13

Reordering of Location Identifiers for Indexing an RFID Tag Object Database

Sungwoo Ahn and Bonghee Hong
Pusan National University
Republic of Korea

1. Introduction

Radio frequency identification (RFID) has become one of the emerging technologies for a wide area of applications such as automated manufacturing, inventory tracking, and supply chain management. RF technologies make it possible to identify individual items in real-time by means of automatic and fast identification. Besides the real-time identification, RF technologies give additional advantages for monitoring of field-based operation by tracking and tracing the location of tags attached to items. By using queries on trajectories of RFID tag data, RFID applications can get events about field-based situation and then respond to them.

To store and retrieve tag data efficiently, it is important to provide an index for the repository of tag data. The EPCglobal, being in charge of a standards management and development for RFID related technologies, proposes EPC Information Service (EPCIS) as the repository for tag events. The EPCIS is a standard interface for access and persistent storage of tag information. Tag data stored in the EPCIS consists of the static attribute data and the timestamped historical data. Historical information is continuously collected and updated whenever each tag is identified by an RFID reader. The EPCIS usually stores them at the base table of a database for efficient management of those data. It is necessary to execute queries on the EPCIS whenever applications want to retrieve the location history of specific tags. However, it is inefficient to look up all the records of the table because a large amount of historical information for tags is to be accumulated in the base table.

For efficient query processing of tracing tags, an index structure can be constructed based on tag events generated when a tag goes in and out a location where a reader places. Among timestamped historical information contained in tag events, an RFID application uses the location identifier (LID), the tag identifier (TID), and the identified time (TIME) as predicates for tracking and tracing tags. To index those values efficiently, we can define the *tag interval* by means of two tag events generated when the tag enters and leaves a specific location, respectively. The tag interval could be represented and indexed as a time-parameterized line segment in a three-dimensional domain which is constituted by LID, TID, and TIME axes.

Tag intervals in a three-dimensional index are sequentially stored and accessed in one-dimensional disk storage. Since logically adjacent tag intervals are to be retrieved together at a query, they should not be stored far away from each other in the disk to minimize the cost

of disk accesses. Logical closeness has been studied to determine the distance between domain values representing the coordinate of those objects. A logically adjacent object to a specific object on the data space has the shortest distance to that object by using some distance measure. Note that the change of the order of domain values results in the variation of distances between objects because of a different distribution of objects on the data space. Thus, domain values should be ordered properly in each domain in order to keep logical closeness between objects.

Most works for clustering spatial objects have used the spatial distance in the spatial domain as the distance measure. To diminish the number of disk accesses at answering spatial queries, they stored adjacent objects sequentially based on the spatial proximity. In addition to the spatial proximity, moving object databases M.F. Mokbel and Y. Theodoridis have applied the temporal proximity to the characteristic for the distance measure in the time domain. Previous works assumed that all domains on the data space provide the proper proximity about measuring the distance between domain values.

Since an LID represents the location where a tag stays or passes, the LID domain should provide logical closeness for the dynamic flow of tags along locations. The problem is that there is no rule of assigning LIDs to RFID locations in order to keep this property. If LIDs are arbitrarily arranged in the domain without considering tag flows, tag intervals would be scattered into the data space irrespective of logical closeness. Because this situation causes random disk accesses for searching logically adjacent tag intervals, the cost of query processing will be increased.

To solve this problem, we propose a reordering method for arranging LIDs in the LID domain. The basic idea is to compute the distance between two LIDs for preserving logical closeness of tag intervals. To do this, we define the proximity function based on a new *LID proximity* between two LIDs. The proximal distance between LIDs can be computed by the tag movements. To determine LID proximity, we need to examine *the path of tag flows* which is generated by tag movements. Then, we define the *LID proximity function* which computes the distance between LIDs with the dynamic flow of tags. To determine a sequence of LIDs based on LID proximity, we construct a weighted graph and generate the ordered LID set. It is possible to store logically adjacent tag intervals close to each other in the disk because our reordering method can keep the correlation between the distance and logical closeness of tag intervals. To prove this, we evaluate the performance of the index scheme using LIDs based on LID proximity as domain values. We also compare it with the index scheme using the numerical order of LIDs.

The reminder of this paper is organized as follows. Section 2 defines the problem of an LID as the domain value for tag intervals and describes the needs of reordering LIDs. Section 3 examines the path of tag flows based on the characteristics of RFID locations and tag movements, and then defines the LID proximity function. In Section 4, we propose a reordering scheme of LIDs using a weighted graph that is constructed by LID proximity. Section 5 presents some experimental results of performance evaluation for the proposed reordering scheme. A summary is presented in Section 6.

2. Problem definition

2.1 Target environment

Whenever the tag attached to an item passes through an RFID reader, the reader collects the tag's information within its interrogation zone. In an RFID middleware system, gathered

information are represented as *EPCIS tag events* and stored at the persistent storage in order to answer tag related queries. Since a tag event contains several timestamped historical information, it could represent the dynamic flow of tagged items between RFID locations placed along tag routes. If an RFID application wants to know a history of these items, a query processor can make an answer to the application by retrieving suitable tag events in a repository of tag events.

In timestamped historical information, a query processor usually employs the tag identifier (TID), the location identifier (LID), and the timestamp (TIME) as the predicates of queries for tracing tag locations. For efficient query processing of tracing tags, the tag trajectory should be modeled and indexed by using these predicates.

Note that RFID locations are different from the spatial locations to represent real positions on the map. There are two types of location related to EPCIS tag events according to a business perspective for an RFID location. One is the physical position which identifies the tag. We denote this position as the *read point* (RP). The read point does not provide the information where a tag visited or stays by itself because it designates only the place at which a tag was detected. The other is the region where a tag stays. We denote this region as the *business location* (BizLoc). The business location represents the place where a tag is assumed to be until a subsequent tag event is generated by a different business location. Since most of RFID applications trace a business flow of tagged items, they have an interest in the business location instead of the read point as the location type of the tag. Therefore, it is natural to use the business location as the LID predicate for tracing tag locations.

The EPCIS tag event could be modeled as the time parameterized interval in a three-dimensional domain whose axes are LID, TID, and TIME. We denote this interval as the *tag interval* (TI). The tag interval is a line segment that connects two coordinates in a three-dimensional space when the tag enters and leaves a specific business location. In this manner, the trajectory of a tag is represented as a set of tag intervals which are associated with the tag.

Predicate			Query results	Query types
LID	TID	TIME		
point/set/ range	*	point/range	TID(s)	Observation Query (OQ)
*	point/set/rang e	point/range	LID(s)	Trajectory Query (TQ)

Table 1. Query classification for tracing tag locations

Queries for tracing tags are classified into two types according to a kind of restricted predicate as shown in Table 1. An *observation query* (OQ) is used to retrieve the tag(s) that are identified by the specified business location(s) in the specified time period. A *trajectory query* (TQ) is used to retrieve the business location(s) that the specific tag(s) enters and leaves within the specified period. Queries in Table 1 can be extended to a combined query by performing two queries in the order OQ and TQ.

To support fast retrieving of desired trajectories of tags, it is necessary to store and search tag trajectories by means of an index structure. Each leaf node of the index references logically adjacent tag intervals on the data space by using minimum bounding box (MBB). Then, tag intervals referenced by index nodes are sequentially stored and accessed in one-

dimensional disk storage. Tag intervals on each leaf node are stored at the same disk page in order to minimize disk seeks.

2.2 Problem of using an LID as the domain value

(a) Logical closeness between tag intervals is very important for simultaneous accessing at the query. It gives a great influence on the performance of query processing because the cost of disk accesses depends on the sequence of storing tag intervals on the disk. For example, let us assume that a query, *Qi*, would search tag intervals by the index structure. If all tag intervals accessed by *Qi* are stored in P3 as shown in Fig. 1-(a), a query processor needs to access just one disk page, P3. If those tag intervals are dispersed to disk pages, P2, P3, and P5 as shown in Fig. 1-(b), however, a query processor usually require the additional cost about accessing two pages, P2 and P5. To minimize the cost of disk accesses, logical closeness between tag intervals in the same disk page should be higher than logical closeness to others.

(a) All tag intervals accessed by the query Q_i are stored in P3

(b) Tag intervals accessed by the query Q_i are stored in P2, P3, P5

Fig. 1. An example of different access cost of the disk

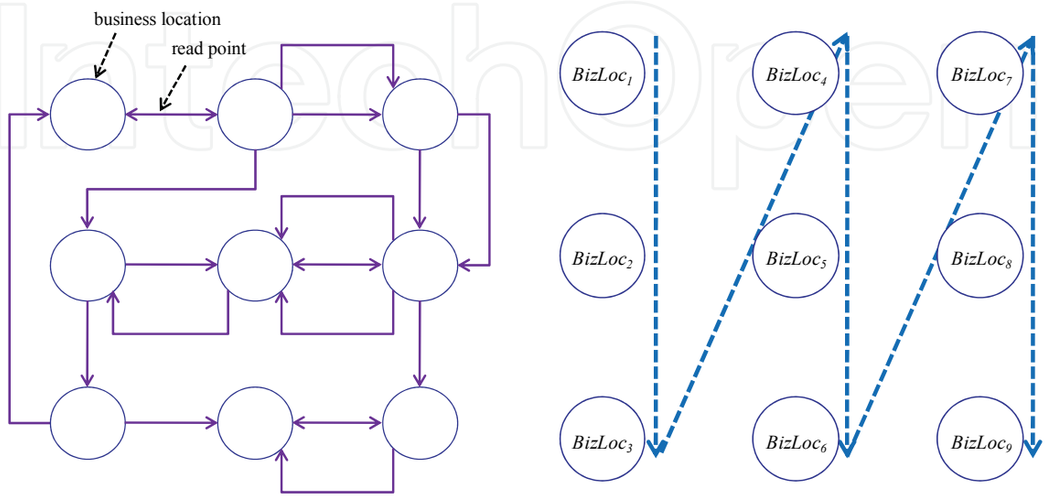
Distance between two tag intervals on the data space should be computed for measuring logical closeness between them. If the distance measure keeps logical closeness between tag intervals, we can say that the nearest tag interval to a specific tag interval has the shortest distance to that tag interval. The distance is normally measured based on proximity between domain values on the data space. Thus, we need to examine the characteristic of each domain's proximity in order to keep correlation between the distance and the logical closeness.

The TIME domain in 3-dimensional space should provide chronological closeness between tag intervals. We usually achieve this closeness with assigning timestamps based on the temporal proximity in the TIME domain. The TID is the fixed identifier, which is related to Electronic Product Code (EPC), for a tagged item. The EPC can be composed of three parts – *Company*, *Product* and *Serial*. Since the EPC scheme assign an identifier to a tag by a hierarchical manner with three parts, the TID can imply logical closeness between grouped tags.

A tag produces a dynamic flow while moving between business locations. Since a query for tracing tags would give tag's traces, the LID domain should provide the closeness of tag intervals about tag movements. On the contrary to the TID, the LID is not the predefined identifier. We can assign business locations to LIDs by various numbering methods. For example, it can be some lexicographic method for measuring the distance in an RFID applied system. It is also possible to apply spatial distance measure such as Hilbert curve, Z-ordering, and Row-Prime curve. Figure 2 shows an example of numbering LIDs for describing business locations and read points.

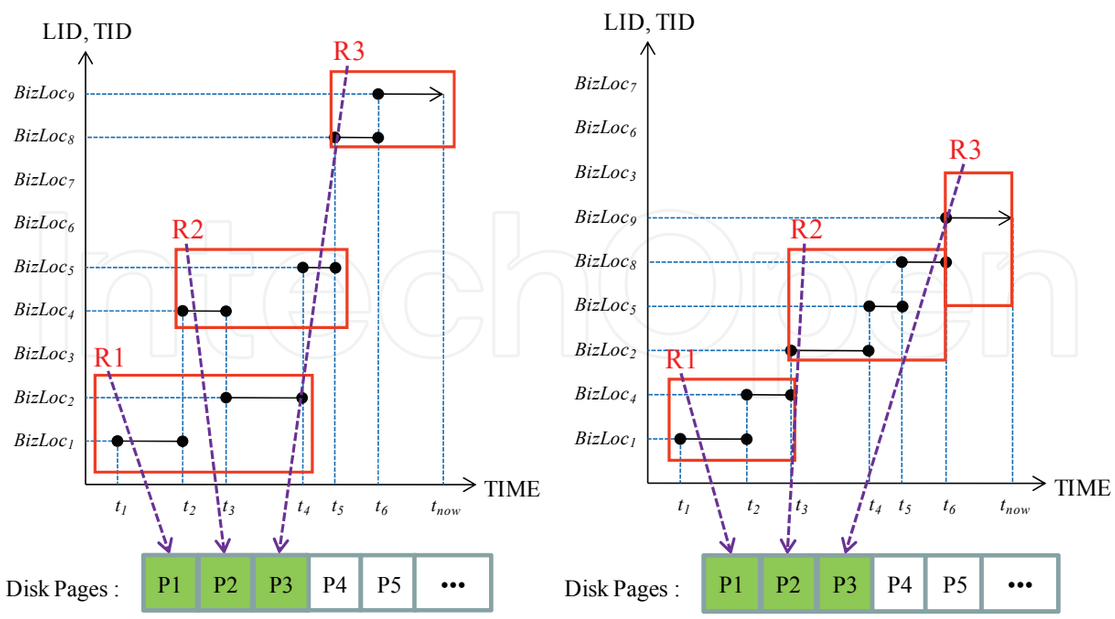
Despite the existence of various LID numbering methods, the problem is that they do not have an inherence property of proximity for providing logical closeness related to the

dynamic flow of tags. If LIDs are assigned to business locations without considering tag's flows, each leaf node of the index may reference tag intervals irrespective of their logical closeness. This means that the index structure does not guarantee a query processor to retrieve results with minimal cost because logically adjacent tag intervals will be stored far away from each other at disk pages.



(a) The organization of RFID locations (b) An example of assigning identifiers to business locations of (a)

Fig. 2. An example of numbering method for business locations



(a) Assigning LIDs by some lexicographic method (b) Assigning LIDs by the tag flow

Fig. 3. Different organization of the index according to the order of LIDs

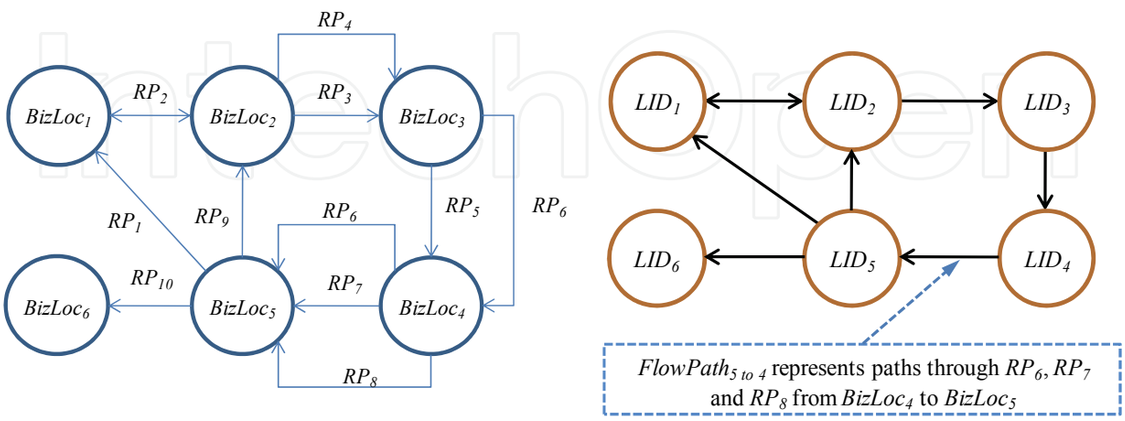
This situation is illustrated in Fig. 3. Assume that a tag, TID_m, passes through business locations of Fig. 2 in $BizLoc_1$, $BizLoc_4$, $BizLoc_2$, $BizLoc_5$, $BizLoc_8$, and $BizLoc_9$ order. If LIDs are arranged according to the order of Fig. 2-(b), tag intervals would be distributed on the data space and stored at disk pages as shown in Fig. 3-(a). Let TQ_i = (*, TID_m, [t_3⊦, t_6⊣]) be the trajectory query for searching LIDs where TID_m stayed during the period t_3 to t_6. When TQ_i is processed at the index organized as shown in Fig. 3-(a), a query processor should access disk pages, P1, P2, and P3 because all tag intervals generated during the period t_3 to t_6 are dispersed to all MBBs, R1, R2, and R3. However, if we make LIDs reorder based on the order of TID_m's movement as shown in Fig. 3-(b), tag intervals during the period t_3 to t_6 can be referenced by one leaf node having R2. A query processor needs to access only the page, P2 in order to process TQ_i over the index of Fig. 3-(b).

We solve this problem by defining LID proximity. LID proximity determines the distance between two LIDs in the domain. If two LIDs have higher LID proximity than others, corresponding tag intervals could be distributed closely on the data space. In the remainder of this paper, we analyze factors to deduce LID proximity. Subsequently, we define the LID proximity function based on those factors. To determine the order of LIDs with LID proximity, we also propose the reordering scheme of LIDs.

3. Proximity between LIDs

3.1 LID proximity based on the path of tag flows

Tagged items always move between the business locations passing through the read points placed in the entrance of each business location. If there are no read points connecting with specified business locations, however, the tagged item cannot move directly between them. Although read points exist, the tag movement can also be restricted because of a business process of an applied system. According to these restrictions, there is a predefined path which a tag is able to cross. We designate this path as the *path of tag flows* (FlowPath). The items attached by the tags generate a flow of tags passing through the path. The FlowPath from LID_i to LID_j is denoted as $FlowPath_{i\ to\ j}$.



(a) A graph example for business locations connected by their read points

(b) A generation of FlowPaths between LIDs using the graph (a)

Fig. 4. An example of representing FlowPaths with business locations and their read points

The FlowPath is a simple method for representing the connection property between two business locations. It is possible to generate the FlowPath with a connected graph of business locations and read points as shown in Fig. 4. To do this, $BizLoc_1$ to $BizLoc_6$ in Fig. 4-(a) are corresponding with location identifiers, LID_1 to LID_6 in Fig. 4-(b), respectively. If one or more read points connect particular two business locations, they are represented as a single line connecting two LIDs as shown in Fig. 4-(b). Properties of a FlowPath are as follows.

1. A FlowPath is a directional path because a read point has a directional property among three types of directions – IN, OUT, and INOUT.
2. The number of FlowPaths connecting one LID with other LIDs is more than one because all business locations have one or more read points connecting other business locations.
3. There may be no FlowPath which connect two particular LIDs directly. In this case, a tag should pass through another LIDs connected with those LIDs by FlowPaths in order to move from one to the other.

As mentioned in Section 2, a query for tracing tags is interested in a historical change of locations for the specific tag. This means that tag intervals generated by business locations along the specific FlowPath have higher probability of simultaneous access than others. Therefore, it is necessary to reorder LIDs based on the properties of a FlowPath for the efficient query processing. We first define the proximity between LIDs for applying to the LID reordering as follows.

Definition 1. ***LID Proximity (LIDProx)*** is the closeness value between two LIDs in the LID domain for tag intervals. We denote LID proximity between LID_i and LID_j as $LIDProx_{ij}$ or $LIDProx_{ji}$.

We also denote the LID proximity function for computing $LIDProx_{ij}$ as LIDProx(i, j) or LIDProx(j, i). LID proximity between two LIDs has following properties.

1. Any LID_i in the LID domain should have a LID proximity value to any LID_j where $i \neq j$.
2. $LIDProx_{ij}$ is equal to $LIDProx_{ji}$ for all LIDs.
3. If LID_k, having the property LIDProx(i, j) < LIDProx(i, k), does not exist, the nearest LID to LID_i is LID_j.

It is possible to represent LID proximity between all LIDs with a graph based on the FlowPath. To do this, a graph based on the FlowPath should satisfy following conditions. First, a graph should be a weighted graph that all edges in a graph have a weight value. Second, a graph should be a complete graph by the property (1) of LID proximity. Third, a graph should be an undirected graph by the property (2) of LID proximity. By these conditions, we define the graph G based on the FlowPath as follows.

- $G = (V, E, W)$
- V = LIDSet = $\{LID_1, LID_2, \ldots, LID_n\}$ where n is the number of LIDs in the LID domain
- $E = \{(LID_i, LID_j) \mid LID_i \in \text{LIDSet}, LID_j \in \text{LIDSet}, i \neq j\}$
- $w : E \rightarrow R$, $w(i, j) = \text{LIDProx}(i, j) = \text{LIDProx}(j, i) = w(j, i)$

3.2 LID proximity function

The tag movements along FlowPaths and the frequency of their related queries are changed continuously over time. Consequently, the access probability of tag intervals generated by any two LIDs also changes as time goes by.

For applying dynamic properties of the FlowPath to LID proximity, we define the *LID proximity function* as shown in Eq. 1; we denote T as the time to compute LID proximity, $\text{LIDProx}_T(i, j)$ as the LID proximity function at time T, $\text{LIDProx\_OQ}_T(i, j)$ and LIDProx_TQ(i, j) as proximity functions invented by properties of an observation query and a trajectory query, respectively.

$$\text{LIDProx}_T(i,j) = \alpha \times \text{LIDProx\_OQ}_T(i,j) + (1-\alpha) \times \text{LIDProx\_TQ}_T(i,j) \tag{1}$$

LIDProx(i, j) is the time parameterized function that the closeness value between LID_i and LID_j changes over time. To consider the closeness value for an observation query and a trajectory query altogether, the function calculates the sum of LIDProx_OQ(i, j) and LIDProx_TQ(i, j) with the weight value. The weight α determines the applying ratio between two proximity functions as shown in Eq. 2; we denote $OQ_{ij,t}$ as the number of observation queries for LID_i and LID_j at time t and $TQ_{ij,t}$ as the number of trajectory queries for LID_i and LID_j at time t.

$$\alpha = \begin{cases} 0 \text{ or } 1 & \text{if no queries are processed for } LID_i \text{ and } LID_j \\ \sum_{t=1}^{T} OQ_{ij,t} \Big/ \sum_{t=1}^{T} \left(OQ_{ij,t} + TQ_{ij,t} \right) & \text{otherwise} \end{cases} \tag{2}$$

LID proximity for an observation query is proportionally influenced by the number of tag intervals generated by two LIDs which are predicates of the observation query. The function LIDProx_OQ(i, j) computes LID proximity for an observation query with the ratio of tag intervals generated by LID_i and LID_j to all tag intervals as shown in Eq. 3; we denote $TI_{i,t}$ as the number of tag intervals by LID_i at t, and σ_{OQ} and δ_{OQ} as weight values for LIDProx_OQ(i, j).

$$\text{LIDProx\_OQ}_T(i,j) = \frac{\delta_{OQ}}{\sigma_{OQ}} \times \left(\sum_{t=1}^{T} \left(TI_{i.t} + TI_{j,t} \right) \Big/ \sum_{t=1}^{T} \sum_{a=1}^{n} TI_{a,t} \right) \tag{3}$$

Because of the influence of the tag's flow on LID proximity, we should consider the distribution of tag intervals over time. Equation 4 represents dynamic properties of the tag interval distribution. The difference in the distribution of tag intervals in time domain can be represented by the standard deviation of tag intervals. To apply this property to LID proximity, the variable σ_{OQ} in Eq. 4 is used as the inversely proportional weight to the number of tag intervals. This means that the lower standard deviation indicates that associated distribution of tag intervals is close to the uniform distribution; we denote σ_{OQ} as the standard deviation of tag intervals by LID_i and LID_j and $\overline{TI_i}$ as the average number of tag intervals by LID_i until T.

$$\begin{aligned} \sigma_{OQ} &= \sqrt{\frac{1}{T} \times \sum_{t=1}^{T} \left\{ \left(TI_{i,t} + TI_{j,t} \right) - \left(\overline{TI}_i + \overline{TI}_j \right) \right\}^2} \\ \delta_{OQ} &= \left(\sum_{t=1}^{T} \left(STI_{i,t} + STI_{j,t} \right) \Big/ \sum_{t=1}^{T} \left(TI_{i.t} + TI_{j,t} \right) \right) \times \left(1 \Big/ \sum_{t=1}^{T} OQ_{ij,t} \right) \end{aligned} \tag{4}$$

The hit ratio of tag intervals for an observation query is also the factor determining the LIDProx_OQ(*i*, *j*). As opposed to the standard deviation σ_{OQ}, LID proximity for an observation query should be proportional to the hit ratio of tag intervals. The variable δ_{OQ} in Eq. 4 computes the proportional weight – the hit ratio of tag intervals for OQ_{ij}; we denote $OQ_{ij,t}$ as the number of observation queries for LID_i and LID_j at t and $STI_{i,t}$ as the number of results by LID_i for $OQ_{ij,t}$.

$$\text{LIDProx\_TQ}_T(i,j) = \frac{\delta_{TQ}}{\sigma_{TQ}} \times \left(\sum_{t=1}^{T} \left(TM_{i \text{ to } j,t} + TM_{j \text{ to } i,t} \right) \Big/ \sum_{t=1}^{T} \left(\sum_{a=1}^{n} \sum_{b=1}^{n} TM_{a \text{ to } b,t} - \sum_{c=1}^{n} TM_{c \text{ to } c,t} \right) \right) \quad (5)$$

LID proximity for a trajectory query uses the pattern of tag movements along the FlowPath as the main factor because a trajectory query takes an interest in LIDs where a tag passes at the specified time period. Equation 5 shows the LID proximity function for a trajectory query retrieving tag intervals by LID_i and LID_j. This function, denoted by LIDProx_TQ(*i*, *j*), obtains the simultaneous access probability of LID_i and LID_j through the ratio of tag movements between LID_i and LID_j to the total number of tag movements for all LIDs; we denote $TM_{i \text{ to } j,t}$ as the amount of tag movements from LID_i to LID_j, and σ_{TQ} and δ_{TQ} as weight values for LIDProx_TQ(*i*, *j*).

Similar to the LID proximity function for an observation query, both the tag interval distribution over time and the hit ratio of tag intervals for a trajectory query have an influence on that for a trajectory query. Different with an observation query, however, a trajectory query should consider not the distribution of tag intervals for each individual LID but that of tag intervals between LIDs – the movements of the specified tag. To do this, we define the standard deviation, σ_{TQ}, for computing a degree of the difference in the distribution of tag movements between LID_i and LID_j. We also define the hit ratio of tag intervals by LID_i and LID_j for a trajectory query as δ_{TQ}.

4. Reordering scheme of LIDs

In this section, we define the reordering problem of LIDs based on the LID proximity function and propose the reordering scheme for solving this problem.

Let us assume that there is a set of LIDs, LIDSet = {LID_1, $LID2$, …, LID_{n-1}, LID_n}. To use the LIDSet for the coordinates in the LID domain, an ordered list of LIDs, $OLIDList_i$ = ($OLID_{i.1}$, $OLID_{i.2}$, …, $OLID_{i.n-1}$, $OLID_{i.n}$) should be determined first of all. It is possible to make $n!/2$ combinations of the OLIDList from $OLIDList_1$ to $OLIDList_{n!/2}$. To find out the optimal OLIDList that LID proximity for all LIDs are maximum, we first define the linear proximity as follows.

Definition 2. ***Linear Proximity (LinearProx)*** of $OLIDList_a$($LinearProx_a$) is the sum of LIDProx between adjacent OLIDs for all OLIDs in OLIDLista such that

$$LinearProx_a = \sum_{i=1}^{n-1} \text{LIDProx}(i, i+1) \quad (6)$$

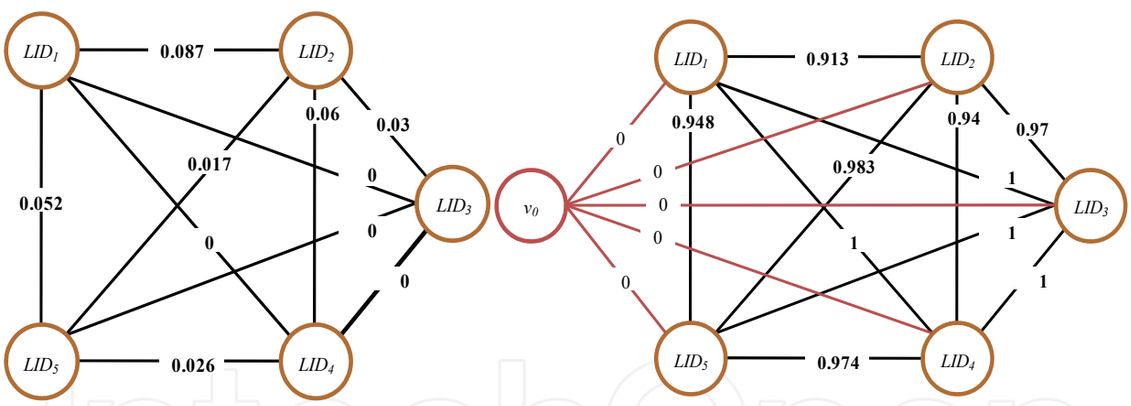
To get the optimal distribution of tag intervals in the domain space, LID proximity between two LIDs should be the maximum for all LIDs. That is, if a query accesses tag intervals generated by the LIDs in the query predicate, corresponding LIDs in the OLIDList should be

ordered closely. As a result, all of LID proximity between adjacent LIDs should also be maximum. With the definition of the linear proximity, we can define the problem for reordering LIDs in order to retrieve the OLIDList which has the maximum access probability as follows.

Definition 3. ***LID reOrdering Problem (LOP)*** is to determine an $OLIDList_o$ = ($OLID_{o.1}$, $OLID_{o.2}$, …, $OLID_{o.n-1}$, $OLID_{o.n}$) for which $LinearProx_o$ is maximum where there is LIDSet = {LID_1, LID_2, …, LID_{n-1}, LID_n} and LID proximity for all LIDs.

To solve the LOP with LID proximity, the graph G is formed by LIDs and their LID proximity values as shown in Fig. 5-(a). The LOP is to find out the optimal OLIDList which has the maximum linear proximity in the graph G according to the Definition 3. In Fig. 5-(a), the optimal $OLIDList_o$ is (LID_5, LID_1, LID_2, LID_4, LID_3) or (LID_3, LID_4, LID_2, LID_1, LID_5) among 60 (5!/2) OLIDLists and its $LinearProx_o$ is 0.199.

The LOP is very similar to the well-known minimal weighted Hamiltonian path problem (MWHP) without specifying the start and termination points. The MWHP finds the Hamiltonian cycle which has a minimal weight in the graph. To apply the LOP to the MWHP, it is necessary to convert the LOP into a minimization problem because the LOP is a maximization problem for finding the order of having maximum LID proximity values for all LIDs. Therefore, the weight value for LID_i and LID_j, $w(i, j)$ in the graph G should be changed to 1 – LIDProx(i, j) or 1 – LIDProx(j, i). The LOP can be treated as a standard traveling salesman problem (TSP) by Lemma 1.



(a) A weighted graph G representing LID proximity between LIDs

(b) The conversion of the graph G into the graph G' for solving the LOP

Fig. 5. An example of a weighted graph for reordering LIDs based on LID proximity

Lemma 1. The LOP is equivalent to the TSP for a weighted graph $G' = (V', E', w')$ such that $V' = V \cup \{v_0\}$ where v_0 is an artificial vertex to solve the MWHP by the TSP

$E' = E \cup \{(LID_i, v_0) \mid LID_i \in \text{LIDSet}\}$

$w' : E \rightarrow R$, $w'(i, j) = 1 - \text{LIDProx}(i, j) = 1 - \text{LIDProx}(j, i) = w'(j, i)$, $w'(i, v_0) = w'(v_0, i) = 0$

Proof: The graph G' contains Hamiltonian cycles because G' is a complete and weighted graph. Assume that a minimal weighted Hamiltonian cycle produced in G' is HC where HC = ((v_0, $OLID_{a.1}$), ($OLID_{a.1}$, $OLID_{a.2}$), …, ($OLID_{a.n-1}$, $OLID_{a.n}$), ($OLID_{a.n}$, v_0)) and $OLID_{a.i} \in$ LIDSet. If two edges, (v_0, $OLID_{a.1}$) and ($OLID_{a.n}$, v_0), containing the vertex v_0 are eliminated from HC, we can get a minimal weighted Hamiltonian path L in G' from $OLID_{a.1}$ to $OLID_{a.n}$. A weight

of *HC* is identical with one of a path *L* because all of edges eliminated in order to produce the path *L* contain the vertex v_0 and weights of these edges are zero. The produced path *L* is translated as an ordered LID list, $OLIDList_a$ where $OLIDList_a$ = ($OLID_{a.1}$, $OLID_{a.2}$, …, $OLID_{a.n-1}$, $OLID_{a.n}$). By this reason, the reordering of LIDs can be defined as a solution of the corresponding TSP for obtaining *HC* in the weighted graph *G'*.

Figure 5-(b) shows an example of the weighted graph *G'* to determine the OLIDList for LIDs in Fig. 5-(a). To apply the WMHP to the LOP, weights of edges are assigned to *w'*, the weight of an edge assigned to one minus LID proximity value. It means that the lower the weight of an edge is, the higher the probability of simultaneously accessing tag intervals generated by the corresponding LIDs of two vertices at each end of the edge is. Since the start and termination points are not determined in the graph *G*, we insert an artificial vertex v_0 and edges from v_0 to all vertices with weight 0 into the graph *G'*. Each Hamiltonian cycle is changed to a Hamiltonian path by removing vertex v_0 in the Hamiltonian cycle with same weight because the weight of all edges incident with v_0 is 0.

Because the TSP is a NP-complete problem, exhaustive exploration of all cases is impractical. To solve the TSP, there have been proposed dozens of methods based on heuristic approaches such as Genetic Algorithms (GA), Simulated Annealing (SA), and Neural Networks (NN). Heuristic approaches, can be used to find a solution for NP-complete problems, takes much less time. Although it might not find the best solution, it can find a near perfect solution – the local optima.

We have used a GA among several heuristic methods to determine the ordered LIDSet by using the weighted graph *G'*. This algorithm has been very successful in practice to solve combinatorial optimization problems including the TSP.

5. Experimental evaluation

We have evaluated the performance of our reordering scheme by applying LIDs as domain values of an index. We also compared it with the numerical ordering scheme of LIDs using a lexicographic scheme. To evaluate the performance of queries, TPIR-tree, R*-tree, and TB-tree are constructed based on the data model for tag intervals with the axes being TID, LID, and TIME. Since indexes use original insert and/or split algorithms, it is possible to preserve essential properties of them.

Since well-known and widely accepted RFID data sets such as the GSTD do not exist, we conducted our experiments with synthetic data sets generated by the Tag Data Generator (TDG). The TDG generates tag events which can be represented as the time-parameterized interval based on the data model for tag intervals. To reflect the real RFID environment, the TDG allows the user to configure its specific variables. All variables of the TDG are based on properties of the FlowPath and tag movements along FlowPaths. According to user-defined variables, tags are created and move between business locations through FlowPaths. The TDG generates a tag interval based on a tag event occurring whenever a tag enters or leaves.

We assigned an LID to each business location by a lexicographic scheme of the TDG based on the spatial distance. To store trajectories of tags over the index, the TDG produces tag intervals from 100,000 to 500,000. Since the LID proximity function uses the quantity for each query, OQ and TQ, as the variable, we should process queries during the TDG produces tag intervals. To do this, we processed 10,000 queries for tracing tags continuously and estimated query specific variables over all periods. Finally, the sequence of LIDs based

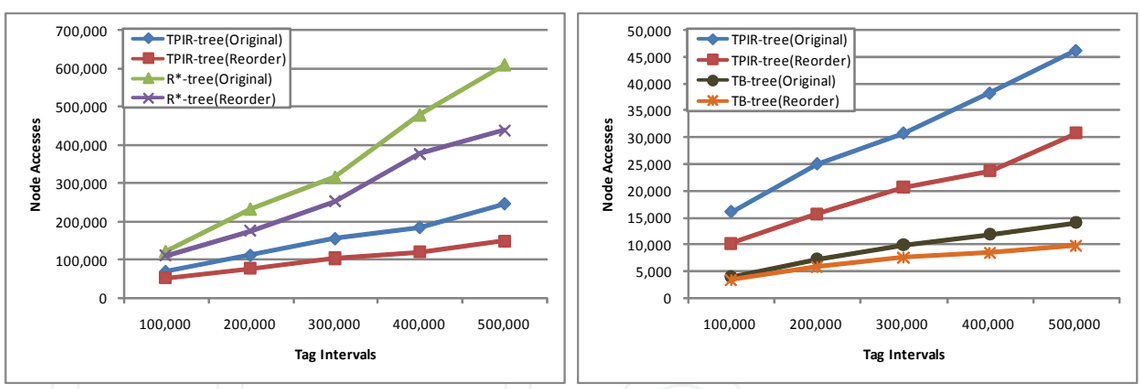
on LID proximity is determined by computing the proximity value between LIDs until all the tag events are produced.

Experiments of this paper used the TDG data set constructed with 200 business locations. To measure average cost, all experiments were performed 10 times for the same data set. In the figures for experimental results, we rename the index by attaching the additional word with a parenthesis in order to distinguish each index according to the arrangement of LIDs. "Original" means the index using the initial arrangement of LIDs on the LID domain. "Reorder" means the index based on LID proximity.

Experiment 1: Measuring the performance of each query type

In this experiment, we attempted to evaluate the performance of queries where only one query type is processed in order to measure the performance of each query type. To obtain the optimized order of LIDs for each query type, we processed 10,000 OQs in Fig. 6-(a) and 10,000 TQs in Fig. 6-(b) before reordering scheme is processed.

Figure 6 shows the performance comparison between "Original" and "Reorder" for each query type. Figure 6-(a) and 6-(b) are related to the performance of OQ and TQ, respectively. Each query set includes 1,000 OQs or TQs. We find out that "Reorder" can retrieve the results with lower cost of node accesses than "Original" for all comparison in Fig. 6. The performance of most "Reorder" is slightly better than the performance of "Original" for the data set of 100,000 tag intervals. Nevertheless, "Reorder" still outperforms "Original" during tag intervals are generated continuously and inserted at the index.



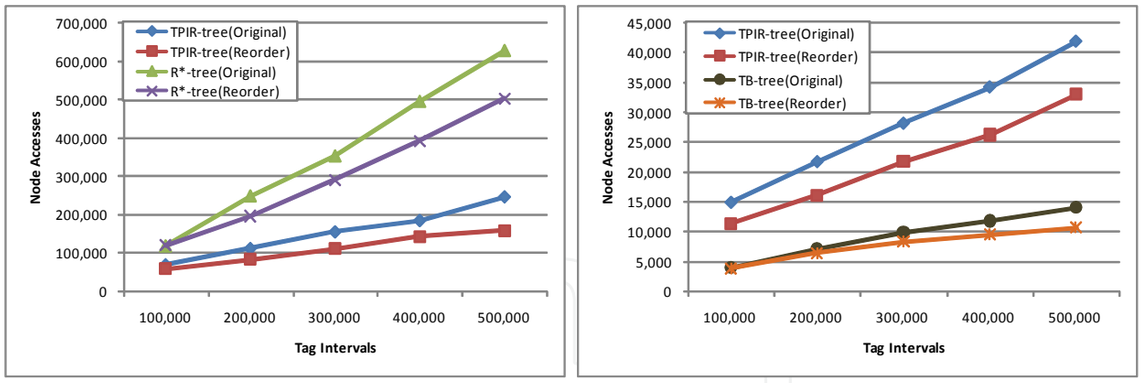
(a) The number of node accesses for OQ (b) The number of node accesses for TQ

Fig. 6. Performance evaluation for indexes where only one type of query is used.

The search performance of OQ and TQ are improved up to 39% and 33%, respectively. This experiment tells us that LID proximity can measure the closeness between business locations more precisely if tag movements and queries happen continuously.

Experiment 2: Performance comparison in case of processing OQ and TQ altogether

Regardless of better performance than an initial arrangement of LIDs, Experiment 1 only evaluates the performance for individual query type. We need to measure the performance in case that OQ and TQ are processed altogether. To do this, we performed the experimental evaluation as shown in Fig. 7. Since LID proximity should reflect properties of all query types together, we processed both of 5,000 OQs and 5,000 TQs before the proximity is measured. Then, 1,000 OQs or TQs are processed for evaluating the performance of each query.



(a) The number of node accesses for OQ (b) The number of node accesses for TQ

Fig. 7. Performance evaluation for indexes when processing both queries altogether

The result of Fig. 7 shows that the number of node accesses of "Reorder" is increased as compared with that in Fig. 6. The reason is that LIDProx_OQ$_T$(i, j) and LIDProx_TQ$_T$(i, j) in Eq. 2 have a negative effect on the performance of a query not related to each proximity under the condition that OQ and TQ are processed together. The performance of "Reorder" is nevertheless better than the performance of "Original" at processing all of OQ and TQ.

6. Conclusions

This paper has addressed the problem of using the location identifier (LID) as the domain value of the index for tag intervals and proposed the solution for solving this problem. The basic idea is to reorder LIDs by the LID proximity function between two LIDs. The LID proximity function determines which an LID to place closely to the specific LID in the domain. By using the LID proximity function, we can find out the distance of two LIDs in the domain so as to keep the logical closeness between tag intervals. Our experiments show that the proposed reordering scheme based on LID proximity considerably improves the performance of queries for tracing tags comparing with the previous scheme of assigning LIDs.

Since LID proximity is computed with the time parameterized properties, it changes over time. Therefore, it is necessary to reorder LIDs periodically or non-periodically for reflecting the changed LID proximity between LIDs. To process queries efficiently over all the time, the reconstruction of the tag interval index should also be required according to changing LID proximity. We are currently developing a dynamic reordering method of LIDs and a restructuring method of the index.

7. References

ChaeHoon , B.; BongHee, H. & DongHyun, K.(2005). Time Parameterized Interval R-tree for Tracing Tags in RFID Systems, International Conference on DEXA, pp.503-513

Dan. L.; HichamG, E.; Elisa, B. & BengChin, O. (2007). Data Management in RFID Applications, International Conference on DEXA, pp.434-444

Darrell, W.(1994). A Genetic Algorithm Tutorial, Statistics and Computing, Vol. 4, pp.65-85

EPCglobal.(2006). EPC Information Services (EPCIS) Specification, Ver. 1.0, EPCglobal Inc.

EPCglobal.(2006). EPC™ Tag Data Standards, Ver. 1.3, EPCglobal Inc.

Fusheng, W. & Peiya, L.(2005). Temporal Management of RFID Data, International Conference on VLDB, pp.1128-1139

HV,J.(1990). Linear Clustering of Objects with Multiple Attributes, ACM SIGMOD, Vol. 19(2), pp.332-342

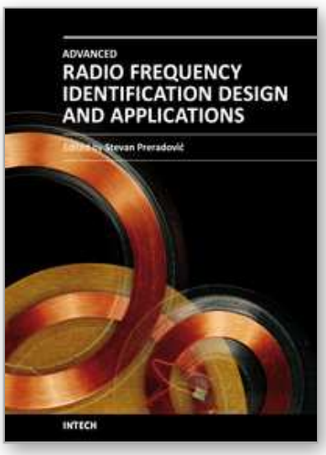
Ibrahim, K.& Christos,F.(1993). On Packing R-trees, CIKM, pp.490-499

Mohamed F, M.; Thanaa M, G.&Walid G,A.(2003). Spatio-temporal Access Methods, IEEE Data Engineering Bulletin, Vol. 26(2), pp.40-49

Mark, H.(2003) EPC Information Service – Data Model and Queries, Technical Report, Auto-ID Center

Steven S, S.(1998). The Algorithm Design Manual, Springer-Verlag, New York Berlin Heidelberg

Yannis, T.;Jefferson R O, S.&Mario A, N.(1999). On the Generation of Spatiotemporal Datasets, International Symposium on Spatial Databases, pp.147-164



Advanced Radio Frequency Identification Design and Applications
Edited by Dr Stevan Preradovic

ISBN 978-953-307-168-8
Hard cover, 282 pages
Publisher InTech
Published online 22, March, 2011
Published in print edition March, 2011

Radio Frequency Identification (RFID) is a modern wireless data transmission and reception technique for applications including automatic identification, asset tracking and security surveillance. This book focuses on the advances in RFID tag antenna and ASIC design, novel chipless RFID tag design, security protocol enhancements along with some novel applications of RFID.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Sungwoo Ahn and Bonghee Hong (2011). Reordering of Location Identifiers for Indexing an RFID Tag Object Database, Advanced Radio Frequency Identification Design and Applications, Dr Stevan Preradovic (Ed.), ISBN: 978-953-307-168-8, InTech, Available from: http://www.intechopen.com/books/advanced-radio-frequency-identification-design-and-applications/reordering-of-location-identifiers-for-indexing-an-rfid-tag-object-database

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen