Updating and Querying Databases that Track Mobile Units

Ouri Wolfson,[1] A. Prasad Sistla,[1] Sam Chamberlain,[2] Yelena Yesha[3]

Abstract

In this paper we consider databases representing information about moving objects (e.g. vehicles), particularly their location. We address the problems of updating and querying such databases. Specifically, the update problem is to determine when the location of a moving object in the database (namely its database location) should be updated. We answer this question by proposing an information cost model that captures uncertainty, deviation, and communication. Then we analyze dead-reckoning policies, namely policies that update the database location whenever the distance between the actual location and the database location exceeds a given threshold, x. Dead-reckoning is the prevalent approach in military applications, and our cost model enables us to determine the threshold x. We propose several dead-reckoning policies and we compare their performance by simulation.

Then we consider the problem of processing range queries in the database. An example of a range query is 'retrieve the objects that are currently inside a given polygon P'. We propose a probabilistic approach to solve the problem. Namely, the DBMS will answer such a query with a set of objects, each of which is associated with a probability that the object is inside P.

[1]Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60607, wolfson,sistla@eecs.uic.edu
Wolfson's research was supported in part by NSF grants IRI-9712967, CCR-9816633, and CCR-9803974, DARPA grant N66001-97-2-8901, Army Research Labs grant DAAL01-96-2-0003, NATO grant CRG-960648. Sistla's work was supported in part by NSF grants IRI-97-11925 and CCR-9803974
[2]Army Research Laboratory, Aberdeen Proving Ground, MD

[3]Center of Excellence in Space Data and Information Sciences at NASA, Goddard Space Flight Center, Greenbelt, MD

1 Introduction

1.1 Background

Consider a database that represents information about moving objects and their location. For example, for a database representing the location of taxi-cabs a typical query may be: retrieve the free cabs that are currently within 1 mile of 33 N. Michigan Ave., Chicago (to pick-up a customer); or for a trucking company database a typical query may be: retrieve the trucks that are currently within 1 mile of truck ABT312 (which needs assistance); or for a database representing the current location of objects in a battlefield a typical query may be: retrieve the friendly helicopters that are in a given region, or, retrieve the friendly helicopters that are expected to enter the region within the next 10 minutes. The queries may originate from the moving objects, or from stationary users. We will refer to the above applications as MOtion-Database (MOD) applications or moving-objects-database applications.

In the military, MOD applications arise in the context of the digital battlefield (see [23, 22]), and in the civilian industry they arise in transportation systems. For example, Omnitracs developed by Qualcomm (see [19]) is a commercial system used by the transportation industry, which enables MOD functionality. It provides location management by connecting vehicles (e.g. trucks), via satellites, to company databases.

Currently, MOD applications are being developed in an ad hoc fashion. Database management system (DBMS) technology provides a potential foundation for MOD applications, however, DBMS's are currently not used for this purpose. The reason is that there is a critical set of capabilities that have to be integrated, adapted, and built on top of existing DBMS's in order to support moving objects databases. The added capabilities include, among other things, support for spatial and temporal information, support for rapidly changing real time data, new indexing methods, and imprecision management. The objective of our Databases fOr MovINg Objects (DOMINO) project is to build an envelope containing these capabilities on top of existing DBMS's.

In this paper we address the imprecision problem. The location of a moving object is inherently imprecise because, regardless of the policy used to update the database location of a moving object (i.e. the object-location stored in the database), the database location cannot always be identical to the actual location of the object. There may be several location update policies, for example, the location is updated every x time units. In this paper we address dead-reckoning policies, namely policies that update the database whenever the distance between the actual location of a moving object m and its database location exceeds a given threshold h, say 1 mile. This means that the DBMS will answer a query "what is the current location of m?" by an answer A: "the current location is (x,y) with a deviation of at most 1 mile". Dead-reckoning is the prevalent approach in military applications.

One of the main issues addressed in this paper is how to determine the update threshold h in dead-reckoning policies. This threshold determines the location imprecision, which encompasses two related but different concepts, namely deviation and uncertainty. The deviation of a moving object m at a particular point in time t is the distance between m's actual location at time t, and its database location at time t. For the answer A above, the deviation is the distance between the actual location of m and (x,y). On the other hand, the uncertainty of a moving object m at a particular point in time t is the size of the area in which the object can possibly be. For the answer A above, the uncertainty is the area of a circle with radius 1 mile. The deviation has a cost (or penalty) in terms of incorrect decision making, and so does the uncertainty. The deviation (resp. uncertainty) cost is proportional to the size of the deviation (resp. uncertainty). The ratio between the costs of an uncertainty unit and a deviation unit depends on the interpretation of an answer such as A above, as will be explained in section 3.

In MOD applications the database updates are usually generated by the moving objects themselves. Each moving object is equipped with a Geographic Positioning System (GPS), and it updates its database location using a wireless network (e.g ARDIS, RAM Mobile Data Co., IRIDIUM, etc.). This introduces a third information cost component, namely communication. For example, RAM Mobile Data Co. charges a minimum of 4 cents per message, with the exact cost depending on the size of the message. Furthermore, there is a tradeoff between communication and imprecision in the sense that the higher the communication cost the lower the imprecision and vice versa. In this paper we propose a model of the information cost in moving objects databases, which captures imprecision and communication. The tradeoff is captured in the model by the relative costs of an uncertainty unit, a deviation unit, and a communication unit.

1.2 Location update policies

Consider an object m moving along a prespecified route. We model the database location of m by storing in the database m's starting time, starting location, and a prediction of future locations of the object. In this paper the prediction is given as the speed v of the object. Thus the database location of m can be computed by the DBMS at any subsequent point in time. [1] This method of modeling the database location was originally introduced in [11, 12] via the concept of a dynamic attribute; the method is modified here in order to handle uncertainty. The actual location of a moving object m deviates from its database location due to the fact that m does not travel at the constant speed v.

A *dead-reckoning update policy* for m dictates that there is a database-update threshold th, i.e. a deviation for which m should send a location/speed update to the database. (Note that at any point in time, since m knows its actual location and its database location, it can compute its current deviation.) *Speed dead-reckoning* [2] (sdr) is a dead-reckoning policy in which the threshold th is fixed for the duration of the trip.

In this paper we introduce another dead-reckoning update policy, called *adaptive dead reckoning(adr)*. Adr provides with each update a new threshold th that is computed using a cost based approach. th minimizes the total information cost per time unit until the next update. The total information cost consists of the update cost, the deviation cost, and the uncertainty cost. In order to minimize the total information cost per time unit between now and the next update, the moving object m has to estimate when the next update will occur, i.e. when the deviation will reach the threshold. Thus, at location update time, in order to compute the new threshold, adr predicts the future behavior of the deviation. The thresholds differ from update to update because the predicted behavior of the deviation is different.

A problem common to both sdr and adr is that the moving object may be disconnected or otherwise unable to generate location updates. In other words, although the DBMS "thinks" that updates are not generated since the deviation does not exceed the update threshold, the actual reason is that the moving object is disconnected. To cope with this problem we introduce a third policy, "disconnection detecting dead-reckoning (dtdr)". The policy avoids the regular process of checking for disconnection by trying to communicate with the moving object, thus increasing the load on the low bandwidth wireless channel. Instead, it uses a novel technique that decreases the uncertainty threshold for disconnection detection. Thus, in dtdr the threshold continuously decreases as the time interval since the last location update increases. It has a value K during the first time unit after the update, it has value $K/2$ during the second time unit after the update, it has value $K/3$ during the third time unit, etc. Thus, if the object is connected, it is increasingly likely that it will generate an update. Conversely, if the moving object does not generate an update, as the time interval since the last update increases it is increasingly likely that the moving object is disconnected. The dtdr policy computes the K that minimizes the total information cost, i.e. the sum of the update cost, the deviation cost, and the uncertainty cost.

To contrast the three policies, observe that for sdr the threshold is fixed for all location updates. For adr the threshold is fixed between each pair of consecutive updates, but it may change from pair to pair. For dtdr the threshold decreases as the period of time between a pair of consecutive updates increases.

We compare by simulation the three policies introduced in this paper namely adr, dtdr, and sdr. The parameters of the simulation are the following. The update-unit cost, namely the cost of a location-update message; the uncertainty-unit cost, namely the cost of a unit of uncertainty; deviation-unit cost, namely the cost of a unit of deviation; a speed curve, namely a function that for a period of time gives the speed of the moving object at any point in time. The comparison is done by quantifying the total information cost of each policy for a large number of combinations of the parameters. Our simulations indicate that adr is superior to sdr in the sense that it has a lower or equal information cost for every value of the update-unit cost, uncertainty-unit cost, and deviation-unit cost. Adr is superior to dtdr in the same sense; the difference between the costs of the two policies quantifies the cost of disconnection detection. For some parameters combinations the information cost of sdr is six times as high as that of adr.

Additionally, we compare the above policies with the best update policy developed in our previous work ([12]), called immediate-linear (il). The policies in ([12]) are not dead-reckoning policies, i.e. they do not update the database when the deviation reaches some bound known to the DBMS. Instead, the update time-point depends on the overall behavior of the deviation since the last update. Partly as a consequence of this, the il policy cannot consider the uncertainty factor in deciding when to update the database. Although the moving object does not provide an uncertainty at location-update time, the DBMS can in some cases compute an upper limit on the uncertainty using additional assumptions such as maximum and minimum speed of the object. However, these upper limits are unnecessarily high. Thus, when considering the cost of uncertainty the total information cost of the il policy is higher than the cost of adr, and often higher than the cost of dtdr as well.

Finally, an additional contribution of this paper is a probabilistic model and an algorithm for query processing in motion databases. In our model the location of the moving object is a random variable, and at any point in

[1] Our simulation experiments show that, even when the speed fluctuates sharply, this temporal technique reduces the number of updates to 15% of the number used by the traditional, nontemporal method in which the database simply stores the latest known location for each object; this saves 85% of the location-updates overhead.

[2] We use the term *speed* dead-reckoning to contrast it with the plain dead-reckoning (pdr) policy in which the database location is fixed until it is explicitly updated by the moving object; namely, pdr does not use dynamic attributes.

time the database location and the uncertainty are used to determine a density function for this variable. Based on this model we developed an algorithm that processes range queries such as Q='retrieve the moving objects that are currently inside a given region R'. The answer to Q is a set of objects, each of which is associated with the probability that currently the object is inside R.

In summary, our contributions in this paper are as follows:

- We propose an information cost model for moving objects databases that captures uncertainty, deviation and communication.
- We propose the adr database update policy for moving objects. It adapts the dead-reckoning threshold to the relative costs of the deviation, uncertainty and communication, and to the predicted behavior of the deviation, so that the total information cost is minimized.
- We introduce a novel technique for disconnection detection in mobile computing, namely decreasing uncertainty threshold. Intuitively, it postulates that the probability of communication should increase as the period of time since the last communication increases. Based on the decreasing uncertainty threshold technique, we propose the dtdr database update policy for moving objects. The initial threshold is optimized to minimize the total information cost.
- We use a simulation testbed to compare the policies, sdr, adr, dtdr, and the il policy developed in previous work ([12]). We conclude that adr is superior to all the other policies in the sense that it has a lower information cost. The difference between the cost of dtdr and that of adr is the cost of disconnection detection.
- We introduce a probabilistic model and algorithm for processing range queries in motion databases.

The rest of this paper is organized as follows. In section 2 we introduce the data model and discuss location attributes of moving objects. In section 3 we discuss the information cost of a trip, and in section 4 we introduce our approach to cost optimization. In section 5 we describe the three location update policies. In section 6 we present our approach to probabilistic query processing. In section 7 we discuss relevant work, and in the last section we conclude and discuss future work. In appendix A we discuss the comparison of the update policies by simulation. In appendix B we demonstrate the adr and dtdr update policies by examples.

2 The data model

In this section we define the main concepts used in this paper. A *database* is a set of object-classes. An *object-class* is a set of attributes. Some object-classes are designated as *spatial*. Each spatial object class is either a point-class, a line-class, or a polygon-class in two-dimensional space (all our concepts and results can be extended to three-dimensional space).

Point object classes are either mobile or stationary. A point object class O has a *location attribute* L. If the object class is *stationary*, its location attribute has two sub-attributes $L.x$, and $L.y$, representing the x and y coordinates of the object. If the object class is *mobile*, its location attribute has six sub-attributes, $L.route$, $L.startlocation$, $L.starttime$, $L.direction$, $L.speed$, and $L.uncertainty$.

The semantics of the sub-attributes are as follows. $L.route$ is (the pointer to) a line spatial object indicating the route on which an object in the class O is moving. Although we assume that the objects move along predefined routes, our results can be extended to free movement in space (e.g. by aircraft). We will comment on that option in the last paragraph of this section. $L.startlocation$ is a point on $L.route$; it is the location of the moving object at time $L.starttime$. In other words, $L.starttime$ is the time when the moving object was at location $L.startlocation$. We assume that whenever a moving object updates its L attribute it updates the $L.startlocation$ subattribute; thus at any point in time $L.starttime$ is also the time of the last location-update. We assume in this paper that the database updates are instantaneous, i.e. valid- and transaction- times (see [21]) are equal. Therefore, $L.starttime$ is the time at which the update occurred in the real world system being modeled, and also the time when the database installs the update. $L.direction$ is a binary indicator having a value 0 or 1 (these values may correspond to north-south, or east-west, or the two endpoints of the route). $L.speed$ is a function that represents the predicted future locations of the object. It gives the distance of the moving object from $L.startlocation$ as a function of the number t of time units elapsed since the last location-update, namely since $L.starttime$. The function has the value 0 when $t = 0$. In its simplest form (which is the only form we consider in this extended abstract) *L.speed* represents a constant speed v, i.e. the distance is $v \cdot t$. [3] *L.uncertainty* is either a constant, or a function of the number t of time units elapsed since *L.starttime*. It represents the threshold on the location deviation (the deviation is formally defined at the end of this section); when the deviation reaches the threshold, the moving object sends a location update message. Observe that the uncertainty may change automatically as the time elapsed since *L.starttime* increases; this is indeed the case for the dtdr policy.

We define the *route-distance* between two points on a given route to be the distance along the route between the two points. We assume that it is straightforward to compute the route-distance between two points, and the point at a given route-distance from another point. The *database location* of a moving object at a given point in time is defined as follows. At time *L.starttime* the database location is *L.startlocation*; the database location at time $A.starttime + t$ is the point (x,y) which is at route-distance $L.speed \cdot t$ from the point *L.startlocation*. Intuitively, the database location of a moving object m at a given time point t is the location of m as far as the DBMS knows; it is the location that is returned by the DBMS in response to a query entered at time t that retrieves m's location. Such a query also returns the uncertainty at time t, i.e. it returns an answer of the form: m is on *L.route* at most *L.uncertainty* ahead of or behind (x,y).

Since between two consecutive location updates the moving object does not travel at exactly the speed *L.speed*, the actual location of the moving object deviates from its database location. Formally, for a moving object, the *deviation* d at a point in time t, denoted $d(t)$, is the route-distance between the moving object's actual location at time t and its database location at time t. The deviation is always nonnegative. At any point in time the moving object knows its current location, and it also knows all the subattributes of its location attribute. Therefore at any point in time the (computer onboard the) moving object can compute the current deviation. Observe that at time *L.starttime* the deviation is zero.

At the beginning of the trip the moving object updates all the sub-attributes of its location attribute. Subsequently, the moving object periodically updates its current location and speed stored in the database. Specifically, a *location update* is a message sent by the moving object to the database to update some or all the sub-attributes of its location attribute. The moving object sends the location update when the deviation exceeds the *L.uncertainty* threshold, or when the moving object changes route or direction. The location update message contains at least the values for *L.speed* and *L.startlocation*. Obviously, other subattributes can also be updated. The subattribute *L.starttime* is written by the DBMS whenever it installs a location update; it denotes the time when the installation is done.

Before concluding this section we would like to point out that the results of this paper hold for free-movement modeling, i.e. for objects that move freely in space (e.g. aircraft) rather than on routes. In this case *L.route* is an infinite straight line (e.g. 60 degrees from the starting point) rather than a line-object stored in the database. Then there are two possibilities of modeling the uncertainty. The first is identical to the one described above, i.e. the uncertainty is a segment on the infinite line representing the route. In this case every change of direction constitutes a change of route, thus necessitating a location update. The second possibility is to redefine the deviation to be the Euclidean distance between the database location and the actual location, and to remove the requirement that the object updates the database whenever it changes routes. In this case *L.uncertainty* defines a circle around the database location, and a query that retrieves the location of a moving object m returns an answer of the form: m is within a circle having a radius of at most *L.uncertainty* from (x,y). Observe that the second possibility of modeling uncertainty necessitates less location updates, but the answer to a query is less informative since the uncertainty is given in two dimensional space rather than one-dimensional.

3 The information cost of a trip

In this section we define the information cost model for a trip taken by a moving object m, and we discuss information cost optimality.

At each point in time during the trip the moving object has a deviation and an uncertainty, each of which carries a penalty. Additionally the moving object sends location update messages. Thus the information cost of a trip consists of the cost of deviation, cost of communication, and cost of uncertainty.

Now we define the deviation cost. Observe first that the cost of the deviation depends both on the size of the deviation and on the length of time for which it persists. It depends on the size of the deviation since decision-making is clearly affected by it. To see that it depends on the length of time for which the deviation persists, suppose that there is one query per time unit that retrieves the location of a moving object m. Then, if the deviation persists for two time units its cost will be twice the cost of the deviation that persists for a single time

[3] Another possibility for representing future locations is a sequence of speeds, i.e., the object will move at speed v_1 until time t_1, at speed v_2 until time t_2, etc. Such a future plan is typical of, for example, a vehicle that expects various traffic conditions; or a package that first travels by truck, then by plane, then waits (speed 0) for another truck loading, etc.

unit; the reason is that two queries (instead of one) will pay the deviation penalty. Formally, for a moving object m the cost of the deviation between two time points t_1 and t_2 is given by the *deviation cost function*, denoted $COST_d(t_1, t_2)$; it is a function of two variables that maps the deviation between the time points t_1 and t_2 into a nonnegative number. In this paper we take the penalty for each unit of deviation during a unit of time to be one (1). Then, the cost of the deviation between two time points t_1 and t_2 is:

$$COST_d(t_1, t_2) = \int_{t_1}^{t_2} d(t)dt \tag{1}$$

(Recall that $d(t)$ is the deviation as a function of time).

The *update cost*, denoted C_1, is a nonnegative number representing the cost of a location-update message sent from the moving object to the database. This is the cost of the resources (i.e. bandwidth and computation) consumed by the update. The update cost may differ from one moving object to another, and it may vary even for a single moving object during a trip, due for example, to changing availability of bandwidth. The update cost must be given in the same units as the deviation cost. In particular, if the update cost is C_1 it means the ratio between the update cost and the cost of a unit of deviation per unit of time (which is one) is C_1. It also means that the moving object (or the system) is willing to use $1/C_1$ messages in order to reduce the deviation by one during one unit of time.

Now we define the uncertainty cost. Observe that, as for the deviation, the cost of the uncertainty depends both, on the size of the uncertainty and on the length of time for which it persists. Formally, for a moving object m the cost of the uncertainty between two time points t_1 and t_2 is given by the *uncertainty cost function*, denoted $COST_u(t_1, t_2)$; it is a function of two variables that maps the uncertainty between the time points t_1 and t_2 into a nonnegative number. Define the *uncertainty unit cost* to be the penalty for each unit of uncertainty during a unit of time, and denote it by C_2. Then, the cost of the uncertainty of m between two time points t_1 and t_2 is:

$$COST_u(t_1, t_2) = \int_{t_1}^{t_2} C_2 u(t)dt \tag{2}$$

where $u(t)$ is the value of the *L.uncertainty* subattribute of m as a function of time.

The uncertainty unit cost C_2 is the ratio between the cost of a unit of uncertainty and the cost of a unit of deviation. Consider an answer returned by the DBMS: "the current location of the moving object m is (x,y), with a deviation of at most u units". C_2 should be set higher than 1 if the uncertainty in such an answer is more important than the deviation, and lower than 1 otherwise. Observe that in a dead-reckoning update policy each update message establishes a new uncertainty which is not necessarily lower than the previous one. Thus communication reduces the deviation but not necessarily the uncertainty.

Now we are ready to define the information cost of a trip taken by a moving object m. Let t_1 and t_2 be the time-stamps of two consecutive location update messages. Then the *information cost* in the interval $[t_1, t_2)$ is:

$$COST_I[t_1, t_2) = C_1 + COST_d[t_1, t_2) + COST_u[t_1, t_2) \tag{3}$$

Observe that $COST_I[t_1, t_2)$ includes the message cost at time t_1 but not the cost of the one at time t_2. Observe also that each location update message writes the actual current location of m in the database, thus it reduces the deviation to zero. The total information cost of a trip is computed by summing up $COST_I[t_1, t_2)$ for every pair of consecutive update points t_1 and t_2. Formally, let the time points of the update messages sent by m be $t_1, t_2, ..., t_k$. Furthermore, let 0 be the time point when the trip started and t_{k+1} the time point when the trip ended. Then the *total information cost* of a trip is

$$COST_I = COST_d[0, t_1) + COST_u[0, t_1) + \sum_{i=1}^{k} COST_I[t_i, t_{i+1}) \tag{4}$$

4 Cost based optimization for dead reckoning policies

As mentioned in the introduction, a dead-reckoning update policy for a moving object m dictates that at any point in time there is a database-update threshold th, of which both the DBMS and m are aware. When the deviation of m reaches th, m sends to the database an update consisting of the current location, the predicted speed, and the new deviation threshold K. The objective of the dead reckoning policies that we introduce in this paper is to set K (which the DBMS installs in the *L.uncertainty* subattribute), such that the total information cost is minimized. Intuitively, this is done as follows. First, m predicts the future behavior of the deviation.

Based on this prediction, the average cost per time time unit between now and the next update is obtained as a a function f of the new threshold K. Then K is set to minimize f [4]. It is important to observe that we optimize the average cost per time unit rather than simply the total cost between the two time points; clearly, the total cost increases as the time interval until the next update increases.

The next theorem establishes the optimal value K for *L.uncertainty* under the assumption that the deviation between two consecutive updates is a linear function of time.

Theorem 1: Denote the update cost by C_1, and the uncertainty unit cost by C_2. Assume that for a moving object two consecutive location updates occur at time points t_1 and t_2. Assume further that between t_1 and t_2, the deviation $d(t)$ is given by the function $a(t-t_1)$ where $t_1 \leq t \leq t_2$ and a is some positive constant; and *L.uncertainty* is fixed at K throughout the interval (t_1, t_2). Then the total information cost per time unit between t_1 and t_2 is minimized if $K = \sqrt{\frac{2aC_1}{2C_2+1}}$.

Proof: Based on equations 1, 2 and 3:

$$COST_I[t_1,t_2) = C_1 + \int_{t_1}^{t_2} a(t-t_1)dt + C_2K(t_2-t_1) = C_1 + \frac{a(t_2-t_1)^2}{2} + C_2K(t_2-t_1) \tag{5}$$

Denote by $f(t_2)$ the average information cost per time unit between t_1 and t_2, for the update time t_2. Namely,

$$f(t_2) = \frac{COST_I[t_1,t_2)}{(t_2-t_1)} \tag{6}$$

Since the update immediately following t_1 occurs at time t_2, it means that at that time the deviation reaches the threshold *L.uncertainty*, namely:

$$K = a(t_2-t_1) \tag{7}$$

Thus we can substitute $K/a + t_1$ for t_2 in equation 6 and obtain $f(K) = \frac{aC_1}{K} + (\frac{1}{2} + C_2)K$. Using the derivative it is easy to calculate that the minimum of $f(K)$ is obtained when $K = \sqrt{\frac{2aC_1}{1+2C_2}}$. □

The implication of theorem 1 is the following. Suppose that a moving object m is currently at time point t_1, i.e. its deviation has reached the uncertainty threshold *L.uncertainty*. Now m needs to compute a new value for *L.uncertainty* and send it in the location update message. Suppose further that m predicts that following the update the deviation will behave as the linear function $a(t-t_1)$, and in the update message it has to set the uncertainty threshold *L.uncertainty* to a value that will remain fixed until the next update. Then, in order to optimize the information cost, m should set the threshold to $K = \sqrt{\frac{2aC_1}{2C_2+1}}$.

Next assume that, in order to detect disconnection, one is interested in a dead-reckoning policy in which the uncertainty threshold *L.uncertainty* continuously decreases between updates. Particularly, we consider a particular type of decrease, that we call fractional decrease; other types exist, but we found this one convenient. Let K be a constant. If the uncertainty threshold *L.uncertainty decreases fractionally starting with K*, then during the first time unit after a location update u its value is K, during the second time unit after u its value is $K/2$, during the third time unit after u its value is $K/3$, etc., until the next update (which establishes a new K).

Theorem 2: Assume that for a moving object two consecutive location updates occur at time points t_1 and t_2. Assume further that between t_1 and t_2, the deviation $d(t)$ is given by the function $a(t-t_1)$ where $t_1 \leq t \leq t_2$ and a is some positive constant; and in the time interval (t_1, t_2) *L.uncertainty* decreases fractionally starting with a constant K. Then the total information cost per time unit between t_1 and t_2 is given by the following function of K. $f(K) = \frac{C_1 + \frac{1}{2}K + C_2K(1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{\sqrt{\frac{K}{a}}})}{\sqrt{\frac{K}{a}}}$.

Proof: Based on equations 1, 2 and 3:

$$COST_I[t_1,t_2) == C_1 + \frac{a(t_2-t_1)^2}{2} + C_2K(1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{t_2-t_1}) \tag{8}$$

The deviation at time t_2 is $a(t_2-t_1)$, and the threshold at that time is $K/(t_2-t_1)$. Since the deviation reaches the threshold at time t_2 the above values are equal, and therefore $K = a(t_2-t_1)^2$ and $\sqrt{\frac{K}{a}} = (t_2-t_1)$. Thus, if we divide equation 8 by (t_2-t_1) in order to obtain the information cost *per time unit*, and then substitute K for $a(t_2-t_1)^2$ and $\sqrt{\frac{K}{a}}$ for (t_2-t_1) in the resulting equation, the theorem follows. □

[4]Let us observe that the proposed method of optimizing the new threshold K is not unique. We have devised other methods which are omitted from this extended abstract. A performance comparison among these methods is the subject of future work.

Similarly to theorem 1, the implication of theorem 2 is the following. Suppose that a moving object is currently at time point t_1, i.e. it is about to send a location update message, and it can predict that following the update the deviation will behave as the linear function $a(t - t_1)$, and in the update message it sets the uncertainty threshold *L.uncertainty* to a fractionally decreasing value starting with K. Then in order to optimize the information cost it should set K to the value that minimizes the function of theorem 2.

5 Description of the location update policies

In this section we describe and motivate the following three location update policies.

The speed dead-reckoning (sdr) policy. At the beginning of the trip the moving object m sends to the DBMS an uncertainty threshold that is selected in an ad hoc fashion, it is stored in *L.uncertainty*, and it remains fixed for the duration of the trip. The object m updates the database whenever the deviation exceeds *L.uncertainty*; the update simply includes the current location and current speed. [5] □

The adaptive dead reckoning (adr) policy. At the beginning of the trip the moving object m sends to the DBMS an initial deviation threshold th_1 selected arbitrarily. [6] Then m starts tracking the deviation. When the deviation reaches th_1, the moving object sends an update message to the database. The update consists of the current speed, current location, and a new threshold th_2 that the DBMS should install in the *L.uncertainty* subattribute. th_2 is computed as follows. Denote by t_1 the number of time units from the beginning of the trip until the deviation reaches th_1 for the first time, by I_1 the cost of the deviation (which is computed using equation 1) during the same time interval, and let $a_1 = \frac{2I_1}{t_1^2}$. Then th_2 is $\sqrt{\frac{2a_1C_1}{1+2C_2}}$ (remember, C_1 is the update cost, C_2 is the unit-uncertainty cost). When the deviation reaches th_2, a similar update is sent, except that the new threshold th_3 is $\sqrt{\frac{2a_2C_1}{1+2C_2}}$, where $a_2 = \frac{2I_2}{t_2^2}$ (I_2 is the cost of the deviation from the first update to second update, t_2 is the number of time units elapsed since the first location update). Since a_2 may be different than a_1, th_2 may be different than th_3. When th_3 is reached the object will send another update containing th_4 (which is computed in a similar fashion), and so on. □

The mathematical motivation for adr is based on theorem 1 in a straight-forward way. Namely, at each update time point p_i adr simply sets the next threshold in a way that optimizes the information cost per time unit (according to theorem 1), assuming that the deviation following time p_i will behave as the following linear function: $d(t) = \frac{2I_i}{t_i^2}t$, where t is the number of time units after p_i, and t_i is the number of time units between the immediately preceding update and the current one (at time p_i), and I_i the cost of the deviation during the same time interval. The reason for this prediction of the future deviation is as follows. Adr approximates the current deviation, i.e. the deviation from the time of the immediately preceding update to time p_i, by a linear function[7] (see [24]) with slope $\frac{2I_i}{t_i^2}t$. Observe that at time p_i this linear function has the same deviation cost (namely I_i) as the actual current deviation [8]. Based on the locality principle, adr predicts that after the update at time p_i, the deviation will behave according to the same approximation function.

For example, consider figure 4.1. At time point p_i adr predicts that the future deviation is given by the linear function with slope l.

The disconnection detection dead reckoning (dtdr) policy. At the beginning of the trip the moving object m sends to the DBMS an initial deviation threshold th_1 selected arbitrarily. The moving objects sets the uncertainty threshold *L.uncertainty* to a fractionally decreasing value starting with th_1. That is, during the first time unit the uncertainty threshold is th_1; during the second time unit period it is $\frac{th_1}{2}$, and so on. Then it starts tracking the deviation. At time t_1 when the deviation reaches the current uncertainty threshold, namely $\frac{th_1}{t_1}$, the moving object sends a location update message to the database. The update consists of the current speed, current location, and a new threshold th_2 to be installed in the *L.uncertainty* subattribute.

th_2 is computed using the function $f(K)$ of theorem 2. Since $f(K)$ uses the slope a of the future deviation, we first estimate the future deviation as in the adr case, as follows. Denote by I_1 the cost of the deviation (which is computed using equation 1) since the beginning of the trip, and let $a_1 = \frac{2I_1}{t_1^2}$. Now, observe that

[5] Sdr can also use another speed, for example, the average speed since the last update, or the average speed since the beginning of the trip, or a speed that is predicted based on knowledge of the terrain. This comment holds for the other policies discussed in this section.

[6] When comparing the adr and sdr policies by simulation (see the appendix), both policies use as their first threshold the same value, and the same holds for the initial speed.

[7] More powerful, nonlinear approximation functions have been considered, but their discussion is omitted from this extended abstract.

[8] In this sense we are using a simple linear regression, but instead of the common least squares method we employ an equal sums method that is more appropriate for our cost function.

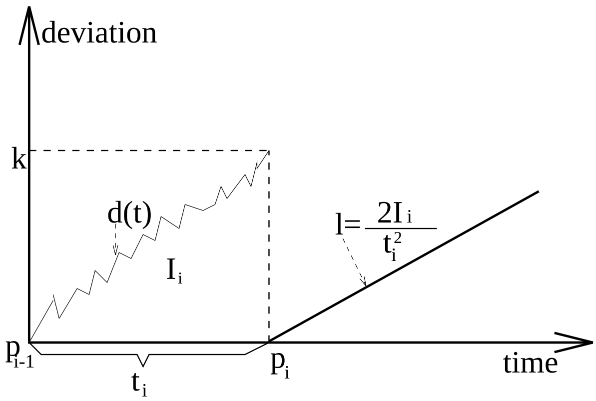


Figure 4.1 At time p_i the deviation is predicted to behave as a linear function with slope l

$f(K)$ does not have a closed form formula. Thus we first approximate the sum $\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\sqrt{\frac{K}{a_1}}}$ by $ln(\sqrt{\frac{K}{a_1}})$ (since ln$n$ is an approximation for the nth harmonic number). Thus the approximation function for $f(K)$ is $g(K) = \frac{C_1 + \frac{K}{2} + C_2 K(ln(\sqrt{\frac{K}{a_1}}) + 1)}{\sqrt{\frac{K}{a_1}}}$. The derivative of $g(K)$ is zero when K is the solution to the following equation.

$$ln(K) = \frac{d_1}{K} - d_2 \tag{9}$$

where in the equation, $d_1 = \frac{2C_1}{C_2}$ and $d_2 = \frac{1}{C_2} + 4 - ln(a_1)$. We find a numerical solution to this equation using the Newton Raphson method. The solution is the new threshold th_2, and the moving object sets the uncertainty threshold $L.uncertainty$ to a fractionally decreasing value starting with th_2.

After t_2 time units, when the deviation reaches the current uncertainty threshold, namely $\frac{th_2}{t_2}$, a location update containing th_3 is sent. th_3 is computed as above, except that a new slope (as in adr) a_2 is used; and I_2 is the cost of the deviation during the previous t_2 time units. The process continues until the end of the trip. That is, at each update time point, dtdr determines the next optimal threshold by the constants C_1, C_2, and the slope a_i of the current deviation approximation function. □

6 Querying with Uncertainty

In this section we present a probabilistic method for specifying and processing range queries about motion databases. For example, a typical query might be "Retrieve all objects o which are within the region R". Since there is an uncertainty about the location of the various objects at any time, we may not be able to answer the above query with absolute certainty. Instead, our query processing algorithm outputs a set of pairs of the form (o, p) where o is an object and p is the probability that the object is in region R at time t; actually, the algorithm retrieves only those pairs for which p is greater than some minimum value. Note that here we are using probability as a measure of certainty.

As indicated, we assume that all the objects are traveling on routes. Since the actual location is not exactly known, we assume that the location of an object o on its route at time t is a random variable [8]. We let $f_o(x)$ denote the density function of this random variable. More specifically, for small values of dx, $f_o(x)dx$ denotes the probability that o is at some point in the interval $[x, x+dx]$ at time t (actually, f_o is a function of x and t; however we omit this as t is understood from the context). The mean m_o of the above random variable is given by the database location of o (this equals $o.L.startlocation\ +\ o.L.speed(t\ -\ o.L.starttime)$; see section 2).

Now we discuss some possible candidates for the density functions f_o. Many natural processes tend to behave according to the normal density function. Let $\mathcal{N}_{m,\sigma}(x)$ denote a normal density function with mean m and standard deviation σ. We can adopt the normal density functions follows. We take the mean m to be equal to m_o given in the previous paragraph. Next we relate the standard deviation to the uncertainty of the object location. We do this by setting $\sigma = \frac{1}{c}(o.L.uncertainty)$ where $c > 0$ is constant. In this case, the probability that the object is within a distance of $o.L.uncertainty$ (i.e. within a distance of $c\sigma$) from the location m_o will be higher for higher values of c; for example, this probability will be equal to .68,.95 and .997 for values of c equal to 1,2 and 3 respectively (see [8]). The value of c is a function of the update policy used, the reliability of the network, the time since the last update and the ratio between uncertainty and deviation unit costs. Whatever may be the value c, there is still a non-zero probability p that the object is at a distance greater than $o.L.uncertainty$ from the the mean m_o; we can interpret this probability to be the probability that there is a disconnection. An alternative is to make p zero. This can be done by modifying the normal distribution to be a bounded normal distribution. This is done by conditioning that the object is within distance $o.L.uncertainty$ from m_o. More specifically, we first use the normal distribution as above by choosing an appropriate c. Then we compute the probability q that the object is within a distance of $o.L.uncertainty$ from m_o. Define the density function $f_o(x)$ to be equal to $\frac{1}{q}\mathcal{N}_{m_o,\sigma}(x)$ for values of x within the interval $[(m_o - o.L.uncertainty),\ (m_o + o.L.uncertainty)]$ and to be zero for other values of x.

A range query on motion databases is of the following form:
RETRIEVE o FROM Moving-objects WHERE C.
Here the condition part C is a disjunction of clauses where each clause is a conjunction of two conditions C_1 and C_2; C_1 only refers to the static attributes of the objects (such as 'type', 'color' etc.) and is called the static part; C_2 depends upon the location attributes and is called the dynamic part of the condition (if C is not in this form then we can convert it into disjunctive normal form; in the resulting condition each disjunct can be separated into two parts— the static part and the dynamic part). To process such a query, we process each disjunct as a separate query and take the union. Now we describe a method to process query whose condition part is a conjunction of the static and dynamic parts C_1 and C_2 respectively (one can envision other possible methods). Using the underlying database management system we execute the query whose condition part is only C_1. The set of objects thus retrieved are processed against the condition C_2 and appropriate probability values are calculated as follows.

We assume that the dynamic part of the query condition is formed by using atomic predicates $inside(o, R)$, $within\_distance(o, R, d)$ and boolean connectives $\wedge$ ("and") and $\neg$ ("not")(note that $\vee$, i.e. the "or" operator, can be defined using $\wedge$ and $\neg$). In the atomic predicate $inside(o, R)$, o is an object variable and R is the name of a region. An object o_1 satisfies this predicate at time t if its location lies within the region R. The predicate $within\_distance(o, R, d)$ is satisfied by an object o_1 traveling on route r_1 if the location of o_1 is within distance d of the region R (here the distance is measured along the route, i.e. the route distance). The following is an example query.
RETRIEVE o FROM Moving-objects WHERE $o.type =' ambulance' \wedge inside(o, R)$
Consider an object o_1 traveling on route r_1. We assume that the route r_1 intersects the region R at different places, and certain segments of the route are in the region R; each such segment is given by an interval $[u, v]$ (where $u \leq v$). For the route r_1, we let $Inside\_Int(r_1, R)$ denote the set of all such intervals. Clearly, object o_1 is in region R at time t, if its location at t lies within any of the intervals belonging to $Inside\_Int(r_1, R)$. Using the set of intervals $Inside\_Int(r_1, R)$, we can easily compute another set of intervals on route r_1 , denoted by $Within\_Int(r_1, R, d)$, such that every point belonging to any of these intervals is within distance d of region R.

Now consider a condition q formed using the above atomic predicates and using the boolean connectives. We assume that q has only one free object variable o. Now we describe a procedure for evaluation of this condition against a set of objects. The satisfaction of this condition by an object o_1 traveling on route r_1 at time t only depends on the location of the object at time t. We first compute the set of all such points. We say that a point x on the route r_1 satisfies the query q, if an object o_1 at location x satisfies q. By a simple induction on the length of q, it is easily seen that the set of points on route r_1 that satisfy q is given by a collection of disjoint intervals (if q is an atomic predicate then this is trivially the case as indicated earlier; if q is a conjunction q_1 and q_2 the resulting set of intervals for q is obtained by taking pairwise intersection of an interval belonging to that of q_1 and another belonging to that of q_2 etc.). We let $Int(r_1, q)$ denote this set of intervals.

Theorem 3: For a query q and route r_1, let $\{I_1, ..., I_i, ..., I_k\}$ be all the intervals in $Int(r_1, q)$, where $I_i = [u_i, v_i]$. Then, the probability that object o_1 traveling on route r_1 satisfies q at time t is given by

$$\sum_{i=1}^{k} \int_{u_i}^{v_i} f_{o_1}(x)dx$$

Proof: The probability that object o_1 satisfies q at time t equals the probability that the current location of o_1 lies within any of the intervals in $Int(r_1, q)$. Since all the intervals in $Int(r_1, q)$ are disjoint, it is the case that for any two distinct intervals I_i and I_j the events indicating that o_1 is inside the interval I_i (resp., inside I_j) are independent. Hence, the probability that o_1 satisfies q is equal to the sum, over all intervals I in $Int(r_1, q)$, of the probabilities that o_1 is in the interval I. For any interval $I_i = [u_i, v_i]$, the probability that the location of o_1 is in the interval I_i equals $\int_{u_i}^{v_i} f_{o_1}(x)dx$. The theorem follows from this and our earlier observations. □

A simple algorithm for computing $Int(r_1, q)$ is given below. For the route r_1, the set of intervals $Int(r_1, q)$ is computed inductively on the structure of q as follows.

q is an atomic predicate: If q is $inside(o, R)$, $Int(r_1, q)$ is the same as $Inside\_Int(r_1, R)$ and this is obtained directly from the database, possibly using a spatial indexing scheme. If q is $within\_distance(o, R, d)$ then $Int(r_1, q)$ is same as $Within\_Int(r_1, R, d)$, and this can be computed directly from $Inside\_Int(r_1, R)$. The list of intervals $Int(r_1, R)$ is output in sorted order.

$q = q_1 \wedge q_2$**:** First we compute the lists $Int(r_1, q_1)$ and $Int(r_1, q_2)$. After this, we take an interval I_1 from the first list and an interval I_2 from the second list, and output the interval $I_1 \cap I_2$ (if it is non-empty); the set of all such intervals will be the output. Since the original two lists are sorted, the above procedure can be implemented by a modified merge algorithm. The complexity of this procedure is proportional to the sum of the two input lists.

$q = \neg q_1$**:** First we compute $Int(r_1, q_1)$. We assume that the length of the route r_1 is l_1; thus the set of all points on r_1 is given by the single interval $[0, l_1]$. The set of all points on r_1 that satisfy q is the complement of the set of points that satisfy q_1 where this complement is taken with respect to all the points on the route; clearly, this set of points is a collection of disjoint intervals. Now, it is fairly straightforward to see how the sorted list of intervals in $Int(r_1, q)$ can be computed from $Int(r_1, q_1)$; the complexity of such a procedure is simply linear in the number of intervals in $Int(r_1, q_1)$.

If $L_1, L_2, ..., L_k$ are the lists of intervals corresponding to the atomic predicates appearing in q and l is the sum of the lengths of these lists, and m is the length of q then it can be shown that the complexity of the above procedure is $O(lm)$.

Now consider the query
RETRIEVE o FROM Moving-objects WHERE $C_1 \wedge C_2$
where C_1, C_2, respectively, are the static and the dynamic parts of the condition.

The overall algorithm for processing the query is as follows.

1. Using the underlying database process the following query.
   RETRIEVE o FROM Moving-objects WHERE C_1.
   Let O be the set of objects retrieved.

2. Using the underlying database retrieve the set of routes R on which the objects in O are traveling.

3. For each atomic predicate p appearing in C_2 and for each route r_1 in R, retrieve the list of intervals $Int(r_1, p)$. This is achieved by using any spatial indexing scheme.

4. Using the algorithm presented earlier, for each route r_1, compute the list of intervals $Int(r_1, q)$.

5. For each route r_1 and for each object o_1 traveling on r_1, compute the probability that it satisfies q using the formula given in theorem 3.

7 Relevant Work

One research area to which this paper is related is uncertainty and incomplete information in databases (see for example [18, 1] for surveys). However, as far as we know this area has so far addressed complementary issues to the ones in this paper. Our current work on location update policies addresses the question: what uncertainty to initially associate with the location of each moving object. In contrast, existing works are concerned with management and reasoning with uncertainty, after such uncertainty is introduced in the database. Our probabilistic query processing approach is also concerned with this problem. However, our uncertainty processing problem is combined with a temporal-spatial aspect that has not been studied previously as far as we know; for treatment of the spatial issue alone see [9].

Our problem is also related to mobile computing, particularly works on location management in the cellular architecture. These works address the following problem. When calling or sending a message to a mobile user, the network infrastructure must locate the cell in which the user is currently located. The network uses the location database that gives the current cell of each mobile user. The record is updated when the user moves from one cell to another, and it is read when the user is called. Existing works on location management (see, for example, [25, 10, 4, 15, 14, 2, 17]) address the problem of allocating and distributing the location database such that the lookup time and update overhead are minimized. Location management in the cellular architecture can be viewed as addressing the problem of providing uncertainty bounds for each mobile user. The geographic bounds of the cell constitute the uncertainty bounds for the user. Uncertainty at the cell-granularity is sufficient for the purpose of calling a mobile user or sending him/her a message. When it is also sufficient for MOD applications, the location database can be sold by wireless communication vendors to mobile fleet operators. However, often uncertainty at the cell granularity is insufficient. For example, in satellite networks the diameter of a cell ranges from hundreds to thousands of miles.

Another relevant research area is constraint databases (see [16] for a survey and [6, 13] for some notable systems). In this sense, our location attributes can be viewed as a constraint, or a generalized tuple, such that the tuples satisfying the constraint are considered to be in the database. Constraint databases have been separately applied to the temporal (see [5, 7, 3]) domain, and to the spatial domain (see [20]). Constraint databases can be used as a framework in which to implement the proposed update policies and query processing algorithm. Since our main problem is communication tradeoffs in replicating the position information, data replication (e.g. [26]) is also somewhat related to our current research.

Finally, the present paper extends the work on which we initially reported in [11, 12] in two important ways. First, in this paper we introduce a quantitative new probabilistic model and method of processing range queries. In contrast, in previous works we took a qualitative approach in the form of "may" and "must" semantics of queries. Second, in this paper we introduce uncertainty as a separate concept from deviation. The previous work on update policies (i.e. [12]) is not equipped to distinguish between uncertainty and deviation. Consequently, The location update policies discussed in this paper are different in two respects from the update policies in [12]. First, they take uncertainty into consideration when determining when to send a location update message. Second they are dead reckoning policies; namely they provide the uncertainty, i.e. the bound on the deviation, with each location update message. In contrast, the [12] policies are not dead reckoning in the sense that the moving object does not update its location when the deviation reaches some threshold; the update time-point depends on the overall behavior of the deviation since the last update. Our simulation results reported in the appendix indicate that the [12] policies are inferior to adr (and often to dtdr as well) when the uncertainty cost is taken into consideration, and this inferiority increases as the cost per unit of uncertainty increases.

8 Conclusion and future work

In this paper we considered dead-reckoning policies for updating the database location of moving objects, and the processing of range queries for motion database. When using a dead-reckoning policy, a moving object equipped with a Geographic Positioning System periodically sends an update of its database location and provides an uncertainty threshold th. The threshold indicates that the object will send another update when the deviation, namely the distance between the database location and the actual location, exceeds th.

Dead-reckoning policies imply that the DBMS answers a query about the location of an object m by: "the current location of m is (x,y) with a deviation of at most th". When making decisions based on such an answer, there is a cost in terms of the deviation of m from (x,y), and in terms of the uncertainty about its location. These costs should be balanced against the cost (in terms of wireless bandwidth, and update processing) of sending location update messages. We introduced a cost model that captures the tradeoffs between communication, uncertainty and deviation by assigning costs to an uncertainty unit, a deviation unit, and a communication unit. We explained that these costs should be determined by answering questions such as: how many messages is the system willing to utilize in order to reduce the deviation by one unit during a unit of time? Is a unit of uncertainty more important than a unit of deviation, or vice versa?

Then we introduced two dead-reckoning policies, adaptive dead-reckoning (adr), and disconnection detection dead-reckoning (dtdr). Both adjust the uncertainty threshold at each update to the current motion (or speed) pattern. This pattern is captured by the concept of the predicted deviation. The difference between the two policies is that dtdr uses a novel technique for disconnection detection in mobile computing, namely decreasing uncertainty threshold. Intuitively, the technique postulates that the probability of communication should increase as the period of time since the last communication increases. Thus, the probability of the object being disconnected increases as the period of time since the last update increases. Dtdr demonstrates the use of this technique.

Then we reported on the development of a simulation testbed for evaluation of location update policies. We used it in order the compare the information cost of adr, dtdr, and speed dead-reckoning (sdr) in which the uncertainty threshold is arbitrary and fixed. The result of the comparison is that adr is superior to the other policies in the sense that it has a lower information cost. Actually, it may have an information cost which is six times lower than that of sdr. We quantified the disconnection detection cost as the difference between the cost of dtdr and that of adr. We also determined that when taking uncertainty into consideration, the information costs of adr and dtdr are lower than that of non-dead-reckoning policies which we developed previously.

Finally, an additional contribution of this paper is a probabilistic model and an algorithm for query processing in motion databases. In our model the location of the moving object is a random variable, and at any point in time the database location and the uncertainty are used to determine a density function for this variable. Then we developed an algorithm that processes range queries such as 'retrieve the moving objects that are currently inside a given polygon P'. The answer is a set of objects, each of which is associated with the probability that currently the object is inside P.

Now consider the following variant of the location update problem. In some cases MOD applications may not be interested in the *location* of moving objects at any point in time, but in their arrival time at the destination. Assume that the database arrival information is given by "The object is estimated to arrive at destination X at time t, with an uncertainty of U". In other words, t is the database estimated-arrival-time[9] (eat) and we assume that at any point in time before arrival at destination X, the moving object can compute the actual eat[10], t'. The difference between t and t' is the deviation, and the uncertainty U denotes the bound on the deviation of the eat; the object will send an eat update message when the deviation reaches U. In this variant, the motion database update problem is to determine when a moving object should update its database estimated-arrival-time. The results that we developed in this paper for the location update problem carry over verbatim to the eat update problem. Actually, the location update problem is more general in the sense that the DBMS holds the estimated arrival time at any future location, not just the final destination.

Concerning future work, we believe that moving objects databases will become increasingly important, and we believe that DBMS's should be made a platform for developing moving-objects applications. For this purpose, as mentioned in the introduction, much remains to be done in terms of spatio-temporal query languages, support for rapidly changing real-time data, indexing, and imprecision.

Another issue is to extend the present work to handle uncertainty for moving objects that do not report their location; instead their location is sensed by possibly unreliable means. This is the case, for example, for enemy forces in a battlefield.

References

[1] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison Wesley, 1995.

[2] R. Alonso, H. F. Korth, *Database System Issues in Nomadic Computing*, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington DC, May 1993.

[3] M. Abadi and Z. Manna, *Temporal logic programming*, Journal of Symbolic Computation, 8, 1989.

[4] B. R. Badrinath, T. Imielinski, A. Virmani, *Locating Strategies for Personal Communication Networks*, Workshop on Networking for Personal Communications Applications, IEEE GLOBECOM, December 1992.

[5] M. Baudinet, M. Niezette, P. Wolper, *On the representation of infinite data and queries*, ACM Symposium on Principles of Database Systems, May 1991.

[6] A. Brodsky, V. E. Segal, J. Chen, and R. A. Exarkhopoulo, The CCUBE Constraint Object-Oriented Database System, manuscript, 1997.

[7] J. Chomicki and T. Imielinski, *Temporal deductive databases and infinite objects*, ACM Symposium on Principles of Database Systems, March 1988.

[8] W. Feller, *An Introduction to Probability Theory*, John Wiley and Sons, 1966

[9] M. Goodchild and S. Gopal, editors, *Accuracy of Spatial Databases*, Taylor and Francis, 1989

[10] J. S. M. Ho, I. F. Akyildiz, *Local Anchor Scheme for Reducing Location Tracking Costs in PCN*, 1st ACM International Conference on Mobile Computing and Networking (MOBICOM'95), Berkeley, California, November 1995.

[11] P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, *Modeling and Querying Moving Objects*, Proceedings of the Thirteenth International Conference on Data Engineering (ICDE13), Birmingham, UK, Apr. 1997.

[12] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, G. Mendez, *Cost and Imprecision in Modeling the Position of Moving Objects,* Proceedings of the Fourteenth International Conference on Data Engineering (ICDE14), 1998

[9] equivalent to the database location

[10] equivalent to the actual location

[13] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin, DEDALE, A Spatial Constraint Database, manuscript, 1997.

[14] T. Imielinski and H. Korth, *Mobile Computing*, Kluwer Academic Publishers, 1996.

[15] R. Jain, Y.-B. Lin, C. Lo, and S. Mohan, *A Caching Strategy to Reduce Network Impacts of PCS*, IEEE Journal on Selected Areas in Communications, Vol(12), October 1994.

[16] P. Kanellakis, *Constraint programming and database languages*, ACM Symposium on Principles of Database Systems, May 1995.

[17] Lazoff, B. Stephens, Y. Yesha, *Optimal Location of Broadcast Sources in Unreliable Tree Networks*, IEEE International Conference on Computers and Communications Networks, Rockville, Maryland, 1996.

[18] A. Motro, *Management of Uncertainty in Database Systems*, In Modern Database Systems, Won Kim ed., Addison Wesley, 1995.

[19] OmniTRACS, *Communicating Without Limits* , http://www.qualcomm.com/ProdTech/Omni/prodtech/omnisys.html.

[20] J. Paradaens, J. van den Bussche, D. Van Gucht, *Towards a theory of spatial database queries*, ACM Symposium on Principles of Database Systems, May 1994.

[21] R. Snodgrass and I. Ahn, *The temporal databases*, IEEE Computer, Sept. 1986.

[22] S. Chamberlain, *Model-Based Battle Command: A Paradigm Whose Time Has Come,* 1995 Symposium on C2 Research & Technology, NDU, June 1995

[23] S. Chamberlain, *Automated Information Distribution in Bandwidth-Constrained Environments* MILCOM-94 conference, 1994.

[24] S.D. Silvey, *Statistical Inference*, Chapman and Hall, 1975

[25] N. Shivakumar, J. Jannink and J. Widom, *Per-User Profile Replication in Mobile Environments: Algorithms, Analysis, and Simulation Results*, ACM/Baltzer Journal on Special Topics in Mobile Networks and Applications, special issue on Data Management, 1997.

[26] Y. Yesha, K. Humenik, P. Matthews, B. Stephens, *Minimizing Message Complexity in Partially Replicated Databases on Hypercube Networks*, accepted for publication in Networks

9 Appendix A: Comparison of the update policies by simulation

In this section we first discuss the simulation method and then the simulation results.

9.1 Simulation method

The objective of the simulations is to compare the information cost of sdr, adr and dtdr policies. As parameters to the simulation we use ten moving objects, each taking a two-hour trip. Each trip is represented by a speed curve, i.e. the actual-speed of a moving object as a function of time. In figures 1 and 2 at the end of the paper we show two typical speed curves. [11] The average distance traveled during a trip is 82 miles. For each speed curve, update policy, update cost C_1, and uncertainty unit cost C_2 we execute a simulation run. The run computes the information cost [12] (a single number) of the policy on the curve. (Observe that for the purpose of computing this information cost the actual route taken by the moving object is irrelevant; the deviation and the information cost at each point in time can be computed using only the speed curve.) Then, for each policy, we average the information cost over all the speed curves, and plot this average as a function of the update cost C_1.

Each simulation run is executed as follows. A speed-curve is a sequence S of actual speeds, one for each time unit. In our simulations a time unit is 10 seconds. Using S we simulate the moving object's computer working

[11] The speed curves were generated using an approximation of the speed during a trip on highways and local streets in the Chicago area, for various times of day (e.g. rush hour, night, etc.)

[12] Remember, the *information cost* of an update policy on a given speed-curve is computed by using equation 3 for every time interval between two consecutive update points.

with a particular update policy. This is done as follows. For each time unit there is an uncertainty threshold th, as well as a database speed and an actual speed. The deviation at a particular point in time t is the difference between the integral of the actual-speed as a function of time, and the integral of the database-speed (the integrals are taken from the last update until t). Denote by T the sequence of deviations, one at each time unit. Denote by Q the sequence of uncertainty thresholds, one at each time unit. If the deviation at time t reaches the threshold, then we generate an update record consisting of: the current time, the current location, the current speed, the next threshold (for adr and dtdr it is computed as explained in section 5); the deviation at time t becomes zero. Denote by U the sequence of update records. Using T we compute the total cost of the deviation, denoted c_1, and using U we compute the total cost of the updates, c_2. Using Q we compute the total cost of the uncertainty, c_3. The information cost of the policy on the speed curve is $c_1 + c_2 + c_3$.

9.2 Simulation results

We compared the information cost for the adr, sdr, dtdr, and il (see [12]) policies. Figures 7-18 plot the results of the comparison. Each figure compares il, adr, dtdr and sdr(X), i.e, sdr with threshold X, where X = 0.2, 1.0, 2.5 or 7.5. Each figure uses a particular value of C_2. The C_2 values that we consider are 0.25, 0.75, and 3.00 (when $C_2 = 3$ it means that the cost of a unit of uncertainty is three times as high as the cost of a unit of deviation). When adr and dtdr are compared with sdr(X), the first (i.e. initial) uncertainty threshold of adr and dtdr is taken to be X; the following thresholds are determined dynamically, as explained in section 5. The parameter X is irrelevant for the il policy because il is not a dead-reckoning policy. Each figure compares the information cost of the four policies. Each curve in figures 7-18 plots the information cost of a policy as a function of the update cost C_1.

For information about each cost component, figures 3-6 plot the information cost, the cost of updates, the cost of uncertainty, and the cost of deviation for the parameter setting X=2.5 and $C_2 = 0.75$. Observe that since the number of messages used by sdr is independent of the cost of a message, the uncertainty cost of sdr is independent of the cost of a message. The same holds for the deviation cost. On the other hand, the other policies adapt the number of messages to the message cost. Thus, they use less messages as the cost of a message increases, and consequently, for them, the deviation and uncertainty costs increase as the update cost increases. Observe that although the cost of a unit of uncertainty is lower than the cost of a unit of deviation (0.75 versus 1.00), uncertainty is the highest cost component for il (which is not true for the other policies) and it is higher than the uncertainty cost of the other policies.

Figures 7-18 are arranged such that each column gives the information cost for a particular uncertainty threshold X, for the three values of C_2. Thus for example figures 7, 8, 9 give the information cost for X=2.5. The basic conclusion from the simulations is that for almost all the experiments, the adr policy is superior to the other policies. The information cost of sdr is often several times higher than that of adr. One exception is the extreme threshold value X= 7.5, where the costs of uncertainty and deviation between the beginning of the trip and the next update dominate the total information cost of adr; this is an anomaly due to the unnecessarily high initial threshold. Concerning il, observe that since it cannot account for the cost of uncertainty, its total cost increases as the cost of uncertainty increases. Even for small values of C_2 it is not better than adr (except for the anomaluos threshold X=7.5).

Observe that the information cost curves of the adr policy are almost identical for all the initial uncertainty thresholds. The same holds for the dtdr and il policies. This indicates that these policies exhibit a stable behavior which is independent of initial conditions.

Figure 1 Speed Curve for A Two Hour Trip

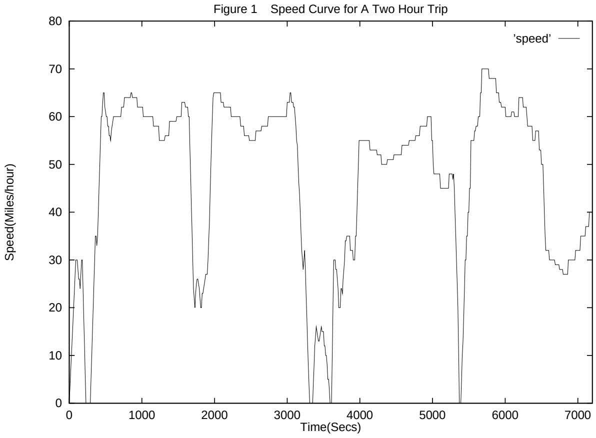


Figure 2 Speed Curve for A Two Hour Trip

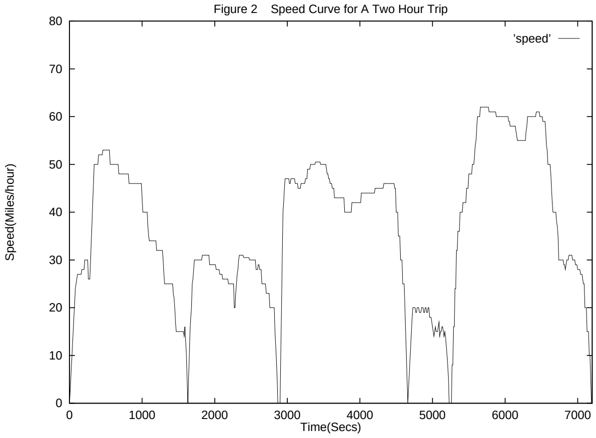


Figure 3 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 2.5, UncerCost=0.75

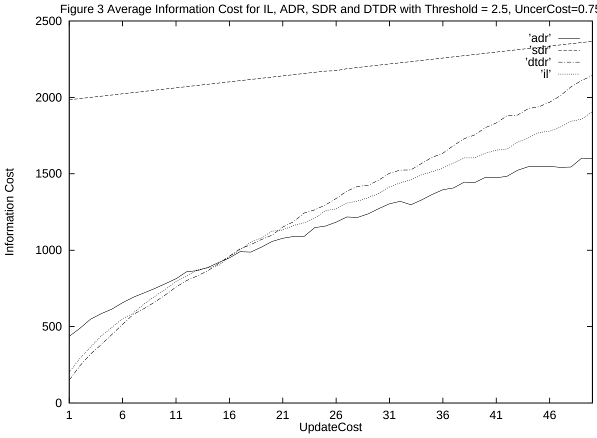


Figure 5 Average Cost of Update Messages for IL, ADR ,SDR and DTDR with Threshold = 2.5, UncerCost=0.75

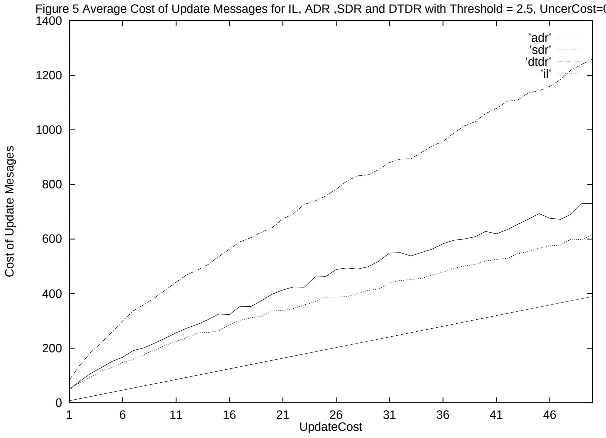


Figure 4 Average Cost of Uncertainty for IL, ADR, SDR and DTDR with Threshold = 2.5, UncerCost=0.75

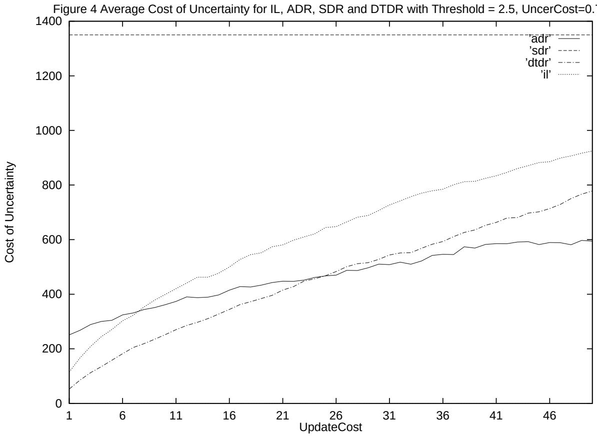


Figure 6 Average Cost of Deviation for IL, ADR, SDR and DTDR with Threshold = 2.5, UncerCost=0.75

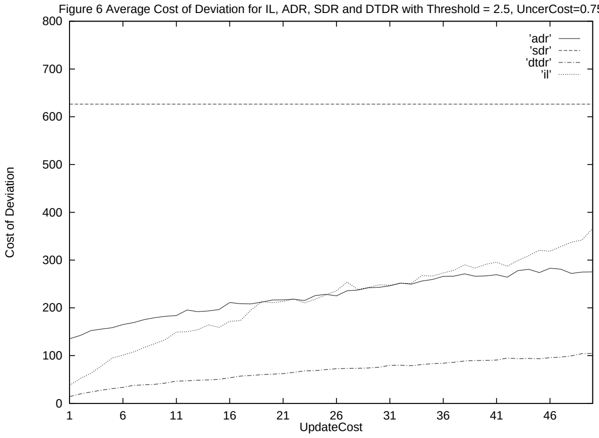


Figure 7 Average Information Cost for IL, ADR, SDR and DTDR with Initial Threshold = 0.2, UncerCost=0.25

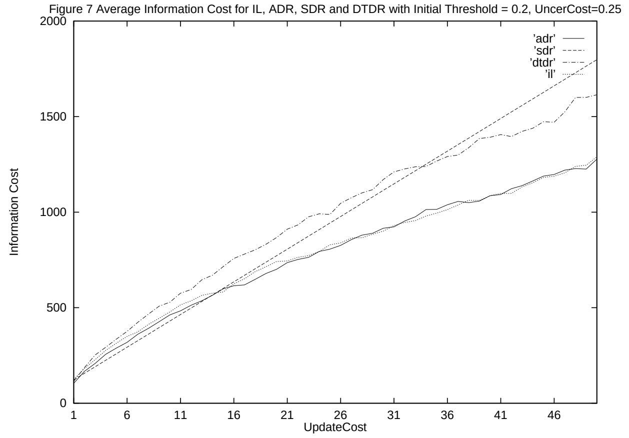


Figure 8 Average Information Cost for IL, ADR, SDR and DTDR with Initial Threshold = 0.2, UncerCost=0.75

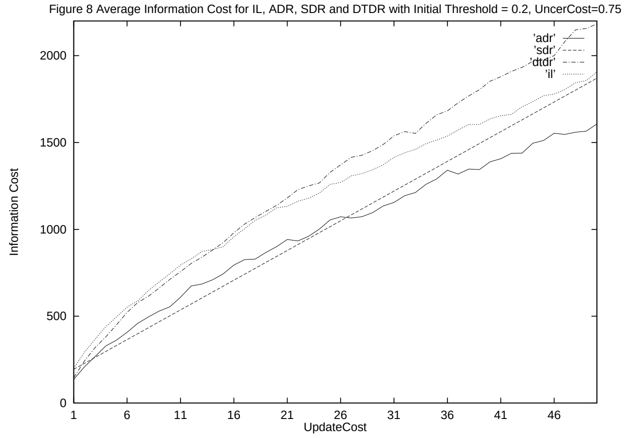


Figure 9 Average Information Cost for IL, ADR, SDR and DTDR with Initial Threshold = 0.2, UncerCost=3.00

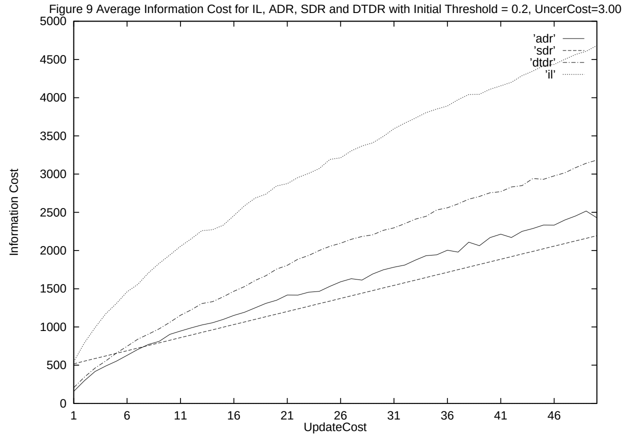


Figure 10 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 1.0, UncerCost=0.25

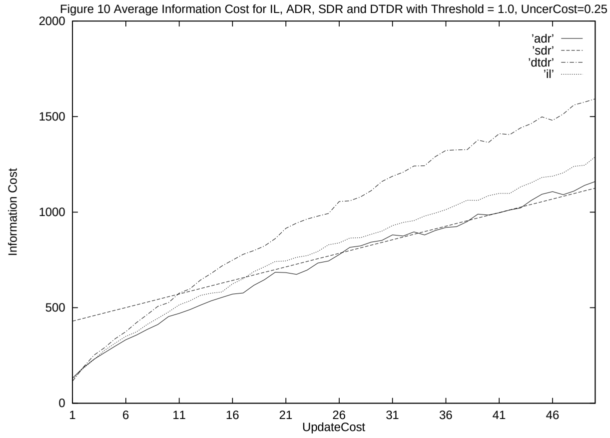


Figure 11 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 1.0, UncerCost=0.75

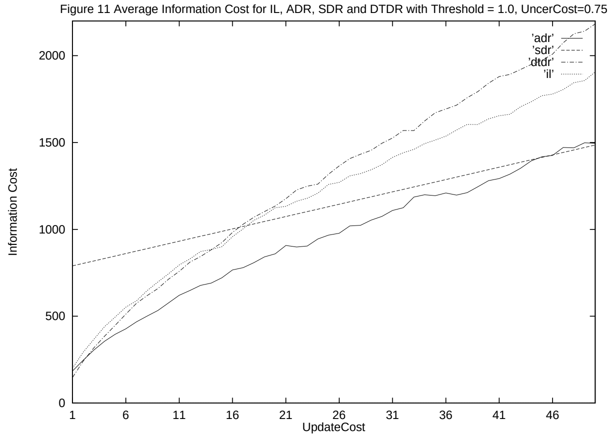


Figure 12 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 1.0, UncerCost=3.00

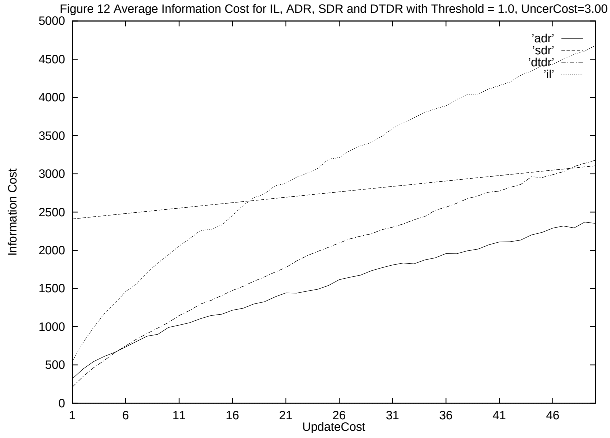


Figure 13 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 2.5, UncerCost=0.25

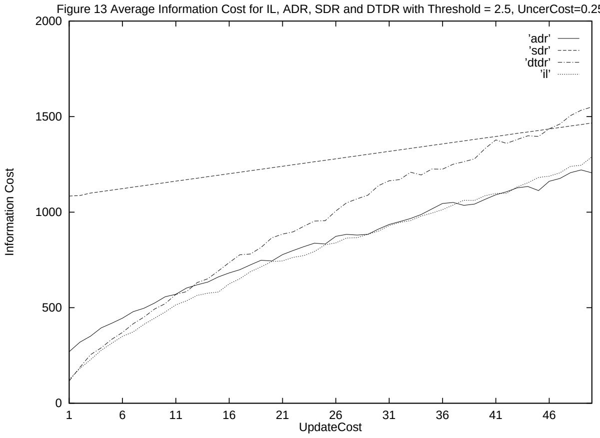


Figure 14 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 2.5, UncerCost=0.75

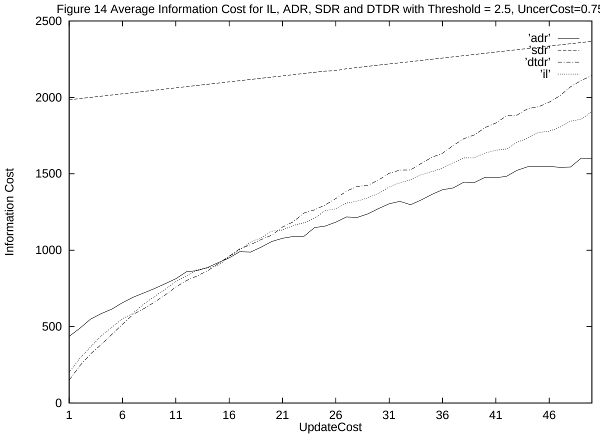


Figure 15 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 2.5, UncerCost=3.00

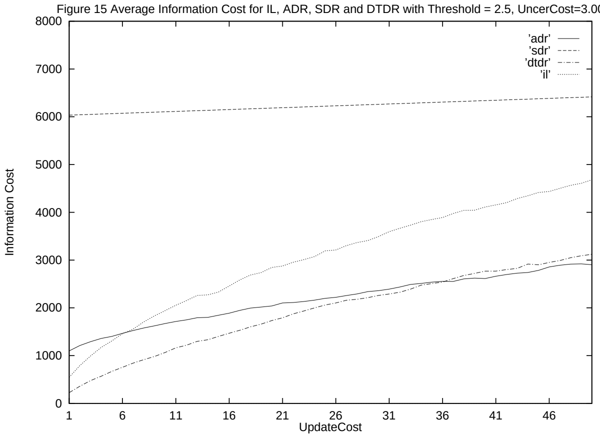


Figure 16 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 7.5, UncerCost=0.25

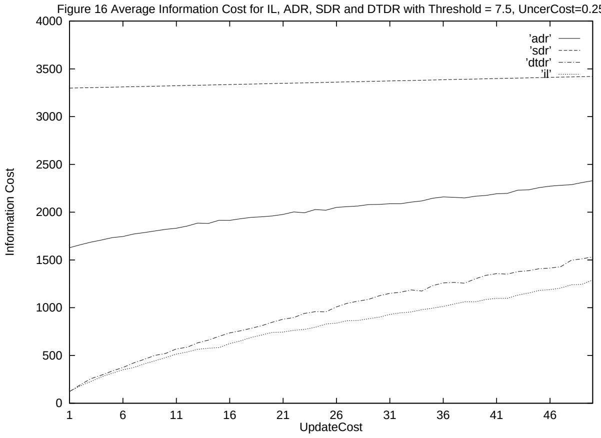


Figure 17 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 7.5, UncerCost=0.75

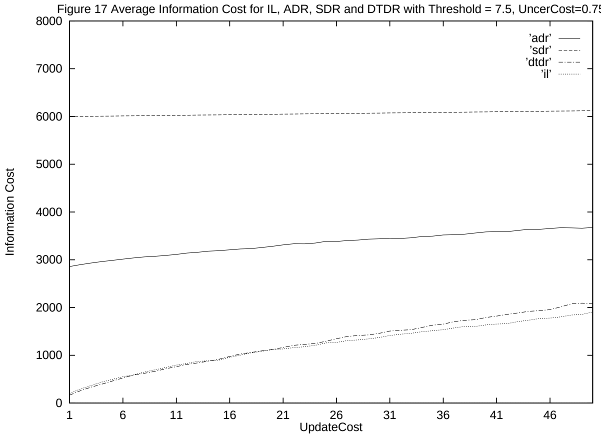
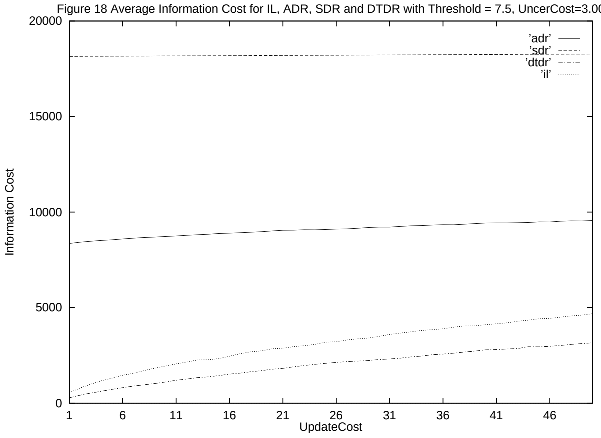


Figure 18 Average Information Cost for IL, ADR, SDR and DTDR with Threshold = 7.5, UncerCost=3.00



10 Appendix B: Demonstration of the adr and dtdr update policies

Demonstration of adr by an example: At the beginning of the trip the moving object m sends a location update message giving the route, its location on the route, $L.speed = 0.2$ (miles/minute), and $L.uncertainty = 0.5$ (miles). Suppose that after 4 minutes the deviation reaches the threshold 0.2. At that time m sends a location update message containing its current location on the route, its current speed, and a new value for $L.uncertainty$. Suppose that the integral of the deviation from the beginning of the trip (time 0) to time 4 (minutes) is 1, and $C_1 = 8$, and $C_2 = 1$. Then the new value for $L.uncertainty$ is 0.82. Suppose that after 10 more minutes the deviation reaches this new threshold, and at that time the integral of the deviation from time 4 to time 14 is 1.5. Then the new value for $L.uncertainty$ is 0.4. □

Demonstration of dtdr by an example: At the beginning of the trip, the moving object m sends a location update message giving the route, its location on the route, $L.speed = 0.2$ (miles/minute), and $L.uncertainty = 0.5$ (miles). A time unit is 1 minute. Then for the first minute the threshold is 0.5, and for the second minute the threshold changes to 0.25 (miles). Suppose that during the second minute of the trip, the deviation reaches the current threshold 0.25. At that time m sends a location update message containing its current location on the route, its current speed, and a new initial value for $L.uncertainty$. That value is computed as follows. Suppose that the integral of the deviation from the beginning of the trip (time 0) to time 2 is 0.5, and $C_1 = 8$, $C_2 = 1$. Then the slope of deviation estimator is $a = \frac{2*0.5}{2^2} = 0.25$, and the new value for $L.uncertainty$ is 2.226 (it is computed by a numerical solution to the equation 9). Suppose that after 5 more minutes, the deviation reaches the current threshold which is $\frac{2.226}{5} = 0.445$, and at that time the integral of the deviation from time 2 to time 7 is 1.2. Then the slope of deviation estimator is $a = \frac{2*1.2}{5^2} = 0.096$, and the new initial $L.uncertainty$ is 1.992. □