DYNAMIC LOCATION PROBLEMS[*]

N. MEGIDDO

IBM Almaden Research Center, San Jose, California 95120, USA, and
Tel-Aviv University, Tel-Aviv, Israel

Abstract

A class of dynamic location problems is introduced. The relationship between a static problem and its corresponding dynamic one is studied. We concentrate on two types of dynamic problems. The first is the global optimization problem, in which one looks for the all-times optimum. The second is the steady-state problem in which one seeks to determine the steady-state behavior of the system if one exists. General approaches to these problems are discussed.

Keywords and phrases

Dynamic location, euclidean center problem, global optimization.

1. Introduction

The purpose of this paper is to present a class of dynamic location problems which can be solved by modifying existing algorithms for corresponding static ones. The present paper is based in part on [5] and was prepared for the ISOLDE III Symposium.

We concentrate here on geometric location problems. Usually, a geometric location problem is defined by a set of n points in a Euclidean space. A dynamic problem is generated when we let the points move in space. Such a framework for dealing with dynamic problems was also presented in [1]. As an example, consider the traditional (static) 1-center problem in the plane (also referred to as the problem of the

[*]Supported in part by the National Science Foundation under grants MCS-8300984, ECS-8218181 and ECS-8121741.

minimum spanning circle). The problem is defined by n (demand) points, and one is asked to find another (supply) point whose distance to the farthest demand point is minimized. Now, suppose the demand points are moving. The motion can in general be complicated. However, we discuss here only motion which is linear in the sense that the points follow straight line trajectories, possibly with different constant speeds. Equivalently, the coordinates of the points are linear functions of time. Many of the results for this linear case can be carried over to a more general case in which the motion is determined by low-order polynomials or even rational functions.

Various questions can be asked within the dynamic framework. We concentrate here on two types of questions. The first may be called the "global optimization" problem. The global optimization problem is to find an instant of time t^* at which the optimum of the static problem (obtained by freezing the points at t^*) is the best over all times. For example, in the context of the dynamic 1-center problem, we would look for an instant of time and a supply point that minimizes (over all times) the distance to the farthest demand point at that time. An obvious "application" is for finding an instant of time at which the smallest bomb could hit all the moving objects.

The second type of question may be called the "steady-state" problem. This type of question, which was recently raised in the context of computational geometry by Atallah [1], calls for determining the "steady-state" behavior in the following sense. Let us distinguish the "solution", (that is, the optimal objective-function value) from the "basis" at the optimum. For example, it is known that in the plane the 1-center is determined by two or three demand points. These points may be regarded as the basis and this definition can also be justified on traditional optimization theoretic grounds. If the demand points are moving in a polynomial manner, then there exists a basis that supports the solutions of all the static problems for sufficiently large t. In other words, the 1-center (which is itself moving) enters a steady state in which it is determined by the same two or three moving demand points. There are two facets to the steady-state problem. The first is to find the steady-state basis. The second is usually more difficult and calls for determining the precise time at which the system enters the steady state. The steady-state problem can also be regarded as the sensitivity analysis question at infinity. A similar type of question is also discussed in [3] (see also references at the end of that paper).

We use the dynamic 1-center problem to demonstrate solutions for the two types of questions we have presented.

2. Global optimization

In this section we demonstrate a solution of a global optimization problem. We work with the dynamic 1-center problem. To be more specific, consider the problem as follows. Given are n objects that follow straight line trajectories in R^3 at various speeds. Find a time and a place at which a smallest ball encloses all the objects. This problem is *not* equivalent to that of finding the smallest ball enclosing n points in R^4. Also, a related problem of finding the smallest ball that touches all the n trajectories (that is, each object touches the ball at some time) is easier than our present problem. These other two problems are somewhat "static" and can be solved in linear time (whenever the dimension is fixed) by methods resembling those of [8]. It is interesting to mention that several related problems of computational geometry in R^3 (see [1]) are not known to have algorithms that run in $o(n^2)$ time. For some, like the one of recognizing whether any two of n given lines in R^3 meet, we know how to design algorithms with time complexity of $O(n^{2-\epsilon})$ [10] and it is still an open question whether an algorithm of order $O(n\,p\,(\log n))$ (where p is a polynomial) exists. Surprisingly, the dynamic smallest ball problem in R^3 can be solved in $O(n\,(\log n)^4 \log\log n)$ time. Moreover, a randomizing algorithm for the same problem solves it in an expected time of $O(n\,(\log n)^2\,(\log\log n)^2)$. This can be accomplished by applying two powerful results: (i) The general scheme of using parallel algorithms in the design of serial ones, as proposed in [4,6], and (ii) fast parallel algorithms for the static smallest ball problem in R^3 [5,7].

In general, for dealing with a global optimization problem (of the type of dynamic 1-center) in sequential computation, it is useful to have a good parallel computation algorithm for its static counterpart. A poly-logarithmic parallel algorithm for the static problem, on $O(n\,p\,(\log n))$ processors, would in certain cases give rise to an $O(n\,p\,(\log n))$ algorithm for the dynamic problem.

Following is a brief sketch of the algorithm for the dynamic smallest ball problem. Let $F(t)$ denote the radius of the smallest ball enclosing the objects at time t. The function $F(t)$ is convex and we know how to evaluate it at any t in $O(n)$ time [7]. Moreover, we have developed in [5] a *parallel* algorithm for evaluating $F(t)$ in $O((\log n)^3 \log\log n)$ time with n processors. Denote the minimum by t^*. Simulating this parallel algorithm, we can run in $O((\log n)^3 \log\log n)$ stages with t not specified but confined to an interval $[\alpha_k, \beta_k]$ during stage k. Typically, there will also be a value γ_k ($\alpha_k < \gamma_k < \beta_k$) such that $F(\alpha_k) > F(\gamma_k)$ and $F(\beta_k) > F(\gamma_k)$, and therefore $\alpha_k < t^* < \beta_k$. The task of each processor is the same for all $t \in [\alpha_k, \gamma_k)$ and for all $t \in [\gamma_k, \beta_k]$. However, each processor may produce a small number (independent of n) of critical values of t which determine how the algorithm should proceed in the succeeding stage. Denoting these values by $\alpha_k = t_1, \ldots, t_m = \beta_k$ (where $\gamma_k = t_\ell$ for some ℓ), we then search for j such that $F(t_{j-1}) > F(t_j)$ and $F(t_{j+1}) > F(t_j)$. The next interval is $[\alpha_k, \beta_k] = [t_{j-1}, t_{j+1}]$. Since F is convex, we can perform a "Fibonacci search" over the set $\{t_1, \ldots, t_m\}$ which amounts to $O(\log m)$ evaluations of F. Each processor may need to compare two pairs of polynomials of degree 9 in t (one pair over $[\alpha_k, \gamma_k)$ and another one over $[\gamma_k, \beta_k]$), so it may produce at most eighteen critical values. It follows that $m \leqslant 18n$. The effort per stage is therefore $O(n \log n)$ and the total effort is $O(n\,(\log n)^4 \log\log n)$. We should mention here that this is in fact the number of equations of degree not greater than 9 that we may need to solve.

However, the effort involved in solving a single equation is of course independent of n. A randomizing algorithm can be designed with the aid of a parallel randomizing algorithm for the static smallest ball problem. The latter runs in $O(\log n\ (\log\log n)^2)$ expected parallel time. For the dynamic 1-center problem in R^3 it yields a sequential randomizing algorithm which runs in $O(n\ (\log n)^2\ (\log\log n)^2)$ expected time.

3. Steady-state problems

In this section we deal with a class of parameterizations which is relatively convenient for analysis, that is, the case in which the coordinates of the points are *polynomial* functions of bounded degree in terms of the time parameter t. The steady-state problem is to find a basis which supports the solution for all sufficiently large values of t (assuming there exists such a basis). The existence of a steady-state solution is closely related to the fact that any two distinct polynomial functions intersect only finitely many times. A constructive way of investigating the existence of a steady-state solution is to look at algorithms for the corresponding static problem. Suppose P is a static problem and denote by $P(t)$ the dynamic problem where each numerical input a is a polynomial function $a(t) = a_0 + a_1 t + \ldots + a_k t^k$ of the time. Suppose we know an algorithm A that can solve $P(t)$ for any given t, using only comparisons and the arithmetic operations $+$, $-$ and $\times$. For the ease of presentation, we omit divisions from our discussion. Thus, all the program variables in this case are polynomial functions of t. We can view the algorithm as working in the ring of polynomials, where comparisons are performed relative to the obvious lexicographic order. Unless we need to make some comparisons, we can solve $P(t)$ simultaneously for all values of t. In the context of steady-state computation, we can always assume that t is sufficiently large. More precisely, suppose we need to compare two polynomials $p_1(t)$ and $p_2(t)$. From the steady-state computation point of view, the comparison is concluded with one of the following three possibilities:

(i) There is t_0 such that for all $t > t_0$, $p_1(t) > p_2(t)$.
(ii) There is t_0 such that for all $t > t_0$, $p_1(t) < p_2(t)$.
(iii) There is t_0 such that for all $t \geqslant t_0$, $p_1(t) = p_2(t)$.

It is straightforward to compare any two polynomials in this sense. Notice that at this point we do not require that the value of t_0 be determined.

Now, the steady-state problem can be solved as follows. Run an algorithm for the static problem with the value of t undetermined. Carry out any comparisons by assuming a sufficiently large value of t. The steady-state solution is then represented as a polynomial function of t which is correct for all $t \geqslant t_0$ for some unknown t_0. The issue of determining the *exact* value at which the system enters the steady-state is more difficult and is discussed later.

We can now relate the complexity of the steady-state problem to that of the static one. Essentially, the progress of the algorithm for the steady-state problem is the same as that of the static one. However, the cost of making one comparison in the context of steady state may be $k + 1$ times the cost of a regular comparison, where k is the common degree of the two polynomials involved. This is because such a comparison is in fact an operation on $(k + 1)$-vectors. What is even more severe is that degrees of polynomials may grow exponentially, if the algorithm for the static problem multiplies the program variables in a certain way. Fortunately, this does not occur very often in the solution of the common geometric and graphic problems. Let us first define the *height* $h(A)$ of an algorithm A as follows. Let the height $h(I)$ of each input number I be defined as 1. Inductively, when the algorithm computes variable I_3 from variables I_1 and I_2, where I_1 and I_2 are available and have well-defined heights, then, depending on the operation, the height of I_3 is as follows. If $I_3 = I_1 + I_2$ or $I_3 = I_1 - I_2$, then $h(I_3) = \max\{h(I_1), h(I_2)\}$. If $I_3 = I_1 \times I_2$, then $h(I_3) = h(I_1) + h(I_2)$. Let $h_L(A)$ denote the maximum height $h(I)$ of a variable computed by algorithm A while solving an instance with input length of L bits. Obviously, there are algorithms for which $h_L(A)$ is not bounded. Thus, let us define $h(A)$ to be the maximum of $h_L(A)$ with the possibility of $h(A) = \infty$. We note that in most of the problems we know, the "height" is usually a small number. This follows from the fact that for solving most of the geometric problems, it suffices to have a small number of primitives. For example, compare the distances between input points, find the intersection of lines which are determined by input points (e.g. perpendicular bisectors), compare distances between points and lines determined by input points, etc. Usually in geometric problems, none of the input points are involved in a polynomial of degree greater than 5. Also, in computational problems on graphs the role of multiplication is fairly limited. One is usually required to add up weights associated with edges or vertices, or linear combinations thereof. However, the "height" of multiplication is again fixed and small. We can state the following theorem.

THEOREM 3.1.

Suppose P is a problem and A is an algorithm of finite height $h(A)$ that solves any instance of P of size L in time less than $T(L)$, employing comparisons, additions, subtractions and multiplications. Assume the numerical inputs of P are polynomials of degree smaller than k in terms of a parameter t. Under these conditions, algorithm A can be modified to solve the steady-state dynamic problem associated with P in $O(T(L))$ time.

Proof

The proof follows from the discussion preceding the theorem. Note that $kh(A)$ is an upper bound on the degree of any polynomial that may be generated throughout the computation. The cost of comparisons, additions and subtractions (of polynomials) performed by the modified algorithm are therefore at most $kh(A)$ times the respective costs of these operations in the original algorithm. Let $g(s)$ denote the number of elementary operations required for multiplying two polynomials of degree less than s. Obviously, $g(s) = o(s^2)$. Thus, the cost of a multiplication performed by the modified algorithm is no more than $g(kh(A))$ times the cost of an elementary operation. Thus, we obtain an upper bound of $O(g(kh(A))T(L))$ for the steady-state problem.

The theorem implies that, for most of the problems we know in computational geometry and on weighted graphs, the complexities of the steady-state problem and the static problem are related by a factor which is in the worst case proportional to the square of the maximum degree of a polynomial in the parameterization. The theorem generalizes in an obvious way to algorithms of unbounded height. The cost of an operation (in the ring of polynomials) of the modified algorithm is no more than $g(kh_L(A))$ times the cost of an elementary operation when the instance is of length L. Thus, we obtain an upper bound of $O(g(kh_L(A))T(L))$ for the steady-state problem in any case. Our theorem unifies the results about steady state in [1] and can be applied to many other examples. In particular, it follows that the steady-state behavior of the 1-center can be determined in linear time.

Finally, we discuss the difficulties involved in calculating the exact time at which the system enters its steady state. Let t_0 denote the *exact* value of t at which the system enters the steady state. We note that determining just *some* upper bound on t_0 is not difficult. All one has to do is maintain the maximum of upper bounds that can be associated with individual comparisons. Given that $p_1(t) > p_2(t)$ for sufficiently large t, it is easy to state some value t' such that this inequality holds for all $t \geqslant t'$. Determining the exact value of t_0 is in general much more difficult. Obviously, one needs to be able to solve the following problem: given a polynomial $p(t)$, find the largest root of $p(t)$. However, assuming this operation can be performed (at least approximately), it is still a problem to decide which of the critical values obtained along the way is really critical for the result. More specifically, suppose a certain comparison is concluded with $p_1(t) > p_2(t)$ for all $t > t'$ and it is also known that $p_1(t') = p_2(t')$ and that $p_1(t) < p_2(t)$ for $t' - \epsilon < t < t'$. This does not imply that the solution corresponding to t, such that $t' - \epsilon < t < t'$, is not the steady-state solution. There is another inherent difficulty. In most of the problems, the steady-state basis may support the optimal solution several times, alternating with other bases, even before the system enters its steady state. Thus, by evaluating the basis at an arbitrary time t, we may sometimes be able to tell that $t < t_0$ (in case we obtain a different basis), but if the basis at t is the steady-state basis, then it does not imply that $t \geqslant t_0$. Of course, if we know a value $t' \geqslant t_0$, then we can trace t backwards until the optimal basis changes and the point at which this happens is equal to t_0. However, it is desirable to have a more efficient way of computing t_0. A case in which this would be possible is when the objective function is piecewise linear and convex in t. Then, a method described in [2] could apply.

For the case of the dynamic 1-center problem, the following idea is due to Arie Tamir [9]. Suppose we have identified the steady-state basis, that is, we have either two or three points that determine the 1-center for all sufficiently large values of t. Thus, we actually have a formula describing the motion of the center during the steady state. We can evaluate the distances of all demand points from the center and then t_0 is determined as the last time the distance between a demand point and the center exceeds the radius of the steady-state circle. So, t_0 can be found in linear time once the steady-state basis is known. We may get some more insight into the difficulty in general as follows. The critical time t_0 is in general the last time a change of the optimal "basis" occurs. In the 1-center problem, the change is local in the sense that bases differ by one coordinate (so it is actually a "pivot" step). Thus, the number of possibilities for the previous basis is only linear. In contrast, consider the dynamic problem of the two farthest points. The steady-state pair does not have to intersect the pair of farthest points immediately before the system enters the steady state.

References

[1] M.J. Atallah, Dynamic computational geometry, in: *Proc. 24th IEEE Symposium on Foundations of Computer Science* (1983) p. 92.

[2] D. Gusfield, Parametric combinatorial computing and a problem of program module distribution, JACM 30(1983)551.

[3] R.G. Jeroslow, Asymptotic linear programming, Oper. Res. 21(1973)1128.

[4] N. Megiddo, Combinatorial optimization with rational objective functions, Math. of Oper. Res. 4(1979)414.

[5] N. Megiddo, Poly-log parallel algorithms for LP with an application to exploding flying objects, Carnegie-Mellon University (1982), manuscript.

[6] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, JACM 30(1983)852.

[7] N. Megiddo, Linear time algorithms for linear programming in R^3 and related problems, SIAM J. on Computing 12(1983)759.

[8] N. Megiddo, Linear programming in linear time when the dimension is fixed, JACM 31 (1984)114.

[9] A. Tamir, private communication.

[10] F.F. Yao, private communication.