Setting of moving object location with optimized tree structure

OGNIAN NAKOV, DESSISLAVA PETROVA
Faculty of Computer Systems and Control
Technical university of Sofia
8 Kl. Ohridski Blv., Sofia
BULGARIA

Abstract: Object tracking is an important part of the common problem for the autonomous motion planning. The main theme of this paper is to propose a data aggregation model for object tracking. Object tracking typically involves two basic operations: update and query. In general, updates of an object's location are initiated when object moves from one point to another. A query in invoked each time when there is need to find the location of the interested object. Location updates and queries may be done in various ways. We propose a new tree structure for in-network object tracking. The location update part of our solution can be viewed as an extension of [2]. In particular, we take the physical topology of the network into consideration. We will reduce the update cost. Several principles, namely, deviation-avoidance and highest-weight-first ones are pointed out to construct an object tracking tree to reduce the communication cost of location update. Our proposed solution tries to divide the network area into squire-like zones, and recursively combine these zones into a tree.

Keywords: object tracking, data aggregation, data source network, tree structure.

1 Introduction

Object tracking is an important part of the common problem for the autonomous motion planning. Existing research efforts on object tracking can be categorized in two ways. In the first category, the problem of accurately estimating the location of an object is addressed [1], [3]. In the second category, in-network data processing and data aggregation for object tracking are discussed [2], [4]. The main theme of this paper is to propose a data aggregation model for object tracking. Object tracking typically involves two basic operations: update and query. In general, updates of an object's location are initiated when object moves from one point to another. A query is invoked each time when there is need to find the location of the interested object. Location updates and queries may be done in various ways. A naive way for delivering a query is to flood the whole network. The data source whose covering range contains the queried object will reply to the query. Clearly, this approach is inefficient because a considerable amount of energy will be consumed when the network scale is large or when the query rate is high. Alternatively, if all location information is stored at a specific data source (e.g. sink), no flooding is needed. But, whenever a movement is detected, update message have to be sent. One drawback is that when objects move frequently, abundant update messages will be generated. The cost is not justified when the query rate is low. Clearly, these are tradeoffs.

In [2], a Drain-And-Balance (DAB) tree structure is proposed to address this issue. This is an object tracking approach where query messages are not required to be flooded and update messages are not always transmitted to the sink. However [2] has two drawbacks. First, a DAB tree is a logical tree not reflecting the physical structure of the data network hence, an edge may consist of multiple communication hops and a high communication cost may be incurred. Second, the construction of the DAB tree does not take query cost into consideration. Therefore, the result may be efficient in some cases.

To relieve the aforementioned problems, we propose a new tree structure for in-network object tracking. The location update part of our solution can be viewed as an extension of [2]. In particular, we take the physical topology of the network into consideration. We will reduce the update cost. Several principles, namely, deviation-avoidance and highest-weight-first ones are pointed out to construct an object tracking tree to reduce the communication cost of location update. Our proposed solution tries to divide the network area into squire-like zones, and recursively combine these zones into a tree. In our previous research we have proposed another solution, which is described in the next section. The new solution is compared with a naive scheme, DAB scheme and our previous solution – Singleton Subtree Algorithm (SSA).

2 Problem formulation

We consider a network of data sources deployed in a field for the purpose of object tracking. Data sources locations are already known at a special node, called sink, which serves as the gateway of the network to outside world. We adopt a simple nearest-node model, which only requires the data source that receives a message from the object to report the sink. Therefore, the data network's field can be partitioned can be partitioned into a graph, as depicted in Fig.1.

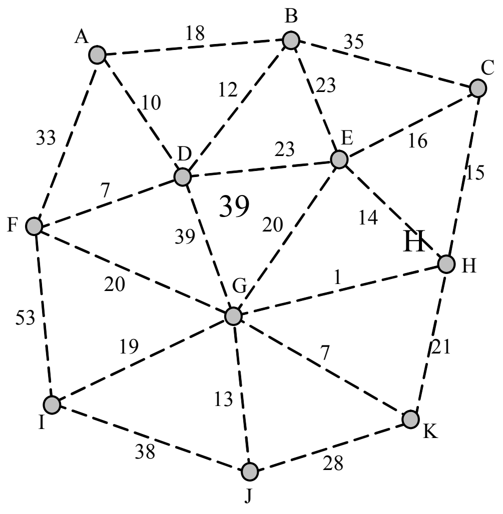


Fig.1. The graph G corresponding to the data source network

We propose a data aggregation model for object tracking. We assume that whenever an object arrives at or departs from the range (polygon) of a data source, a detection event will be reported. Two data sources are called neighbours if their ranges share a common boundary on the graph; otherwise, they are nonneighbours. Multiple objects may be tracked concurrently in the network, and we assume that from statistics, it is possible to collect the event rate between each pair of neighbouring data sources to represent the frequency of object traveling from one point to another. For example, in Fig.1, the arrival and departures rates between data sources are shown on the edges of the graph. In addition, the communication range of data sources is assumed to be large enough so that neighbouring data sources can communicate with each other directly. Thus, the network topology can be regarded as undirected weight graph $G=(V_G, E_G)$ with V_G representing data sources and E_G representing links between neighbouring data sources. The weight of each link $(a,b) \in E_G$, denoted by $w_G(a, b)$, is the sum of event rates from a to b and b to a. This is because both arrival and departure events will be reported in our scheme.

In light of the storage in data sources, the network is able to be viewed as a distributed database. We exploit the possibility of conducting in-network data aggregation for object tracking. Similar to the approach in [2], a logical weighted tree T will be constructed from G. For example, Fig.2 shows an object tracking tree T constructed from the network G in Fig.1. Movement event of objects are reported based on the following rules. Each node a in T will maintain a detected list $DL_a(L_0, L_1, \ldots, L_k)$ such that L_0 is the set of objects currently inside the coverage of data source a itself, and L_i, $i=1, \ldots, k$, is the set of objects currently inside the coverage of any data source who is in the subtree rooted at the ith child of data source a, where k is the number of children of a.

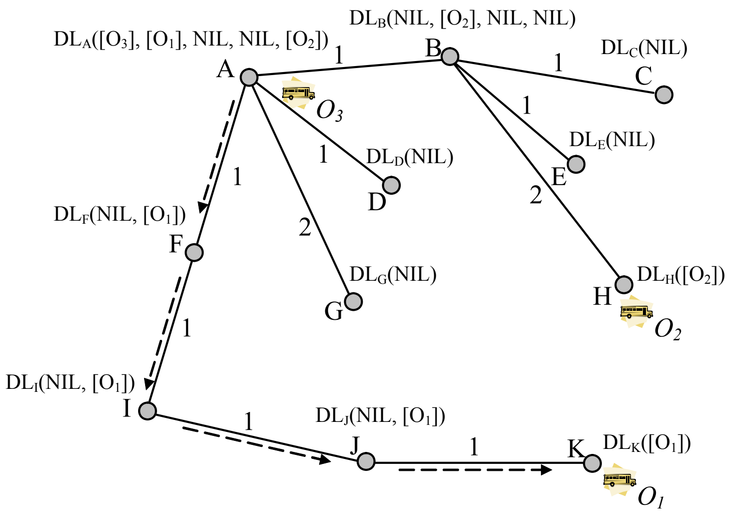


Fig.2. An object tracking tree

When an object O moves from the range of a to that of b, a department event $dep(O, a, b)$ and an arrival event $arv(O, b, a)$ will be reported by a and b, respectively, alone the tree T. On receiving such an event, a data source x takes following actions:

1) If the event is $dep(O, a, b)$, x will remove O from the proper L_i in DL_x such that data source a belongs to the ith subtree of x in T. If $x=a$, O will be removed from L_0 in DL_x. Then x checks whether data source b belongs to the subtree rooted at x in T or not. If not, the event $dep(O, a, b)$ is forwarded to the parent node of node of x in T.

2) If the event is $arv(O, b, a)$, x will add O to the proper L_i in DL_x such that data source b belongs to the ith subtree of x in T. If $x=b$, O will be added to L_0 in DL_x. Then x checks whether data

source *a* belongs to the subtree rooted at *x* in *T* or not. If not, the event *arv(O, b, a)* is forwarded to the parent node *x* in *T*.

The above data aggregation model guarantees that, disregarding transmission delays, the data structure DL_i always maintains the objects under the coverage of any descendant of data source *i* in *T*. Therefore, searching the location of an object can be done efficiently in *T*; a query is only required to be forwarded to a proper subtree and no flooding is needed. For example, Fig.2 shows the forwarding path of a query for *Car1* in *T*.

Our goal in this case is to construct an object tracking tree $T=(V_T, E_T)$ that incurs the lowest communication cost given a network $G=(V_G, E_G)$ and the corresponding event rates, where $V_T=V_G$ and E_T consist of $|V_T|$ - *1* edges with the sink as the root. Intuitively, *T* is a logical tree constructed from *G*, in which each edge $(u,v) \in T$ is one of the shortest paths connecting data sources *u* and *v* in *G*. Therefore, the weight of each edge *(u, v)* in *T*, denoted by $w_T(u, v)$, is modeled by the minimum hop count between *u* and *v* in *G*. The cost function can be formulated as $C(T) = U(T) + Q(T)$, where *U(T)* denotes the update cost and *Q(T)* is the query cost.

In our previous work [5] we have developed algorithm (SSA), which reduces the update cost. Initially, that algorithm treats each node as a singleton subtree. Then more links are included to connect these subtrees together. In the end, all subtrees are connected into one tree *T*.

3 Algorithm Design

This section presents our new algorithm, based on the concept divide-and-conquer – Divide and Conquer Tree Algorithm (DCTA). We devise the DCTA to further reduce the update cost.

The DCTA is based on the following locality concept. Assume that u is *v*'s parent in *T*. For any edge $(x, y) \in E_G$ such that $x \in Subtree(v)$ and $y \notin Subtree(v)$, arrival/departure events between x and y will cause a message to be transmitted on *(p(v),v)*, thus increasing the value of $\sum_{(x,y)\in E_G \wedge x\in Subtree(v) \wedge y \notin Subtree(v)} w_G(x,y)$. Therefore, the perimeter that bounds the area of data source in each *Subtree(v)* will impact the update cost U(T). A longer perimeter would imply more events crossing the boundary. In geometry, it is clear that a circle has the shortest perimeter to cover the same area as compared with other shapes. Circle like shapes, however, are difficult to be used in an iterative tree construction. As a result, DCTA will be developed based on squire-like zones.

The algorithm builds *T* in an iterative manner based on two parameters, *α* and *δ*, where *α* is power of 2 and *δ* is a positive integer. To begin with, DCTA first uses (*α–1*) horizontal lines to divide the network area into *α* strips. For each horizontal line between two strips, we are allowed to further move it up and down within a distance no more than *δ* units. This gives *2δ + 1* possible locations of each horizontal line. For each location of the horizontal line, we can calculate the total event rate that objects may move across the line. Then we pick the line with the lowest total event rate as its final location. After all horizontal lines are determined, we then further partition the network area into α^2 regions by using *(α–1)* vertical lines. Following the adjustment as above, each vertical line is also allowed to move left and fight within a distance no more than *δ* units and the one with the lowest total event rate is selected as its final location.

After the above steps are completed, the network area is divided into α^2 squire-like zones. First, we run SSA in each zone. This will result in one or multiple subtrees in each zone. Next, we will merge subtrees in the above α^2 zones recursively as follows: First, we combine these zones together into $\frac{\alpha}{2} \times \frac{\alpha}{2}$ larger zones, such that each larger zone contains *2 x 2* neighbouring zones. Then, we merge subtrees in these *2 x 2* zones by sorting all interzone edges according to their event rates into a list L and feeding L to the SSA. Second, we further combine the above larger zones together into $\frac{\alpha}{4} \times \frac{\alpha}{4}$ even larger zones, such that each even larger zone contains *2 x 2* neighbouring larger zones. This process is repeated until one single tree is obtained.

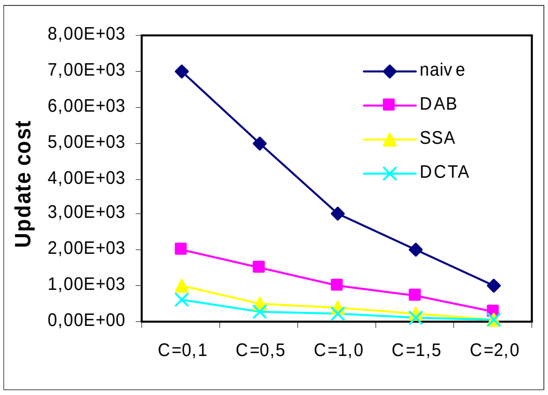
4 Simulation results and conclusions

We have simulated a network field of size 256x256. Unless otherwise stated, 4 096 data sources are deployed in the network. Two deployment models are considered. In the first one, data sources are regularly deployed as a 64x64 grid like network. In the second model, data sources are randomly deployed. In both models the sink may be located near the center of the data network or one corner of the data network.

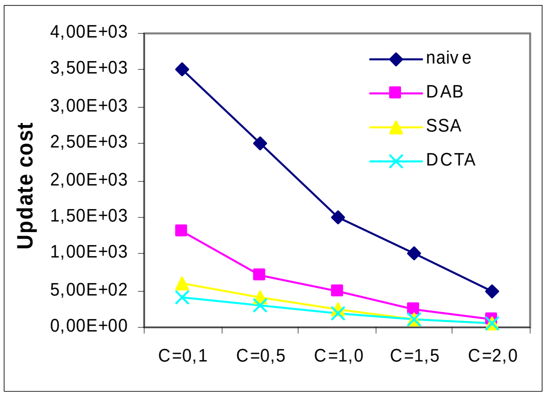
Events rates are generated based on a model similar to the city mobility model in [2]. Assuming the network field as a squire of size *r x r*, the model divides the field into *2 x 2* subsquires called level-1 subregions. Each level-1 subregion is further divided into *2 x 2* subsquires called level-2 subregions. This process is repeated recursively. Given an object located in any position in the field, it has a probability p_1 to leave its current level-1subregion, and a probability p_1-*1* to stay. In the former case, the object will move either horizontally or vertically with a distance *r/2*. In the latter case, the object has a probability p_2 to leave its current level-2 subregion, and a probability p_2-*1* to stay. Again, in the former case, the object will move either horizontally or vertically with a distance $r/2^2$, and in the latter case it may cross level-3 subregions. The process repeats recursively. The possibility p_i is determined by an exponential probability $p_i = e^{-C \cdot 2^{d-1}}$, where *C* is a positive constant and *d* is the total number of levels. In fact, the above behavior only formulates how objects move in the network field.

We compare our scheme with a naive scheme, the DAB scheme [2] and our previous SSA. In the naive scheme, any update is sent to the sink. In this case query cost is always zero, so it is preferable when the query rates are relatively high. For the DAB scheme, all points are considered leaf nodes. When two subtrees are merged into one, the root of the subtree which is closer to the sink will become the root of the merged tree.

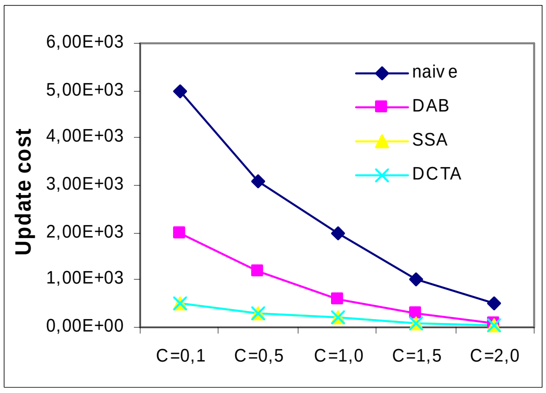
First, we observe the advantages of using in-network processing to reduce update cost. Fig.3 shows the result under different values of *C* for regular and random data source deployment. As can be seen, a larger C implies a higher moving locality, thus leading to a lower update cost. The naive scheme has the highest update cost, which is reasonable. By exploiting the concept of deviation avoidance and taking the physical topology into account, SSA and DCTA further outperform DAB.



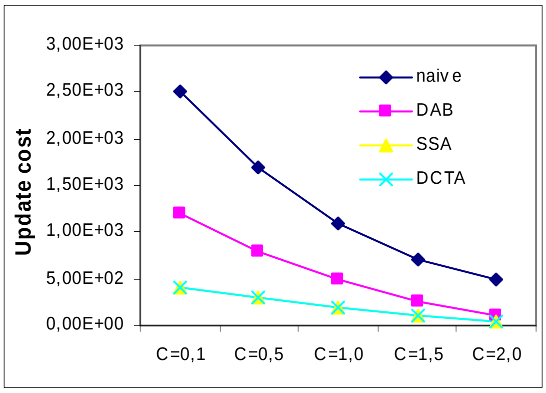
a) Regular deployment, sink at the corner



b) Regular deployment, sink at the center



c) Random deployment, sink at the corner

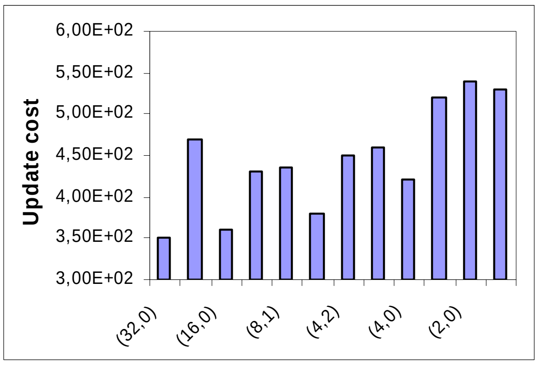


d) Random deployment, sink at the center

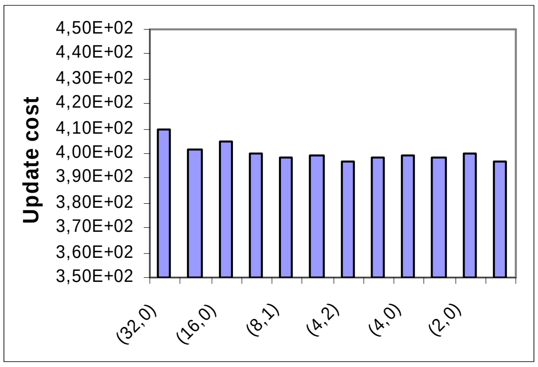
Fig.3. Comparison of update cost

Next, we investigate the effect of deployment models. By comparing, the graphs in Fig.3, we see that DCTA outperforms SSA under regular deployment, but the advantage is almost negligible under random deployment. This is because maintaining the shapes of subtrees in DCTA is difficult.

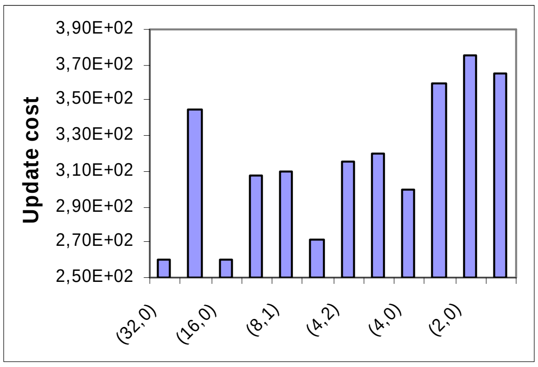
Finally, to get further insight into performance of DCTA, we vary *α* and *δ*, and show the result in Fig.4, where 4096 and 2 500-node networks are simulated. Note that when *α=1* and *δ=0*, DCTA is equivalent to SSA. For regular deployment, DCTA performs well when α is than 4. However, for random deployment, the DCTA does not perform well because maintaining the shapes of subtrees is difficult. Furthermore, it can be seen that when $\delta=0$, DCTA has better performance. This means that a squire-like zone is better than a rectangular-like zone.



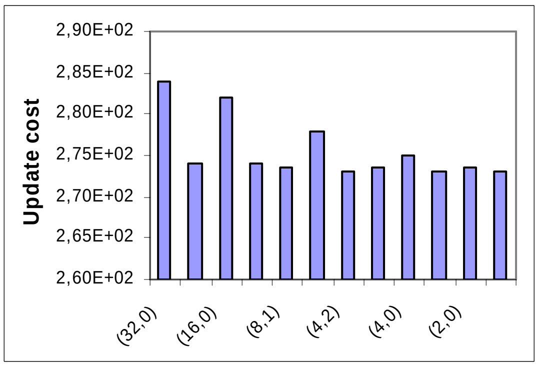
a) Regular deployment, C=0.1, 4096 nodes



b) Random deployment, C=0.1, 4096 nodes



a) Regular deployment, C=0.1, 2500 nodes



b) Random deployment, C=0.1, 2500 nodes

Fig.4. Comparison of update cost under different α and δ

Future work

In the future we have plans to design new algorithm, which will reduce total cost of the network comunication. It will be based on SSA and DCTA.

References:

[1] Aslam J., Z. Butler, F. Constantin, V. Crespi, G. Cybenko, D. Rus. Tracking a Moving Object with Binary Sensors. *Proc. ACM SenSys Conf.*, November, 2003.

[2] Kung H., D. Vlah. Efficient Location Tracking Using Sensor Networks. *Proc. IEEE Wireless Communication and Networking*, 2003.

[3] Mechitov K., S. Sundresh, Y. Kwon, Cooperative Tracking with R-2003-2379. *University of Illinois at Urbana-Champaign*, 2003.

[4] Zhang W., G. Cao. DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks. *IEEE Trans. Wireless Communication.* vol. 3, no. 5, pp. 1689-1701, September, 2004.

[5] Nakov O., D. Petrova. Optimization strategies for moving object tracking. In Year-book of “Telematika” College, Stara Zagora, 2007.