambda(n,s)=f(s)n+g(s)=O(n)$.

4. Is there an $o(n^2)$ time algorithm for testing membership in $L_{n,s}$?

5. *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.

6. 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.)

7. 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're 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

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

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

8. 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.)

7. Summary

We considered 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 ∞ (i.e. the transient behavior of the points), 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

I sincerely thank Rao Kosaraju for countless helpful comments and discussions. It is also a pleasure to acknowledge a useful discussion with each of Greg

Frederickson and Mike O'Donnell.

References

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA 1974.

2. M. Ben-Or, "Lower Bounds for Algebraic Computation Trees", *Proc. of 15th Annual Symposium on Theory of Computing*, 1983, pp. 80-86.

3. J.L. Bentley and M.I. Shamos, "Divide-And-Conquer in Multidimensional Space", *Proc. 8th Annual Symposium on Theory of Computing*, 1976, pp. 220-230.

4. D.P. Dobkin and R.L. Lipton, "On the Complexity of Computations Under Varying Sets of Primitives", *J. Comput. System Sci.*, 1979, pp. 86-91.

5. P.V. Emde Boas, "On the $\Omega(n \log n)$ Lower Bound for Convex Hull and Maximal Vector Determination", *Info. Proc. Letters*, 1980, pp. 132-136.

6. R.L. Graham, "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set", *Info. Proc. Letters*, 1972, pp. 132-133.

7. H.T. Kung, F. Luccio and F.P. Preparata, "On Finding the Maxima of a Set of Vectors", *J. ACM*, 1975, pp. 469-476.

8. N. Megiddo, "Linear-Time Algorithms for Linear Programming in R^3 and Related Problems", *Proc. 23rd Annual Symposium on Theory of Computing*, 1982, pp. 329-338.

9. F.P. Preparata and S.J. Hong, "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", *Comm. of the ACM*, 1977, pp. 87-93.

10. M.I. Shamos, "Geometric Complexity", *Proc. 7th Annual Symposium on Theory of Computing*, 1975, pp. 224-233.

11. M.I. Shamos and D. Hoey, "Closest-Point Problems", *Proc. 16th Annual Symposium on Foundations of Computer Science*, 1975, pp. 151-162.

12. A.C. Yao, "A Lower Bound to Finding Convex Hulls," *J. ACM*, 1981, pp. 780-787.