Location Management in Moving Objects Databases *

Ouri Wolfson[†], Sam Chamberlain[‡], Son Dao[§], Liqin Jiang[†]

1 Abstract

In this paper we first introduce Moving Objects Databases and their related research problems. Then we concentrate on a particular problem, namely reducing the information cost associated with a trip taken by some moving object (e.g. a vehicle). The information cost of a trip consists of the overhead of position-update messages and the deviation of the database position from the actual position of the object. We introduce two position update policies, namely plain dead reckoning (pdr) and adaptive dead reckoning (adr). We show that adr has a lower information cost than pdr.

2 Introduction

2.1 Moving Objects Databases and Relevant Research

Consider a database that represents information about moving objects and their position. 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 position 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 moving-objects-database (MOD) applications.

*This research was supported in part by NSF grant IRI-9224605, NSF grant IRI-9408750, DARPA grant N66001-97-2-8901, NATO grant CRG-960648

[†]Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60607

[‡]Army Research Laboratory, Aberdeen Proving Ground, MD

[§]Hughes Research Laboratories, Malibu, CA

Database management system (DBMS) technology provides a foundation for efficiently answering queries about moving objects. However, 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 objective of our Databases fOr MovINg Objects (DOMINO) project is to build an envelope containing these capabilities on top of existing DBMS's.

The added capabilities include, among other things, support for spatial and temporal information. For example, "retrieve the friendly helicopters that are expected (according to current speed and direction information) to be WITHIN polygon P DURING the next 10 minutes" is a spatio-temporal query. It requires an intuitive query language, that includes spatial operators (e.g. point WITHIN polygon) and temporal operators (e.g. DURING); it also requires indexing and efficient processing algorithms. Although temporal databases ([12, 10, 11, 14, 15, 13]), and spatial databases (see [8] for a survey) have been studied in the literature, the integrated application of spatial and temporal databases for the purpose of modeling moving objects has not been considered, although we believe that it is important.

The problem is further complicated because the location of a moving object is inherently uncertain, i.e. the position reflected in the database at a particular time will usually not be identical to the actual position of the moving object at the time. Again, although uncertainty and incomplete information in databases has been studied before (see [6, 1] for surveys), none of the approaches provide a satisfactory solution to our problem. For example, consider the query Q: "retrieve the friendly helicopters that are expected (according to current speed and direction information) to enter the polygon P within the next 10 minutes". Assume that there exist some a priori bounds on the deviation of the database position from the actual position, namely *uncertainty bounds*. For example, if an uncertainty bound is x it means that the actual position must be within a distance of x from the database position. We'd like to be able to enter the query Q with either, "must" or "maybe" semantics. In the former case the database retrieves the objects that, under deviation bounds assumption, must in P within the next 10 minutes; and in the latter, the ones that maybe in P within the next 10 minutes.

The next problem is how to set the uncertainty bounds for a moving object. The problem is related to works on location management in the cellular architecture. Namely, 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 location database is the 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, [9, 3, 2, 5, 4]) 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. **In satellite networks the diameter of a cell ranges from hundreds to thousands of miles. This makes the location-uncertainty at cell-granularity much too high for MOD applications and queries as discussed above.**

Consequently, in satellite networks and in environments that are not covered by the cellular architecture each moving object has to send to the location-database position update messages [1], i.e. messages that update the object-position in the database. For the purpose of resource management, position update messages may be beneficial even in terrestrial cellular networks with small cells. The reason is that the network can better plan bandwidth allocation if each base station knows the location and motion of each mobile unit in its cell. For example, if the network knows that a user u is on the fringe of cell A, moving to adjacent cell B, then it can reserve bandwidth for u at cell B.

[1] We assume that at any point in time each vehicle knows its exact current position, using, for example, an onboard Geographic Positioning System (GPS). The present work is also applicable to the case where the moving object does not have a GPS, but there is some sensor that can compute the current distance between the moving object's position recorded in the database and its actual position.

2.2 Position update policies

Frequent position-updating may be expensive in terms of performance and uplink wireless-bandwidth overhead. **This paper is concerned with the overhead of position update messages.** In order to drastically reduce the update cost, we model the current position of a moving object as the distance from its starting position, along a given route. The distance continuously increases as a function of time, without being updated. So, for example, the DBMS knows that the moving object m started at 5pm at position (x_0, y_0) on a given route known to the DBMS, and it travels at 60 miles/hour; thus, at any point in time after 5pm, in response to a query, the DBMS can easily compute the current position of m. Our simulation experiments show that, even when the speed fluctuates sharply, this technique reduces the number of updates to 15% of the number used by the traditional, nontemporal method (in this method the database simply stores the latest known position for each object); this saves 85% of the position-updates overhead.

Assuming that the actual position of m deviates from the position computed by the DBMS (namely the database position) due to the fact that m does not travel continuously at **exactly** 60 mi/hr, an *update policy* for m dictates how frequently m should update its position in the database in order to eliminate the deviation. A simple update policy is called in the military "plain dead-reckoning (pdr)". The moving object provides a threshold th, and commits to update its position whenever the deviation, i.e. the difference between the actual position and the database position, exceeds th. [2] The problem with pdr is that the threshold th should vary depending on the update cost (note that the update cost may vary over time depending on the demand for bandwidth) and the expected behavior of the deviation.

To address this problem we introduce another update policy, "adaptive dead reckoning (adr)". It provides at each update a new threshold th that is computed using a cost based approach. th minimizes the total of the update cost and the deviation cost, under reasonable assumptions about the future behavior of the deviation.

We compare by simulation the adr and pdr policies. Our simulations indicate that adr is superior to pdr in the sense that it has a lower information cost, i.e. total cost of updates and deviation.

The rest of this paper is organized as follows. In

[2] Note that at any point in time, since the moving object knows its actual position and the starting point and speed stored in the database, it can compute its current deviation.

section 3 we discuss position attributes of moving objects, and in section 4 we discuss the information cost of a trip. In section 5 we introduce the two position update policies, and in section 6 we discuss the results of their comparison by simulation. In section 7 we summarize the paper.

3 Position attributes

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.

Point object classes are either mobile or stationary. A point object class O has a *position attribute* P. If the object class is *stationary*, its position attribute has two sub-attributes *P.x*, and *P.y*, representing the x and y coordinates of the object. If the object class is *mobile*, its position attribute has seven sub-attributes, *P.starttime*, *P.route*, *P.x.startposition*, *P.y.startposition*, *P.direction*, *P.speed*, and *P.bound*.

The semantics of the sub-attributes are as follows. *P.route* is (the pointer to) a line spatial object indicating the route on which an object in the class O is moving. [3] *P.x.startposition* and *P.y.startposition* are the x and y coordinates of a point on *P.route*; it is the position of the moving object at time *P.starttime*. In other words, *P.starttime* is the time when the moving object was at position (*P.x.startposition*, *P.y.startposition*). We assume that whenever a moving object updates its P attribute it updates the *P.x.startposition* and *P.y.startposition* subattributes; thus *P.starttime* is also the time of the last position-update. We assume in this paper that the database updates are instantaneous, i.e. valid- and transaction- times (see [7]) are equal. Therefore, *P.starttime* is the time at which the update occurred in the real world system being modeled, and the time when the database installed the update. *P.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). *P.speed* is a linear function of the form $f(t) = b \cdot t$. It is defined by the speed b of the moving object, and it gives the current distance from the starting position as a function of the time t elapsed since the last update. *P.bound* is the threshold on the position deviation (the deviation is formally defined at the end of this section); when the deviation reaches the threshold, the moving object sends a position update message.

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. This is the case, for example, if the route is given by a piece-wise linear function. The *database position* of a moving object at a given point in time is defined as follows. At time *P.starttime* the database position is the pair (*P.x.startposition*, *P.y.startposition*); the database position at time $A.starttime + t_0$ is the point on the route which is at route-distance $P.speed \cdot t_0$ from the point with coordinates (*P.x.startposition*, *P.y.startposition*). Intuitively, the database position of a moving object at a given point in time t is the position of the object as far as the DBMS knows; it is the position that is returned by the DBMS in response to a query entered at time t that retrieves the object's position.

We assume that at the beginning of the trip the moving object writes all the sub-attributes of the position attribute. Subsequently, the moving object periodically updates its current position and speed stored in the database. Specifically, a *position update* is an update message sent by the moving object to the database; it consists of values for at least the sub-attributes *P.starttime*, *P.speed*, *P.x.startposition* and *P.y.startposition*.

Since between two consecutive position updates the moving object does not travel at exactly the speed *P.speed*, the actual position of the moving object deviates from its database position. 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 position at time t and its database position at time t. The deviation is always nonnegative. At any point in time the moving object knows its current position, and it knows the parameters of the last position-update. Therefore at any point in time the (computer onboard the) moving object can compute the current deviation.

4 The information cost of a trip

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 nonnega-

[3] For simplicity, our discussion pertains to routes in 2-dimensional space, but our concepts and results can be extended to routes in 3-dimensional space.

tive number. For example, suppose that one (1) is the penalty for each unit of deviation reported in response to a query that retrieves the position of m; and on average, there is one such query per time unit. Then, the cost of a unit of deviation per unit of time is one, and the cost of the deviation between two time points t_1 and t_2 is:

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

We call the function of equation 1 the *uniform* deviation cost function.

In this paper we consider only update policies that have the uniform deviation cost function. However, there exist other deviation cost functions. For example, the step deviation cost function carries a zero-penalty for each time unit in which the deviation stays below some threshold h, and a penalty of one otherwise.

The *update cost*, denoted C, is a nonnegative number representing the cost of a position-update message sent from the moving object to the database. 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, for the uniform deviation cost function, C is the ratio between the update cost, and the cost of a unit of deviation per unit of time. For example, the cost of a wireless message using one of the wireless data transmission services (e.g. RAM mobile data Co. or ARDIS Co.) is 5 cents. Thus, if the cost of a unit of deviation per unit of time is one cent, then $C = 5$.

Another way of interpreting C is the following. If $1/C$ is the number of messages that the moving object is willing to use in order to reduce the deviation by one during one unit of time, then the cost of a message is C.

Let t_1 and t_2 be the time-stamps of two consecutive position update messages. Then the position update policy takes the information cost in the interval $[t_1, t_2)$ to be:

$$COST[t_1, t_2) = C + COST_d(t_1, t_2) \qquad (2)$$

Observe that $COST[t_1, t_2)$ includes the message cost at time t_1 but not at time t_2. Observe also that each position update message writes the actual current position in the database, thus it reduces the deviation to zero. The *information cost* of a trip is the information cost of the deviation plus the position update messages during the trip. It is computed by summing up $COST[t_1, t_2)$ for every pair of consecutive update points t_1 and t_2 (the end of the trip is considered to be an update point).

5 The adr and pdr policies

When using the plain dead-reckoning (pdr) policy, the moving object has a predefined threshold th, and it updates the database whenever the deviation exceeds th. The update simply includes the current position and current speed.

A second policy that we consider in this paper is adaptive dead reckoning (adr). Next we describe the adr policy, and then we'll give the mathematical motivation for it. At the beginning, the moving object declares to the DBMS an initial deviation threshold th_1, and an initial database speed. Then it starts tracking the deviation. When the deviation reaches th_1, the moving object sends an update to the database. The update consists of the current speed, current position, and a new threshold th_2 to be installed in the *P.bound* 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 that same time interval, and let $a_1 = \frac{2I_1}{t_1^2}$. Then th_2 is $\sqrt{2a_1C}$ (remember, C is the update cost). When the deviation reaches th_2, a similar update is sent, except that the new threshold th_3 is $\sqrt{2a_2C}$, where $a_2 = \frac{2I_2}{t_2^2}$ (I_2 is the cost of the deviation from the first update, t_2 is the time elapsed since the first position 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 the following proposition.

Proposition 1: Assume that two consecutive position updates occur at times 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. Denote the update cost by C. Then, for the uniform deviation cost function, the information cost per time unit between t_1 and t_2 is minimized if $t_2 = t_1 + \sqrt{\frac{2a}{C}}$.

Proof: Assume without loss of generality that the last database update occurred at time t_1. Then, based on equations 1 and 2, for $t > t_1$:

$$COST[t_1, t) = C + \int_{t_1}^{t} a(t - t_1)dt = C + \frac{a(t - t_1)^2}{2} \quad (3)$$

Denote by $f(t_2)$ the information cost per time unit between t_1 and t_2, for the update time t_2. Namely, $f(t_2) = \frac{COST[t_2 - t_1)}{(t_2 - t_1)}$. It is easy to see that $f(t_2) = \frac{2C + a(t_2 - t_1)^2}{2(t_2 - t_1)}$. Using the derivative it is easy to calculate that the minimum of $f(t_2)$ is obtained when $t_2 = t_1 + \sqrt{\frac{2C}{a}}$. □

Now we can motivate adr's setting of the threshold to $\sqrt{2a_i C}$. When updating the DBMS at time t_1, adr estimates the deviation after t_1 by the linear function $d(t) = at$, where t is the number of time units after t_1 and a is defined as follows. If we denote by t_0 the time of the position update that immediately precedes the update at t_1, then a is $\frac{2I}{r^2}$ where I is the cost of deviation from t_0 to t_1, and r is the number of time units from t_0 to t_1. This estimation of adr will be explained in a moment. But for this definition of $d(t)$, based on the previous proposition, the optimal update time, i.e. the one that minimizes the information cost per time unit, is at $t_1 + \sqrt{\frac{2C}{a}}$, and at that time the value of the function $d(t)$ is $\sqrt{2aC}$, i.e. $d(t_1 + \sqrt{\frac{2C}{a}}) = \sqrt{2aC}$.

Now we'll motivate the estimation of the future deviation (i.e. the one after the update at time t_1), by the function $d(t) = at$. Adr approximates the current deviation, i.e. the deviation from last update to time t_1, by a linear function with slope $a = \frac{2I}{r^2}$. Observe that at time t_1 this linear function has the same deviation cost (namely I) as the actual current deviation. Based on the locality principle, adr predicts that after the update at t_1, the deviation will behave according to the same approximation function.

6 Comparison of adr and pdr policies by simulation

The analysis compares the information cost of the adr and pdr policies on a set of ten two-hour trips. 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 we show two typical speed curves. For each speed curve, update policy, and update cost C we execute a simulation run. The run computes the information cost [4] (a single number), the number of update messages(also a single number) of the policy on the curve for the given update cost C. 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. We do the same for the average number of update messages.

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 with a particular update policy. This is done as follows. For each time unit there is a 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. If the deviation at time t reaches the threshold th, we generate an update record consisting of: the current time, the current position, the current speed, the next threshold (for pdr it is still th, for adr it is computed as explained in the previous section); the deviation at time t becomes zero. Denote by U the sequence of update records. Using T we compute the total information cost of the deviation, denoted c_1, and using U we compute the total information cost of the updates, c_2. The information cost of the policy on the speed curve is $c_1 + c_2$.

We compare the information cost and number of updates for adr and pdr. Figures 3-10 plot the results of the comparison. Each figure compares adr and pdr(X), i.e. pdr with threshold X, where X = 0.2, 0.5, 1.5, 2.5, or 7.5. When adr is compared with pdr(X), the first threshold of pdr is taken to be X; the following thresholds are determined dynamically, as explained in the previous section. Each figure compares either the information cost, or the number of updates. Each curve representing a policy plots the information cost of the policy or its number of messages as a function of the update cost C.

The basic conclusion from the simulations is that although for some values of the threshold and the update cost the information costs of the two policies are roughly equal, for most thresholds and update costs adr is superior to pdr. The information cost of pdr may be as high as twice the information cost of adr (e.g for threshold=0.2 when C is close to 50, or for threshold 7.5 when $C = 1$).

Observe that the information cost curves of the

[4] Remember, the *information cost* of an update policy on a given speed-curve is computed by using equation 2 for every time interval between two consecutive update points.

adr policy are almost identical, regardless of the initial threshold value (except possibly the extreme value of 7.5). The same holds for the number of updates curves. This indicates that the adr policy exhibits a stable behavior which is independent of the initial conditions.

7 Conclusion

We first introduced Moving Objects Databases and their relationship to spatial and temporal databases, uncertainty, and location management in cellular systems. Then we concentrated on a particular problem, namely reducing the information cost associated with a trip taken by some moving object. The information cost of a trip consists of the cost of position-update messages, plus the cost of the deviation (namely the distance between the database position and actual position).

Then we introduced two position update policies, namely policies for updating the database position of a moving object. These are pdr and adr. Pdr updates the database whenever the deviation exceeds a fixed threshold, and adr does so whenever the deviation exceeds a threshold that varies over time depending on the deviation and position update message cost. We showed by simulation that adr is superior to pdr in terms of information cost.

References

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

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

[3] 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.

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

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

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

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

[8] H. Samet, W.G. Aref, *Spatial Data Models and Query Processing*, In Modern Database Systems, Won Kim ed., Addison Wesley, 1995.

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

[10] R. Snodgrass, *The Temporal Query Language TQuel*, ACM Trans. on Database Systems, 12(2), June 1987.

[11] R. Snodgrass, ed., Data Engineering, Special Issue on Temporal Databases, Dec. 1988.

[12] A. Segev and A. Shoshani, *Logical Modeling of Temporal Data*, Proc. of the ACM-Sigmod International Conf. on Management of Data, 1987.

[13] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev,R. Snodgrass,editors, *Temporal Databases: Theory, design, and Implementation*, Benjamin/Cummings, 1993.

[14] G. Wiederhold, S. Jajodia, W. Litwin, *Dealing with granularity of time in temporal databases*, In Springer-Verlag LNCS, Vol. 498, 1991.

[15] X. Wang, S. Jajodia, V. S. Subrahmanian, *Temporal Modules: An approach toward federated temporal databases*, Information Sciences, Vol. 82 , 1995

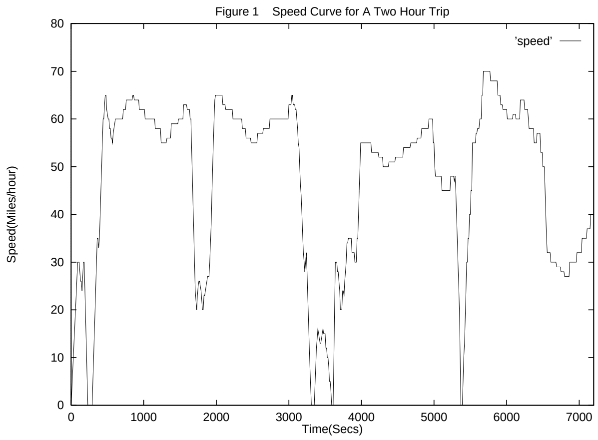
Figure 1 Speed Curve for A Two Hour Trip
'speed'
Speed(Miles/hour)
Time(Secs)
0
10
20
30
40
50
60
70
80
0
1000
2000
3000
4000
5000
6000
7000

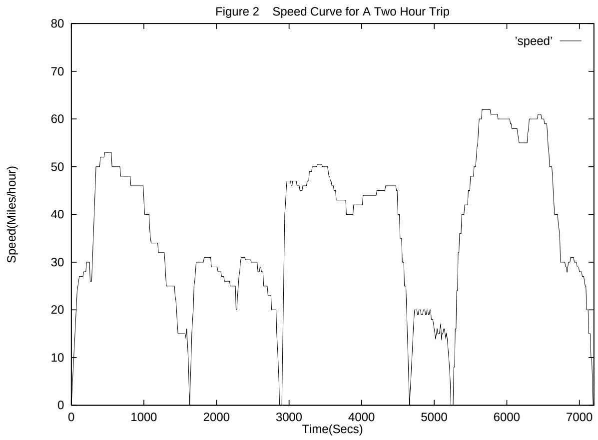
Figure 2 Speed Curve for A Two Hour Trip
'speed'
Speed(Miles/hour)
Time(Secs)
0
10
20
30
40
50
60
70
80
0
1000
2000
3000
4000
5000
6000
7000

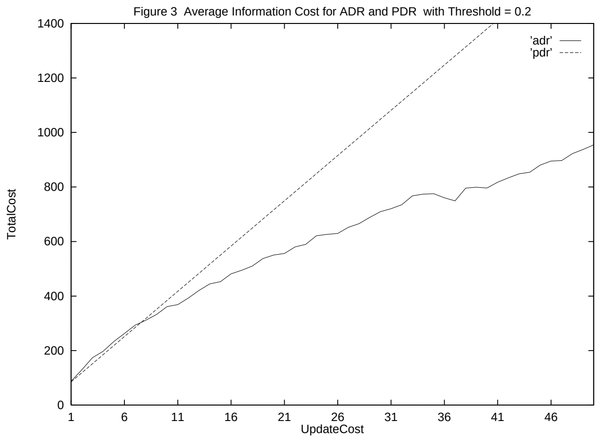
Figure 3 Average Information Cost for ADR and PDR with Threshold = 0.2
'adr'
'pdr'
TotalCost
UpdateCost
0
200
400
600
800
1000
1200
1400
1
6
11
16
21
26
31
36
41
46

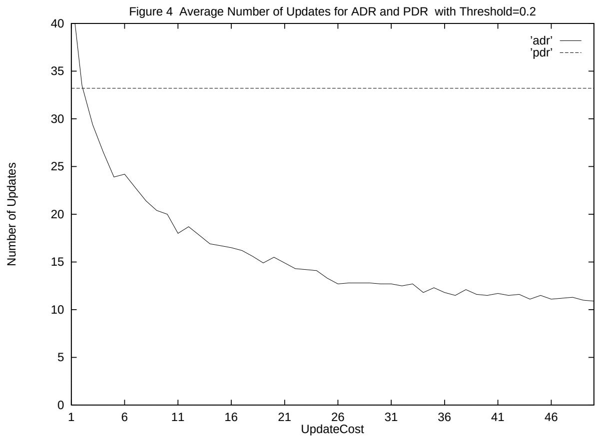
Figure 4 Average Number of Updates for ADR and PDR with Threshold=0.2
'adr'
'pdr'
Number of Updates
UpdateCost
0
5
10
15
20
25
30
35
40
1
6
11
16
21
26
31
36
41
46

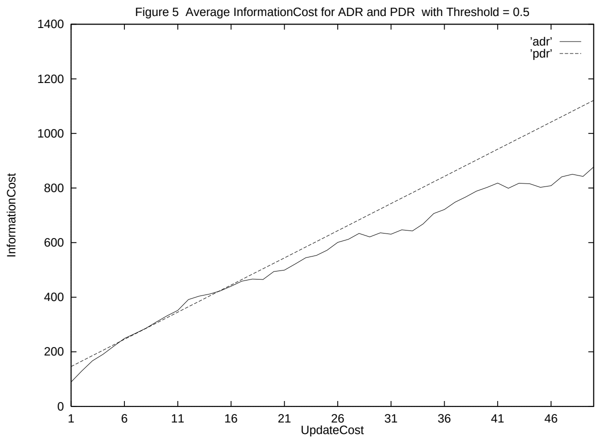
Figure 5 Average InformationCost for ADR and PDR with Threshold = 0.5
'adr'
'pdr'
InformationCost
UpdateCost
0
200
400
600
800
1000
1200
1400
1
6
11
16
21
26
31
36
41
46

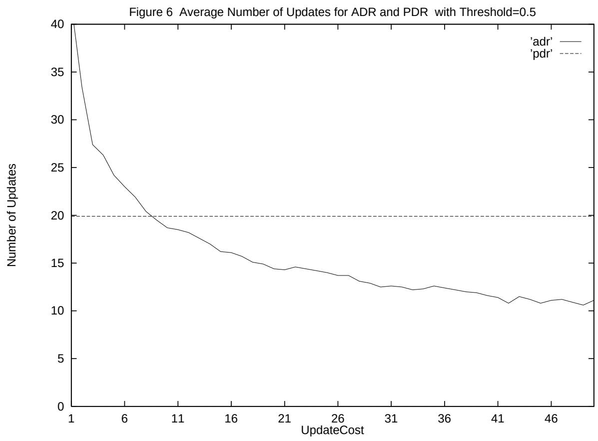
Figure 6 Average Number of Updates for ADR and PDR with Threshold=0.5
'adr'
'pdr'
Number of Updates
UpdateCost
0
5
10
15
20
25
30
35
40
1
6
11
16
21
26
31
36
41
46

Figure 7 Average InformationCost for ADR and PDR with Threshold = 1.5

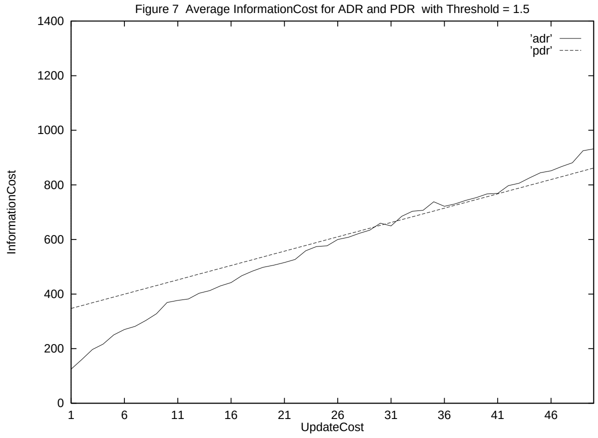

Figure 8 Average Number of Updates for ADR and PDR with Threshold=1.5

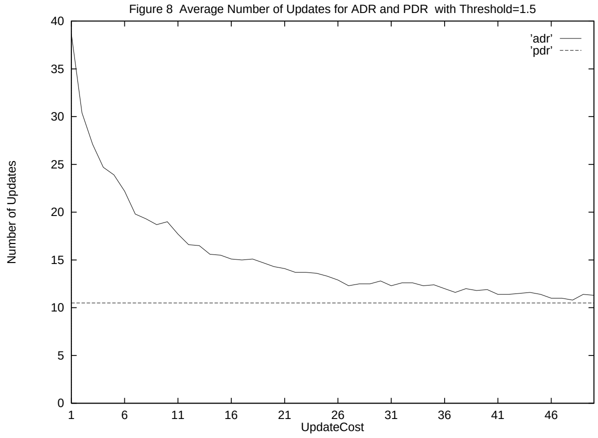

Figure 9 Average Information Cost for ADR and PDR with Threshold = 2.5

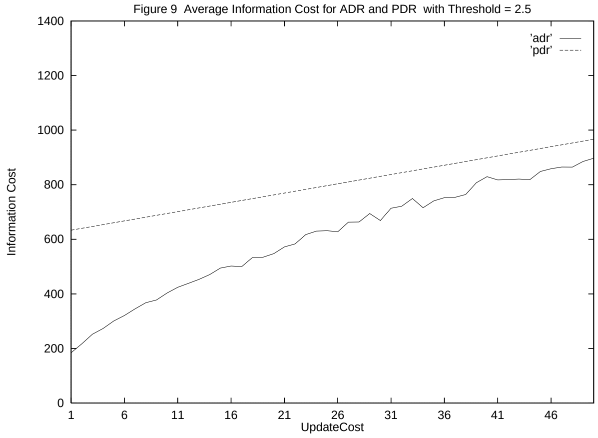

Figure 10 Average Number of Updates for ADR and PDR with Threshold=2.5

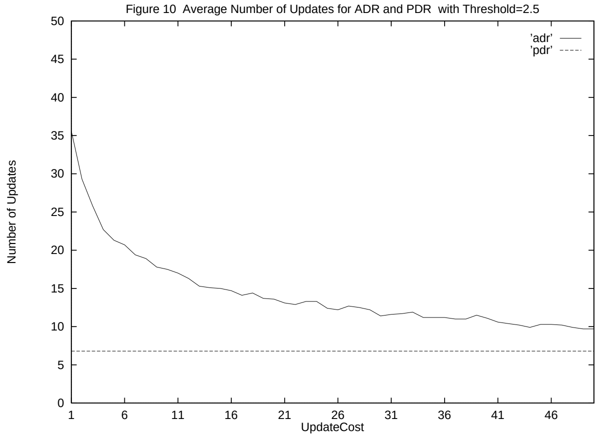

Figure 11 Average Information Cost for ADR and PDR with Threshold = 7.5

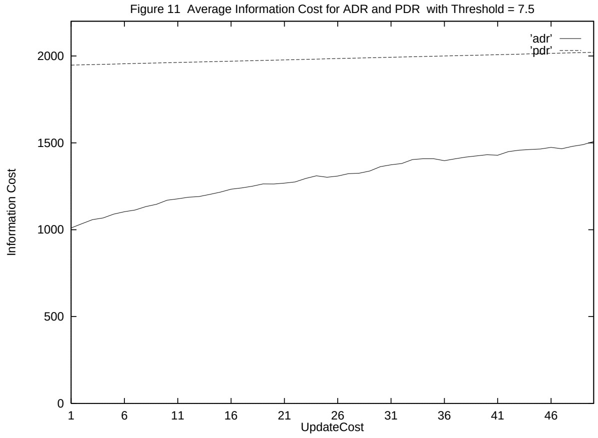

Figure 12 Average Number of Updates for ADR and PDR with Threshold=7.5