

Reordering for Indexing in an RFID Tag Database*

SUNGWOO AHN AND BONGHEE HONG

Department of Computer Engineering

Pusan National University

Busan, 609-735 Korea

The trajectories of RFID tags can be modeled as a set of time-parameterized line segments defined by two tag events, namely entering and leaving a specific location. Each tag interval (TI) is specified in a three-dimensional domain, with axes: Tag Identifier (TID), Location Identifier (LID), and time (TIME). The query performance for tracing tags depends on the distribution of the TIs in the data space. We investigate a more efficient representation of TIs, which orders the set of values in the three-dimensional domain. Our analysis shows that the ordering of LIDs makes a greater contribution to the efficiency of query processing than TID or TIME. To execute queries efficiently, an optimal ordering of LIDs is important. To solve this problem, we propose a new *LID proximity function* for reordering an arbitrary LID ordering. We also propose a *reordering scheme for LIDs* to determine an optimal sequence of LIDs in the domain. Our experiments show that the proposed reordering scheme considerably improves the performance of queries for tracing tag locations, compared with previous methods for assigning LIDs.

Keywords: RFID system, tag, middleware, query for tracing tags, location identifier, proximity, ordering

1. INTRODUCTION

Radio Frequency Identification (RFID) is an emerging technology suitable for a wide range of applications, such as automated manufacturing, inventory tracking, and supply chain management. RF technologies enable the identification of individual items in real time by using automatic and rapid identification [5-7]. In addition to real-time identification, RF technologies offer additional advantages in the monitoring of field-based operations by tracking and tracing the location of tags attached to items [2, 7, 8]. For example, a distributor can monitor tags that enter and leave the warehouse to measure product stock levels. In another example, if it is found that the meat sold in a retail store is tainted, the location history of associated tags provides useful information for an epidemiological investigation. By using queries about the trajectories of RFID tags, RFID applications can monitor events about field-based situations and respond to them.

To store and retrieve tag data efficiently, it is important to provide an index for the repository of tag data. EPCglobal [3], a leader in standards management and development for RFID related technologies, proposes an Electronic Product Code (EPC) Information Service (EPCIS) as the repository for tag events [1, 2]. EPCIS is a standard interface for access to and persistent storage of tag information by RFID applications. To support queries on historical information about tags, EPCIS stores timestamped historical data, which are a set of tag events captured by RFID readers. These events contain historical infor-

Received April 11, 2008; revised June 30 & August 27, 2008; accepted October 31, 2008.

Communicated by Chih-Ping Chu.

* This work was supported by the grant of the Korean Ministry of Education, Science and Technology (The Regional Core Research Program/Institute of Logistics Information Technology).

mation about corresponding tags, such as the containment relationship between tags and the location of tags. To examine the history of tag locations, RFID applications can request specific timestamped historical data via EPCIS query interfaces for tracking and tracing tags.

Because there are many tagged items continuously moving between readers and producing tag events, a large amount of historical data is collected in the storage over time. For efficient management of these data, EPCIS usually stores the data in the base table of a database [2, 6, 7]. RFID applications must execute queries on this table whenever they want to retrieve the location history of specific tags. However, it is inefficient to examine all records in the table because of the large amount of historical information that has accumulated in the base table. It is necessary to build an index structure to enable rapid searching of the location information stored in the table.

An index for tracing tags can be constructed that is based on tag events generated when a tag enters and leaves the location of a reader. From the timestamped historical information contained in tag events, an RFID application can use Location Identifier (LID), Tag Identifier (TID), and the identified time (TIME) as predicates for tracking and tracing tags [2, 8]. To index these values efficiently, we can define a *tag interval* (TI) in terms of the two tag events generated when the tag enters and leaves a specific location. This TI is represented and indexed as a time-parameterized line segment in a three-dimensional domain defined by LID, TID, and TIME axes [8].

TIs in a three-dimensional index are sequentially stored and accessed in one-dimensional storage on disk. Because logically adjacent TIs will be retrieved simultaneously during a query, they should not be stored far apart on the disk to minimize the cost of disk accesses. Logical closeness has been studied to determine the distance between domain values representing the coordinates of these objects. An object is logically adjacent to another if the distance between them is the shortest of distances between objects using some distance measuring function. Note that changing the order of the domain values results in different distances between objects because of the different distribution of objects in the data space. To ensure logical closeness between objects, therefore, domain values should be correctly ordered in each domain.

TID is a fixed identifier, related to the EPC [4], for a tagged item. EPC can have three components: *Company*, *Product*, and *Serial*. Because the EPC scheme assigns an identifier to a tag in a hierarchical manner using its three components, TID can imply logical closeness between grouped tags. Naturally, timestamps in the TIME domain can provide chronological closeness between TIs. We usually achieve this closeness by assigning timestamps based on temporal proximity in the TIME domain [14, 15]. LID is not a predefined identifier. There are many numbering schemes for LIDs, such as manufacturing schemes based on the reader's serial number, and lexicographic schemes associated with RFID applications.

Because LID represents the location of a tag's residence or traversal, the LID domain must provide logical closeness for the dynamic flow of tags across RFID locations. The problem is that there is no rule for assigning LIDs to RFID locations in order to ensure this property. If LIDs are arbitrarily ordered in the domain without considering tag flows, TIs will be scattered across the data space irrespective of their logical closeness. Because this situation causes random disk accesses when seeking logically adjacent TIs, the cost of query processing will be large.

To solve this problem, we propose a reordering method for LIDs. The basic idea is to compute the distance between two LIDs to fix the logical closeness between TIs. To do this, we define a proximity function based on a new *LID proximity* (LIDProx) between two LIDs. The proximal distance between LIDs can be computed by tag movements. To determine the LIDProx, we examine the *path of tag flows* (FlowPath), which is generated by tag movements. Then we define the *LIDProx function*, which computes the distance between LIDs. To determine a sequence of LIDs based on LIDProx, we construct a weighted graph and generate the ordered LID set. It is then possible to store logically adjacent TIs in close proximity on the disk because our reordering method ensures a correlation between the distance and the logical closeness of TIs. To show this, we evaluate the performance of the index scheme using LIDs based on LIDProx as domain values. We also compare it with index schemes that use previous ordering schemes for LIDs.

2. RELATED WORK

The benefit of reordering LIDs based on the LIDProx is to enable clustering of logically adjacent TIs to minimize disk seeks during a query. This is similar to previous studies of object clustering using proximity based on a distance measure. In this section, we analyze several studies on object clustering and identify problems with choosing them in a solution to LID reordering. To be able to use an ordered list of LIDs (OLIDList) in a real index structure, we also examine previous work on indexing schemes for retrieving moving objects, including tags.

2.1 Clustering Methods

Considering clustering in general, previous approaches can be classified into two categories: *data clustering* and *page clustering/ordering*.

Data clustering is a technique for partitioning a data set into a number of subsets having common features [9]. Data clustering methods are often classified into hierarchical and partitional methods. Because data clustering must classify objects into different groups based on common traits, choosing the distance (or *similarity*) measure is an important step in any type of data clustering. Examples of distance measure functions are the Euclidean distance, Manhattan distance, and Hamming distance [9, 24].

There are problems with applying data clustering methods to the reordering of LIDs. First, they assume that all domains are properly ordered based on a specific proximity. In our case, it would be necessary to identify the order of LIDs because the LID domain does not provide adequate closeness between TIs in the data space. Second, although these methods determine the logical closeness between objects using a distance measure, this property is only used to determine the similarity between objects and to classify them into clusters. They do not handle the storage of objects on disk for efficient query processing.

Another approach to object clustering is page clustering/ordering. Page clustering and page ordering methods aim to reduce the cost of disk access during a query by clustering pages or determining the order of pages. They mainly relate to the clustering of pages for storing multidimensional data on disk. Page clustering [10, 11] involves storing

spatially adjacent objects in the same page or in multipage clusters to reduce the number of page accesses. Page ordering [12, 13] is concerned with the ordering of pages of two-dimensional spatial data in a one-dimensional storage medium that reduces the number of disk seeks. To determine the sequence of pages or objects on disk, page clustering/ordering methods use spatial proximity as the distance measure.

As for data clustering methods, these methods do not involve determining the order of domain values. Most work on page clustering/ordering has applied the spatial domain to the data space. Because the spatial domain provides the spatial proximity between spatial coordinates, they need not be concerned with reordering domain values. This is a significant difference between page clustering/ordering and LID reordering methods.

2.2 Indexing Methods for Moving Objects

As moving objects report their locations periodically, moving-object databases keep histories of the spatiotemporal coordinates of each moving object. They should support queries for retrieving the trajectories and positions of continuously moving objects. For efficient query processing about the historical positions of moving objects, there have been various indexing-scheme approaches, including the 3D R-tree [14], the HR-tree [20], and the MV3R-tree [16]. These indexing schemes are often called *locality-based indexing schemes*. Most of them are developments of the R-tree [18]. The R-tree structure is a hierarchy of possibly overlapping index regions that encloses child nodes in a Minimum Bounding Box (MBB). The MBB includes all child node entries in an n-dimensional domain. Because of their hierarchical structure based on locality, these methods show good performance for range queries and time-slice queries. On the other hand, the locality property leads to the traversal of a large number of index paths in trajectory queries (TQs).

To enhance the query performance for retrieving trajectories, *trajectory-preserving indexing schemes* have been devised, including the TB-tree [15] and the OP-tree [17]. These indexing schemes store only one trajectory per leaf node. This property enables these schemes to outperform locality-based indexing schemes in processing TQs. Because trajectory preservation produces much dead space between the MBBs of nonleaf nodes, however, the overlap between nodes is large. This results in poorer performance in range queries than in that of locality-based indexing schemes.

Previous indexing schemes can be constructed for tracing tag trajectories because tags move continuously in the same way as moving objects. However, they cannot find tags that enter but do not leave a reader's locality. To solve this, [8] proposed a data model of a tag's trajectory. Because the data model represents tag trajectories as time-parameterized intervals whose lengths are time dependent, it is possible to retrieve a tag that remains at a reader, irrespective of the index scheme. In addition, [8] proposed an index scheme called the Time Parameterized Interval R-tree (TPIR-tree), based on the R*-tree [19], for storing and retrieving tag trajectories efficiently. By devising insert and split algorithms for the time-parameterized intervals, the TPIR-tree had an improved query performance compared with previous index schemes for moving objects.

3. PROBLEM DEFINITION

In this section, we describe tag events generated by tag movement in the RFID system. After defining the data model and queries for tracing tags, we discuss the problems of ordering LID values in the domain of an index structure.

3.1 Target Environment

Whenever the tag attached to an item traverses the interrogation zone of an RFID reader, the reader collects the tag's information. In an RFID middleware system, the information gathered is represented as *EPCIS tag events* and stored in persistent storage to answer tag-related queries [1, 2]. Because a tag event contains several elements of time-stamped historical information, as shown in Table 1, it can represent the dynamic flow of tagged items between RFID locations along tag routes. If an RFID application needs a history of these items, a query processor can respond to the application by retrieving suitable tag events from a repository.

Table 1. Information included in EPCIS tag events [1].

Semantics	Captured Information
What	Identifier of tag (EPC), quantity of identified tags, type of business transaction
When	Identified time
Where	Identifier of location a tag traverses or resides in
Why	Identifier of business step and disposition

For the timestamped historical information in Table 1, a query processor usually employs TID, LID, and TIME as the predicates of queries for tracing tag locations [2, 8]. For example, consider an application making a query such as "Search the trajectory of the tag #101 from 01/20/2008 06:00 to 02/10/2008 22:00." To process this query, a query processor must retrieve the location identifiers that tag #101 traversed within the specified period.

Note that RFID locations are different from spatial locations, which represent real positions on the map. There are two types of location related to EPCIS tag events, depending on the business perspective for RFID locations. One is the physical position, denoted by the *Read Point* (RP), which identifies the tag. The other is the region, denoted by the *Business Location* (BizLoc), where the tag resides. BizLoc represents the place where a tag is assumed to be until a later tag event occurs with a different BizLoc. Because most RFID applications trace the business flow of tagged items, they are more concerned with BizLoc than RP as the location type of the tag [1]. Therefore, it is natural to use BizLoc as the LID predicate for tracing tag locations.

An EPCIS tag event can be modeled as a time-parameterized interval in a three-dimensional domain, with axes LID, TID, and TIME. We denote this interval as the TI. The TI is a line segment that connects the two coordinate points generated when the tag enters and leaves a specific BizLoc. If the tag object TID_i enters and then remains at the BizLoc $BizLoc_j$ from $TIME_{enter}$ to $TIME_{leave}$, the corresponding TI is the line from $(TID_i, BizLoc_j, TIME_{enter})$ to $(TID_i, BizLoc_j, TIME_{leave})$.

For example, let us assume that there is a warehouse $BizLoc_b$ with two RPs, as shown in Fig. 1 (a). $RP_{b,1}$ is the receiving dock whereby tagged items can enter $BizLoc_b$ via the transit area $BizLoc_a$. $RP_{b,2}$ is the shipping dock for leaving $BizLoc_b$ to transfer items via $BizLoc_c$. If TID_i enters and leaves $BizLoc_b$ at t_{enter} and t_{leave} , respectively, the TIs $TI_{i,2}$ and $TI_{i,3}$ can be represented in a three-dimensional domain, as shown in Fig. 1 (b). Because TID_i continuously traverses $BizLoc_c$ at the current time, t_{now} , $TI_{i,3}$ must be the TI for which the time of leaving $BizLoc_c$ is the current time [8]. In this manner, the trajectory of a tag is represented as a set of TIs associated with the tag.

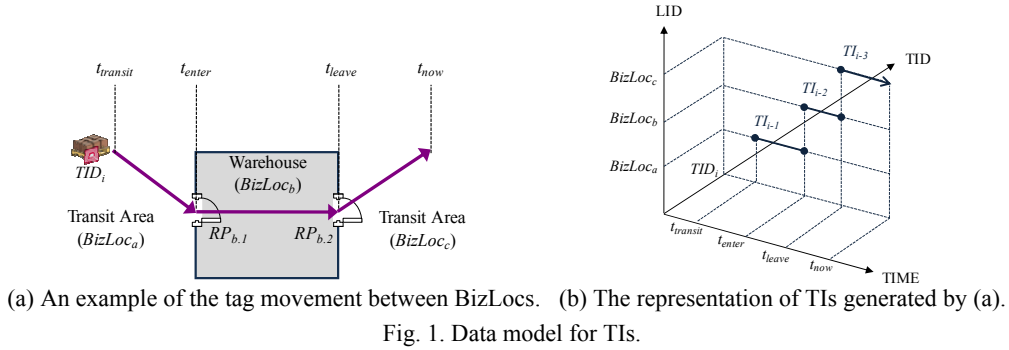


Table 2. Query classification for tracing tag locations.

Predicate			Query Results	Query Types
LID	TID	TIME		
Point/Set/Range	*	Point/Range	TID(s)	Observation Query (OQ)
*	Point/Set/Range	Point/Range	LID(s)	Trajectory Query (TQ)

Queries for tracing tags are classified into two types according to the type of restricted predicate, as shown in Table 2. An *Observation Query* (OQ) is used to retrieve tags that are identified by specified BizLocs within the specified period. A TQ is used to retrieve those BizLocs that the specific tag enters and leaves within the specified period. Queries in Table 2 can be extended to form combined queries by performing two queries: OQ followed by TQ.

To support the rapid retrieval of desired tag trajectories, they should be stored and retrieved by using an index structure. Fig. 2 shows an example of the indexing scheme based on the TPIR-tree proposed by [8]. Each leaf node of the index references logically adjacent TIs in the data space by using an MBB spanning R1 to R5. Then the TIs referenced by index nodes are sequentially stored in and accessed from one-dimensional disk storage. TIs in each leaf node are stored in the same disk page to minimize disk seeks.

3.2 A Problem with LID as the Domain Value

Logical closeness between TIs is very important for simultaneous access during a query. It greatly affects the performance of query processing because the cost of disk access depends on the storage sequence on disk of TIs. For example, assume that a query Q_i

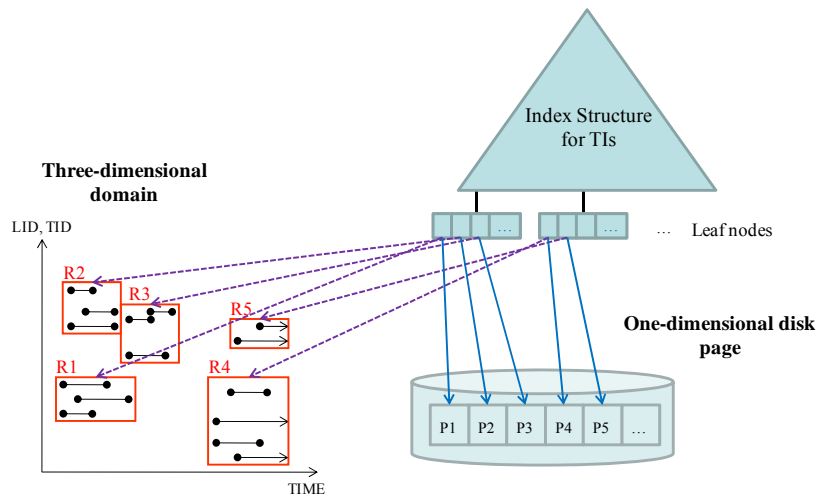


Fig. 2. Indexing scheme for storing and retrieving TIs.

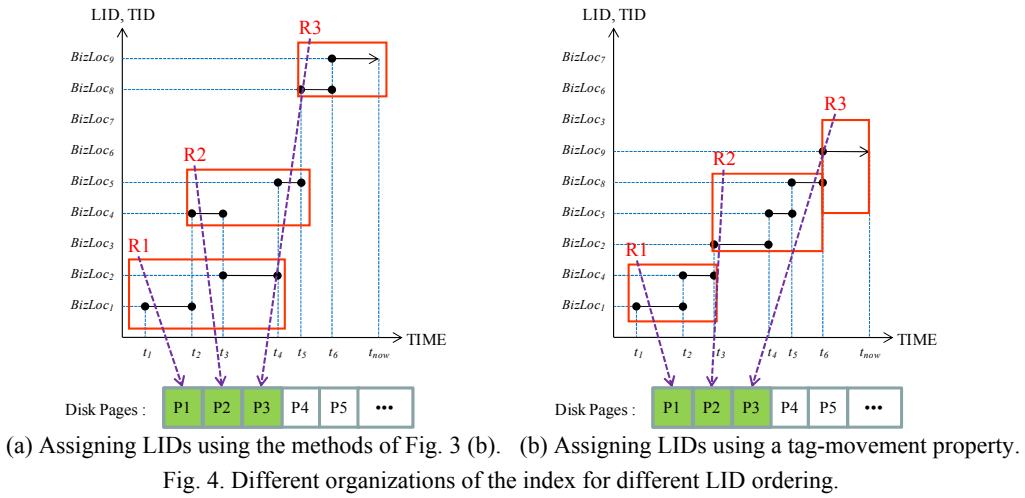
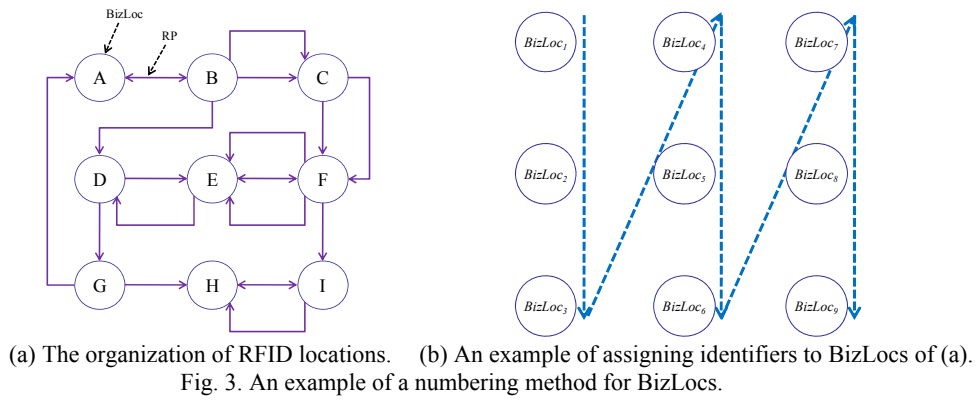
searches TIs using the index in Fig. 2. If all TIs accessed by Q_i are referenced by R3, and then stored in P3, the query processor only needs to access one disk page, namely P3. However, if these TIs are dispersed across disk pages P2, P3, and P5, the query processor usually incurs the additional cost of accessing two extra pages, namely P2 and P5.

The distance between two TIs in the data space must be computed to measure their logical closeness. If the nearest TI to a specific TI has the shortest distance to that TI, this implies that the distance measuring function will ensure logical closeness between TIs. The distance is normally measured using the proximity between domain values in the data space.

A tag produces a dynamic flow while moving between BizLocs. Because a query for tracing tags gives the tag's traces, the LID domain must provide closeness information about the TIs in tag movements. In contrast to TID, LID is not a predefined identifier. We can assign BizLocs to LIDs by using various numbering methods. For example, it can be a lexicographic method for measuring the distance in an RFID application system. It is also possible to apply a spatial distance measure, such as the Hilbert curve, Z-ordering, or Row-Prime curve [12, 13]. Fig. 3 shows an example of a LID numbering method for describing BizLocs and RPs.

Despite the existence of various LID numbering methods, there is the problem that they lack any inherent proximity property that might provide logical closeness information related to the dynamic flow of tags. If LIDs are assigned to BizLocs without considering a tag's flows, a leaf node of the index may reference TIs irrespective of their logical closeness. This implies that the index structure cannot guarantee that a query processor will retrieve results at minimal cost because logically adjacent TIs may be stored on disk pages that are remote from each other.

This situation is illustrated in Fig. 4. Assume that a tag, TID_m , traverses the BizLocs of Fig. 3 (a) in the order: A, B, D, E, F, and I. If the LIDs are ordered in the LID domain using the ordering of Fig. 3 (b), the TIs are distributed in the data space and stored in disk pages as shown in Fig. 4 (a). Let $TQ_i = (*, TID_m, [t_3^{\dagger}, t_6^{\dagger}])$ be the TQ for searching LIDs for TID_m residence during the period t_3 to t_6 . When TQ_i is processed using the index or-



ganized as shown in Fig. 4 (a), the query processor must access disk pages P1, P2, and P3 because all TIs generated during the period t_3 to t_6 are dispersed among MBBs R1, R2, and R3. However, if we reorder LIDs based on the order of TID_m 's movements, as shown in Fig. 4 (b), the TIs during the period t_3 to t_6 are referenced by the one leaf node associated with MBB R2. The query processor only needs to access page P2 to process TQ_i using the index of Fig. 4 (b).

The basic idea for solving this problem is to use a LIDProx function to reorder LIDs in the domain. To achieve this, we define a LIDProx between LIDs that is based on the dynamic flow of tags.

4. PROXIMITY BETWEEN LOCATION IDENTIFIERS

In this section, we first examine the path of tag flows, which is very important for determining proximity between LIDs. Using the properties of this path, we define LID-Prox and propose a LIDProx function.

4.1 LIDProx Based on the Path of Tag Flows

Tagged items always move between BizLocs by traversing RPs placed at the entrance of each BizLoc. If there are no RPs connecting specified BizLocs, the tagged item cannot move directly between two BizLocs. Even though RPs may exist between two particular BizLocs, tag movement may be restricted because of a business process in a system application. Based on these restrictions, there is a predefined path that a tag may traverse. We designate this path as the FlowPath. Items attached to tags generate a flow of tags traversing the path. The FlowPath from LID_i to LID_j is denoted as $FlowPath_{i \rightarrow j}$.

The FlowPath is a simple method for representing the connection property between two BizLocs. It is possible to generate a FlowPath via a connected graph of BizLocs and RPs as shown in Fig. 5. To achieve this, $BizLoc_1$ to $BizLoc_6$ in Fig. 5 (a) correspond to location identifiers LID_1 to LID_6 in Fig. 5 (b), respectively. If one or more RPs connect two particular BizLocs, they are represented by a single line connecting two LIDs.

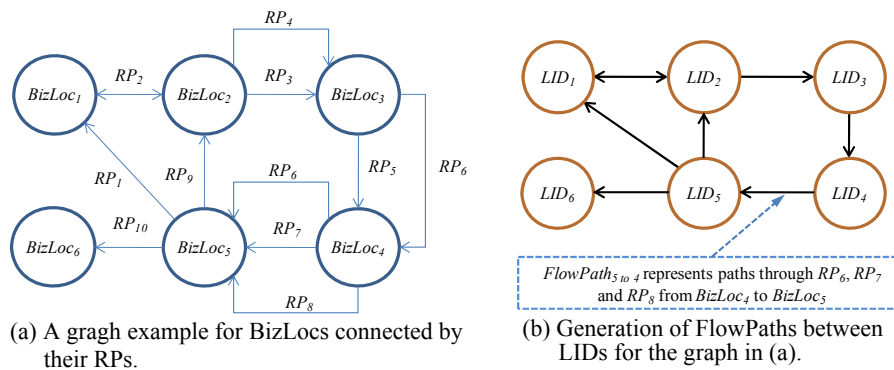


Fig. 5. An example of generating FlowPaths for BizLocs and their RPs.

The FlowPath is a simple method for representing the connection property between two BizLocs. It is possible to generate a FlowPath via a connected graph of BizLocs and RPs as shown in Fig. 5. To achieve this, $BizLoc_1$ to $BizLoc_6$ in Fig. 5 (a) correspond to location identifiers LID_1 to LID_6 in Fig. 5 (b), respectively. If one or more RPs connect two particular BizLocs, they are represented by a single line connecting two LIDs.

The representation FlowPath is the basic structure determining the characteristics of tag movements because a tag moves between BizLocs along FlowPaths. As mentioned in section 3, a query for tracing tags is concerned with the historical change of locations for the specific tag. This implies that TIs generated by BizLocs along a specific FlowPath have a higher probability of simultaneous access than others do. Therefore, it is necessary to reorder LIDs based on FlowPath properties. We first define **LIDProx** as the closeness value between two LIDs in the LID domain of an index. We denote the LIDProx between LID_i and LID_j as $LIDProx_{ij}$ or $LIDProx_{ji}$. The LIDProx between two LIDs has the following properties.

- Any LID_i in the LID domain must have a LIDProx for any LID_j , where $i \neq j$.
- $LIDProx_{ij}$ is equal to $LIDProx_{ji}$, for all LIDs.
- If there is no LID_k for which $LIDProx_{ij} < LIDProx_{ik}$, the closest LID to LID_i is LID_j .

4.2 LIDProx Function

We propose a LIDProx function that measures the closeness between two LIDs. The LIDProx function has a static version based on RP properties, and a dynamic version based on FlowPaths.

4.2.1 Static LIDProx (SLIDProx) function

The SLIDProx function uses the *FlowPath Cardinality* (FC) for computing the closeness between two LIDs. The FC represents the number of directions for entering and leaving two LIDs. If a BizLoc has many more pathways for tag movement than others have, it is likely to accept many more tag traversals at once than others are. Therefore, the FC controls the number of TIs generated by two BizLocs connected by the specified FlowPath. Using the number of directions between LID_i and LID_j , there is an FC from LID_i to LID_j , denoted by $FC_{i \rightarrow j}$, and/or one from LID_j to LID_i , denoted by $FC_{j \rightarrow i}$. We can compute the FC between LID_i and LID_j as the sum of $FC_{i \rightarrow j}$ and $FC_{j \rightarrow i}$.

We define the *SLIDProx function* in Eq. (1), where $i \neq j$ and $FC_{i \rightarrow i} = 0$ for any LID_i in a set of LIDs (LIDSet). We denote $SLIDProx(i, j)$ as the SLIDProx function for LID_i and LID_j .

$$SLIDProx(i, j) = \left(FC_{i \rightarrow j} + FC_{j \rightarrow i} \right) / \sum_{a=1}^n \left(\sum_{b=1}^n FC_{a \rightarrow b} \right) \quad (1)$$

$SLIDProx(i, j)$ in Eq. (1) computes the proximity between LID_i and LID_j using the percentage of $FC_{i \rightarrow j}$ and $FC_{j \rightarrow i}$ to FCs for all existing LIDs. The higher the sum of FCs for two LIDs, the closer they are in the domain. Fig. 6 shows an example of the closeness between LIDs using SLIDProx. If there are RPs connecting LID_i , LID_j , and LID_k from RP_1 to RP_5 , it is possible to obtain six FCs. Because $LIDProx_{ij}$ is 0.5 (4/8), $LIDProx_{jk}$ is 0.125 (1/8), and $LIDProx_{ki}$ is 0.375 (3/8) using Eq. (1), we find that the closeness value is highest between LID_i and LID_j .

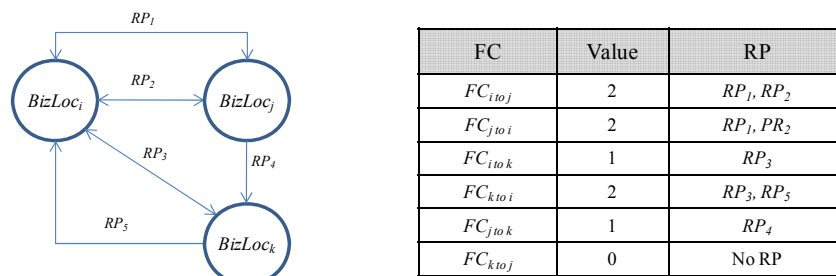


Fig. 6. An example of applying the SLIDProx function.

4.2.2 Dynamic LIDProx (DLIDProx) function

The pattern of tag movements along FlowPaths and the frequency of their related

queries change continuously over time. Consequently, the access probability of TIs generated by any two LIDs will also change. Even if the SLIDProx function reflects the characteristics of tag movements and queries using FlowPath properties, their dynamic properties are still not handled by this function.

We define the *DLIDProx function* in Eq. (2). We denote T as the time of computation of the LIDProx, $DLIDProx_T(i, j)$ as the DLIDProx function at time T , and $LIDProx_OQ(i, j)$ and $LIDProx_TQ(i, j)$ as proximity functions generated by properties of an OQ and TQ, respectively.

$$DLIDProx_T(i, j) = \alpha \times LIDProx_OQ_T(i, j) + (1 - \alpha) \times LIDProx_TQ_T(i, j) \quad (2)$$

$DLIDProx_T(i, j)$ is a time-parameterized function: the closeness value between LID_i and LID_j changes over time. To reflect the closeness value for an OQ and TQ simultaneously, the function calculates a weighted sum of $LIDProx_OQ_T(i, j)$ and $LIDProx_TQ_T(i, j)$. The weight α reflects the proportion of OQs among all queries, as shown in Eq. (3). We denote $OQ_{ij,t}$ as the number of OQs for LID_i and LID_j at time t and $TQ_{ij,t}$ as the number of TQs for LID_i and LID_j at time t .

$$\alpha = \frac{\sum_{t=1}^T OQ_{ij,t}}{\sum_{t=1}^T (OQ_{ij,t} + TQ_{ij,t})} \quad (3)$$

The LIDProx for an OQ is affected by the number of TIs generated by two LIDs, which are predicates of the OQ. In order to examine the effect of this, assume that the ratios of generated TIs per LID over the entire period are $\sum_{t=1}^T TI_{i,t} > \sum_{t=1}^T TI_{j,t} > \sum_{t=1}^T TI_{k,t}$, where $TI_{i,t}$, $TI_{j,t}$, and $TI_{k,t}$ are the number of TIs from LID_i , LID_j , and LID_k at t , respectively. If an application processes OQs that access two LIDs simultaneously, queries for LID_i and LID_j usually retrieve more TIs than other queries because of the difference in the number of TIs. This implies that TIs generated by LID_i and LID_j must be in close proximity in the domain space.

Here, $LIDProx_OQ_T(i, j)$ computes the LIDProx for an OQ with the ratio of TIs generated by LID_i and LID_j to all TIs, as shown in Eq. (4). We denote σ_{OQ} and δ_{OQ} as the weight values for $LIDProx_OQ_T(i, j)$.

$$LIDProx_OQ_T(i, j) = \frac{\delta_{OQ}}{\sigma_{OQ}} \times \left\{ \frac{\sum_{t=1}^T (TI_{i,t} + TI_{j,t})}{\sum_{t=1}^T \sum_{a=1}^n TI_{a,t}} \right\} \quad (4)$$

In addition to the number of TIs, we should consider the distribution of TIs over time. An example of this is illustrated in Fig. 7. Assume that the distributions of TIs from LID_j , LID_k , and LID_i simultaneously with LID_i are (a), (b), and (c) from t_1 to t_n , respectively. All distributions have the same mean value M during the entire period. Consider the case where an application processes three OQs, OQ_{ij} , OQ_{ik} , and OQ_{il} , which have the same time predicate for the TI dataset, as shown in Fig. 7. If the time predicate is assigned to the value close to t_i , the closest LIDs are LID_i and LID_l because OQ_{il} retrieves more results than OQ_{ij} and OQ_{ik} . Otherwise, the closeness between LID_i and LID_l is lower

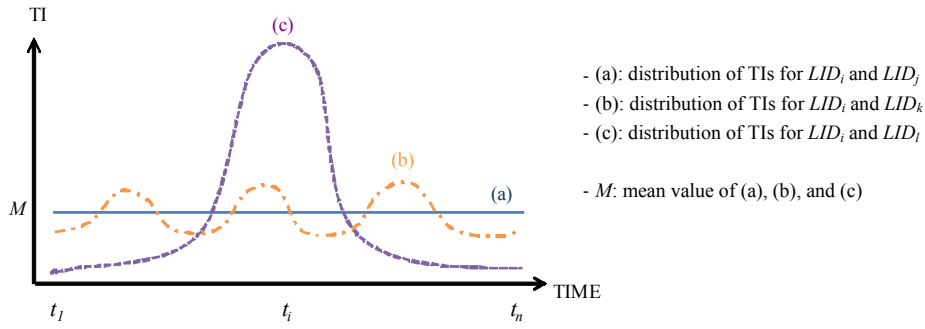


Fig. 7. Examples of TI distributions over time.

than that of other pairs because the number of results for OQ_{il} is less than OQ_{ij} and OQ_{ik} . In the case of distribution (a) for OQ_{ij} , the regular closeness is guaranteed for LID_i and LID_j because its TIs are distributed uniformly across the entire period.

Eq. (5) represents the above properties of the TI distribution. The difference in the distribution of TIs in the time domain can be represented by the standard deviation of TIs. This means that a lower standard deviation indicates that the associated distribution of TIs is close to a uniform distribution, similar to (a) in Fig. 7. We denote σ_{OQ} as the standard deviation of TIs from LID_i and LID_j , and \bar{TI}_i as the average number of TIs from LID_i until T .

$$\sigma_{OQ} = \sqrt{\frac{1}{T} \times \sum_{t=1}^T \{(TI_{i,t} + TI_{j,t}) - (\bar{TI}_i + \bar{TI}_j)\}^2} \tag{5}$$

The ratio of TIs included in results for an OQ to all TIs is also a factor determining $LIDProx_{OQ}(i, j)$. If an RFID application predominantly searches TIs generated by LID_i and LID_j around the time period t_i , the query processor retrieves more TIs in distribution (c) than TIs in distributions (a) or (b), as shown in Fig. 7. Of course, if TIs for LID_i and LID_j are retrieved more frequently than TIs for other LIDs over all periods, the proximity value between LID_i and LID_j should be closer than for the others. However, if an application rarely requests tag intervals from LID_i and LID_j , TIs generated by LID_i and LID_j will not be in close proximity in the domain space.

In contrast to the standard deviation σ_{OQ} , therefore, the $LIDProx$ must be proportional to this ratio. The variable δ_{OQ} in Eq. (6) computes the proportional weight: the percentage of TIs included in results of an OQ_{ij} for all TIs generated by LID_i and LID_j . We denote $OQ_{ij,t}$ as the number of OQs for LID_i and LID_j at t and STI_i as the number of TIs retrieved by OQ_{ij} in TIs generated by LID_i until T .

$$\delta_{OQ} = \left\{ \frac{(STI_i + STI_j)}{\sum_{t=1}^T (TI_{i,t} + TI_{j,t})} \right\} \times \left(\frac{1}{\sum_{t=1}^T OQ_{ij,t}} \right) \tag{6}$$

The $LIDProx$ for a TQ must consider the pattern of tag movements along FlowPaths because a TQ is concerned with LIDs traversed by a tag in the specified period. To com-

pute the LIDProx with this property, consider the following. The TQ attempts to trace the location of a tag residing in LID_i at a specified time. If the tag TID_o did not move from LID_i to LID_j , the query does not access LID_i and LID_j simultaneously, irrespective of the number of TIs by these LIDs. By contrast, the query for TID_o probably accesses two LIDs simultaneously if most tags that have resided in LID_i are moving to LID_j . Therefore, the LIDProx for a TQ is affected by the tag movement rather than the number of TIs per LID.

Eq. (7) shows the LIDProx function for a TQ retrieving TIs from LID_i and LID_j . This function, denoted by $LIDProx_TQ_T(i, j)$, obtains the simultaneous access probability of LID_i and LID_j via the ratio of tag movements between LID_i and LID_j to the total number of tag movements for all LIDs. We denote $TM_{i\ to\ j,t}$ as the number of tag movements from LID_i to LID_j , and σ_{TQ} and δ_{TQ} as weight values for $LIDProx_TQ_T(i, j)$.

$$LIDProx_TQ_T(i, j) = \frac{\delta_{TQ}}{\sigma_{TQ}} \times \frac{\sum_{t=1}^T (TM_{i\ to\ j,t} + TM_{j\ to\ i,t})}{\sum_{t=1}^T \left(\sum_{a=1}^n \sum_{b=1}^n TM_{a\ to\ b,t} - \sum_{c=1}^n TM_{c\ to\ c,t} \right)} \quad (7)$$

In contrast to an OQ, a TQ is not related to the distribution of TIs per individual LID, but to that of TIs between LIDs: the movements of the specified tag. To achieve this, we can define the standard deviation σ_{TQ} as computing the degree of difference in the distribution of tag movements between LID_i and LID_j . We can also define the ratio of TIs included in results for a TQ to all TIs generated by LID_i and LID_j as δ_{TQ} .

5. REORDERING SCHEME FOR LOCATION IDENTIFIERS

In this section, we define the reordering problem for LIDs in terms of the LIDProx function, and propose a reordering scheme for solving this problem.

To use the $LIDSet = \{LID_1, LID_2, \dots, LID_{n-1}, LID_n\}$ for the coordinates in the LID domain, $OLIDList_i = (OLID_{i,1}, OLID_{i,2}, \dots, OLID_{i,n-1}, OLID_{i,n})$, must be determined initially. It is possible to make $n!/2$ OLIDList combinations, from $OLIDList_1$ to $OLIDList_{n!/2}$. To discover the optimal OLIDList, for which the LIDProx for all LIDs is a maximum, we define the **linear proximity of OLIDList_a (LinearProx_a)** as the sum of the LIDProxs between adjacent OLIDs for all OLIDs in $OLIDList_a$ such that:

$$Linear\ Prox_a = \sum_{i=1}^{n-1} LIDProx(a.i, a.i + 1). \quad (8)$$

To obtain the optimal distribution of TIs in the domain space, the LIDProx must be the maximum for all LIDs. That is, if a query accesses TIs generated by LIDs in the query predicate, corresponding LIDs in the OLIDList must be ordered closely. Therefore, all LIDProxs of adjacent LIDs must be a maximum. To retrieve the OLIDList with the maximum access probability using the linear proximity, we can define the **LID reordering problem (LOP)** as determining an $OLIDList_o$ for which $LinearProx_o$ is a maximum.

Solving LOP is very similar to solving the well known Minimal Weighted Hamilto-

nian Path Problem (MWHP) without specifying the start and termination points in a graph structure. The MWHP involves finding the Hamiltonian cycle with 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: finding the ordering with maximum LIDProx values for all LIDs. Therefore, the weight value for LID_i and LID_j should be changed to $1 - \text{LIDProx}(i, j)$ or $1 - \text{LIDProx}(j, i)$. To solve the LOP using LIDProx, we can construct a weighted graph $G = (V, E, w)$, where:

- $V = \text{LIDSet} \cup \{v_0\}$, where v_0 is an artificial vertex for solving the MWHP by the TSP,
- $E = \{(LID_i, LID_j) \mid LID_i, LID_j \in \text{LIDSet}, i \neq j\} \cup \{(LID_i, v_0) \mid LID_i \in \text{LIDSet}\}$,
- $w: E \rightarrow R, w(i, j) = 1 - \text{LIDProx}(i, j) = w(j, i), w(i, v_0) = w(v_0, i) = 0$.

The LOP is equivalent to a standard Traveling Salesman Problem (TSP) for the graph G because it is known that the MWHP can be treated as the TSP. 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 $HC = ((v_0, OLID_{a.1}), (OLID_{a.1}, OLID_{a.2}), \dots, (OLID_{a.n-1}, OLID_{a.n}), (OLID_{a.n}, v_0))$ and $OLID_{a.i} \in \text{LIDSet}$. 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 obtain a minimal weighted Hamiltonian path L in G from $OLID_{a.1}$ to $OLID_{a.n}$. The weight of a minimal weighted Hamiltonian cycle HC is identical to a minimal weighted Hamiltonian path L because all edges eliminated in order to produce the path L contain the vertex v_0 , and the weights of these edges are zero. The path produced, L , is translated into an OLIDList, $OLIDList_a$. Therefore, the reordering of LIDs is defined as a solution of the corresponding TSP in the weighted graph G .

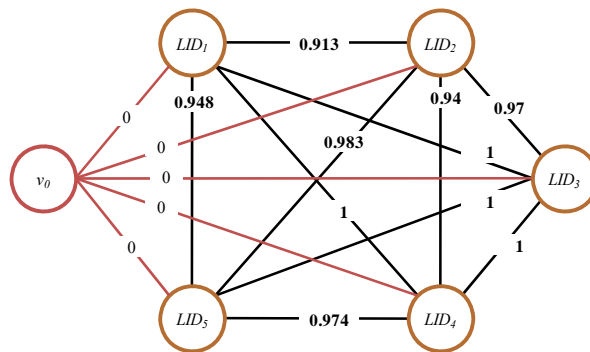


Fig. 8. An example of a weighted graph G for solving the LOP.

Fig. 8 shows an example of a weighted graph G used to determine OLIDList for LIDs. To apply the MWHP to the LOP, the weights of edges are assigned the value w , which is one minus the LIDProx value. This implies that the lower the weight of an edge, the higher the probability of simultaneously accessing TIs generated by the LIDs corresponding to the two vertices at each end of the edge. To determine the start and termination points, we insert an imaginary 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 remov-

ing the vertex v_0 in the Hamiltonian cycle with the same weight because the weight of all edges incident to v_0 is 0.

The TSP is the most complex of NP-complete problems [23]. If the graph G has n vertices, there are $n!$ possible Hamiltonian cycles and the same number of OLIDLists derived by removing vertex v_0 in each Hamiltonian cycle. Testing every possibility for an n -vertex tour is impractical because it involves an exhaustive exploration of all cases. For example, a 50-vertex tour involves measuring 3.04×10^{64} different Hamiltonian cycles, which would require an impractical computation time.

To solve the TSP, dozens of methods based on heuristic approaches have been proposed, including Genetic Algorithms (GAs), Simulated Annealing, and Neural Networks. A heuristic approach can be used to find solutions to NP-complete problems in reasonable time. Although it might not find the best solution, it can find a nearly perfect solution: the local optima [12, 21, 22]. We used GA [21], among several heuristic methods, to determine the OLIDList using the weighted graph G . This algorithm has been very successful in practice for solving combinatorial optimization problems, including the TSP.

6. EXPERIMENTAL EVALUATION

We evaluated the performance of our reordering scheme by applying LIDs as domain values in an index. We compared it with previous ordering schemes involving LIDs. To evaluate the performance of queries for tracing tags, a TPIR-tree [8], an R*-tree [19], and a TB-tree [15] were constructed, with axes TID, LID, and TIME. As an experimental parameter value, we set the page size for leaf and nonleaf nodes to 2,048 bytes. With this page size, the fan-out for all methods was 72 for nonleaf nodes and 61 for leaf nodes. The performance studies were conducted using Java implementations with the JDK 5.0 library.

6.1 Datasets

Because well known and widely accepted RFID datasets for experimental purpose do not exist, we carried out our experiments using synthetic datasets generated by the Tag Data Generator (TDG). The TDG allows the user to configure its specific variables, including the number of BizLocs and RPs, the connection properties of each BizLoc, the number of tags, and the frequency of tag movements per timestamp. Tags are created and move between BizLocs via FlowPaths. They are iteratively produced and consumed at specific BizLocs. For all experiments, we organized an RFID environment, denoted by *Biz200*, which comprises 200 BizLocs connected by their RPs. To explore various FC cases, we made the TDG randomly select each FC value from the range 0 to 5.

Table 3 shows the datasets generated by the TDG for *Biz200* to produce TIs. Because our method of reordering LIDs changes the arrangement of domain values, it is difficult to make a specific distribution of TIs in the data space. To make the TIs in Table 3 distribute uniformly, the movement of tags is always ruled by a random distribution based on the connection property between BizLocs. If the datasets were randomly distributed initially, they will remain randomly distributed over time. By this rule, we achieved an unbiased spread of TIs from 100K to 500K, as shown in Table 3.

Table 3. Datasets generated by the TDG.

Type	Number of Initial Tags	Total Number of Tags	Number of TIs
100K	200	1,000	100,462
200K	200	1,000	200,297
300K	200	1,000	300,274
400K	200	1,000	400,921
500K	200	1,000	500,378

6.2 Performance of SLIDProx

We first evaluated the performance of indices that reorder LIDs using SLIDProx. To compare our method with previous ordering schemes for LIDs, we used three methods for initially assigning LIDs to BizLocs in *Biz200*: lexicographic ordering based on the reader deployment, *Z*-ordering based on the spatial distance, and random ordering. We denote these LID lists as Deploy, Spatial, and Random, respectively. We rearranged the three initial LID lists into OLIDLlists using SLIDProx. We denote these LID lists as Deploy-(Static), Spatial(Static), and Random(Static). To measure the performance for each query type, we evaluated the performance of queries for which only one query type is processed. Each query set includes 1,000 OQs or TQs.

Fig. 9 shows the performance comparison between previous ordering methods and our method using SLIDProx. In the results graph, the horizontal axis represents the number of TIs inserted in the specified index, and the vertical axis represents the total number of node accesses when the query set is executed for the index. Comparing the results

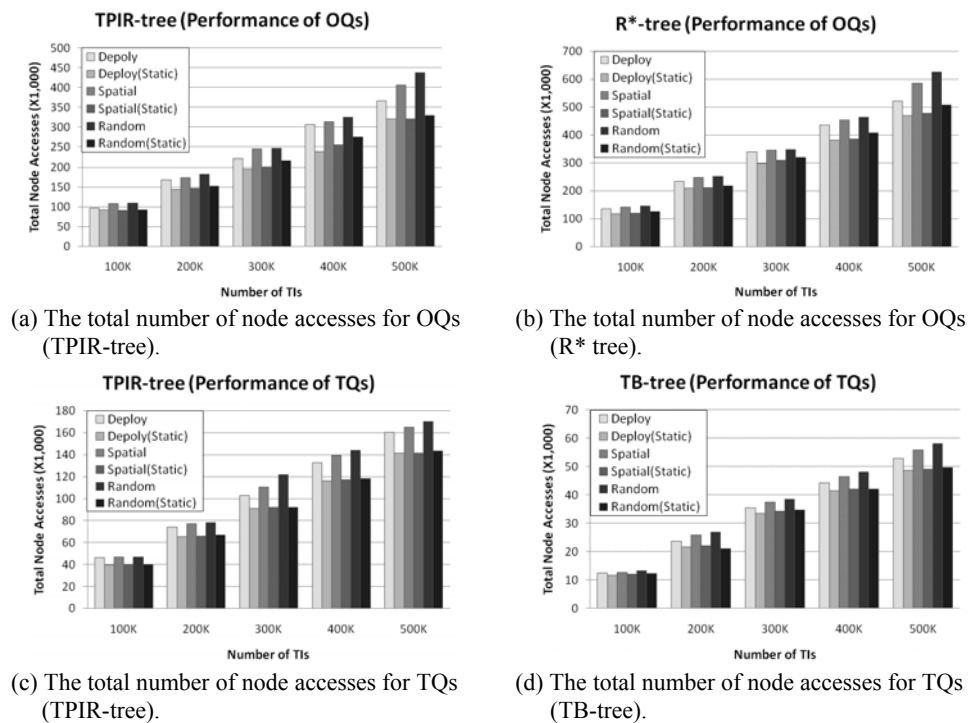


Fig. 9. Performance evaluation using SLIDProx.

for 100K TIs, there is little performance gap between previous methods and our method. However, as more TIs are inserted in the index, the performance gap becomes larger, irrespective of the index type and query type. This implies that SLIDProx correctly reflects the characteristics of queries for tracing tags as tag movements accumulate along FlowPaths.

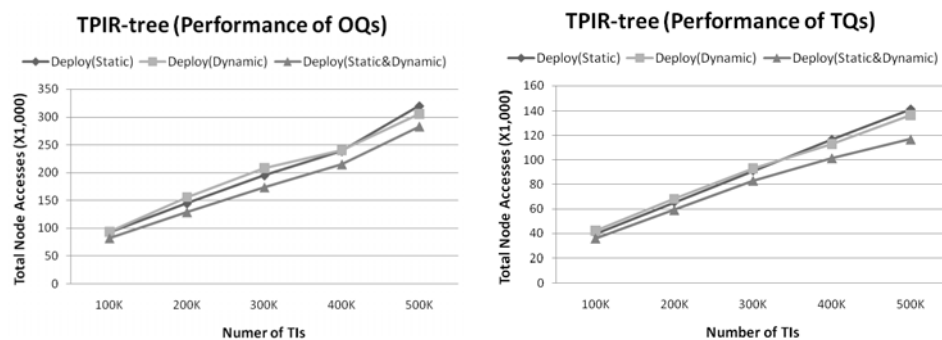
SLIDProx can be easily measured using the FC property because the FC is already fixed at the initial configuration of RFID locations. As shown in the results of Fig. 9, LID lists based on SLIDProx show better performance than the original methods. We discovered that all LID lists based on SLIDProx show a similar performance over all datasets. This means that our heuristic method for solving the LOP found a nearly optimal OLIDList based on SLIDProx, even though it does not find the best solution, as mentioned in section 5. Therefore, our method can be applied to the reordering of previously assigned LIDs to obtain initial domain values in the LID domain of the index.

6.3 Performance of DLIDProx

We also measured the performance of indices that reorder LIDs using DLIDProx. In the results of Fig. 9, Deploy(Static) shows the best performance for all index structures. For this reason, we chose Deploy(Static) for the initial LID list. DLIDProx must consider the number of queries and query results as input variables of the function. To achieve this, we processed queries for tracing tags continuously through the index structure, and estimated query-specific variables over all periods. Finally, Deploy(Dynamic) and Deploy(Static & Dynamic), which are the dynamic versions of Deploy and Deploy(Static), were determined after the TDG finished inserting the datasets of Table 3 in the index structure.

We first carried out an experiment using DLIDProx for only one query type (OQ or TQ). To obtain an optimized order of LIDs per query type, we processed 10,000 OQs or TQs before executing LID reordering. Fig. 10 shows the query performance using DLIDProx on the TPIR-tree. As was the case in the experiment using SLIDProx, each query set included 1,000 OQs or TQs.

We discovered that Deploy(Static&Dynamic) can retrieve results at a lower cost of node access than other methods because it not only reflects FC values but also accumulates dynamic properties for FlowPaths continuously. In the case of Deploy, its performance is poorer than that of Deploy(Static) in Fig. 9, but Deploy(Dynamic)'s performance is similar to that of Deploy(Static) as the number of TIs is increased. When 500K TIs are inserted, the performance of Deploy(Dynamic) is better than that of Deploy(Static). This

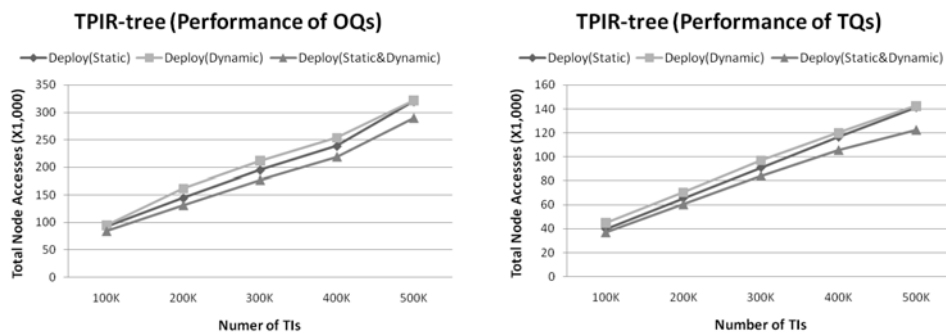


(a) The total number of node accesses for OQs. (b) The total number of node accesses for TQs.
Fig. 10. Performance evaluation using DLIDProx per query type.

proves that DLIDProx can measure the closeness between BizLocs more precisely when tag movements and queries for these tags continuously occur.

Finally, we performed experiments for the case where OQs and TQs are processed simultaneously. The previous experiments only evaluated the performance for an individual type of query (OQ or TQ). Using DLIDProx, the ratios of the number for each query type to the total number of queries influence the value of α , as shown in Eq. (3). Following a change of α , the weight of $LIDProx_OQ_T(i, j)$ and $LIDProx_TQ_T(i, j)$ are also changed in the $DLIDProx_T(i, j)$ of Eq. (2). For this reason, we need to measure the query performance for the case where OQ and TQ are processed simultaneously. To achieve this, we processed 5,000 OQs and 5,000 TQs when inserting each set of 100K TIs.

Fig. 11 shows the search performance for Deploy(Dynamic), Deploy(Static), and Deploy(Static & Dynamic) when 1,000 OQs or TQs are processed. Overall, Deploy(Dynamic) and Deploy(Static & Dynamic) show better performance than Deploy(Static) for both OQ and TQ cases. However, the number of node accesses for dynamic methods is slightly increased, compared with that in Fig. 10. The reason is that evaluating the two subfunctions in Eq. (2) is detrimental to the performance of query processing.



(a) The total number of node accesses for OQs. (b) The total number of node accesses for TQs.
Fig. 11. Performance evaluation for DLIDProx when both query types are processed simultaneously.

7. CONCLUSIONS AND FUTURE WORK

This paper introduced the problem of using LID as the domain value of the index for TIs, and proposed a solution to this problem. To use LID as the domain value, it is necessary to provide a correct proximity measure to ensure logical closeness between TIs. However, although there are many LID numbering methods for RFID locations, they are not concerned with ensuring logical closeness between TIs. The basic idea for solving this problem is to reorder LIDs using a LIDProx function between two LIDs. The LIDProx function determines which LID to place close to a specific LID in the domain. We also proposed a reordering scheme for LIDs, which uses a weighted graph based on LIDProx between LIDs. It is possible to minimize the cost of disk access during a query because logically adjacent TIs, which are frequently accessed simultaneously, can be stored on disk in close proximity. Our experiments showed that the proposed reordering scheme considerably improves the performance of queries for tracing tags, compared with previous schemes for assigning LIDs.

In this paper, we concentrated on providing a solution for reordering LIDs based on LIDProx. Because the proposed LIDProx is computed with time-parameterized properties, it changes over time. Therefore, it is necessary to reorder LIDs, either periodically or from time to time, to reflect the changed LIDProx between LIDs. To process queries efficiently over the entire period, updating the TI index is also required to accord with changing LIDProx. We are currently developing a dynamic reordering method for LIDs and a restructuring method for the index, because this is an important issue in optimizing the RFID tag database.

REFERENCES

1. EPCglobal, "EPC information services (EPCIS) specification," Version 1.0, EPCglobal Inc., 2006.
2. M. Harrison, "EPC information service – data model and queries," White Paper, CAM-AUTOID-WH-025, Auto-ID Center, 2003.
3. EPCglobal, <http://www.epcglobalinc.org>.
4. EPCglobal, "EPC™ tag data standards," Version 1.3, EPCglobal Inc., 2006.
5. K. Römer, T. Schoch, F. Mattern, and T. Dübendorfer, "Smart identification frameworks for ubiquitous computing applications," in *Proceedings of IEEE International Conference on Pervasive Computing and Communications*, 2003, pp. 256-262.
6. D. Lin, H. G. Elmongui, E. Bertino, and B. C. Ooi, "Data management in RFID applications," in *Proceedings of the 18th International Conference on Database and Expert Systems Applications*, Vol. 4653, 2007, pp. 434-444.
7. F. Wang and P. Liu, "Temporal management of RFID data," in *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, pp. 1128-1139.
8. C. H. Ban, B. H. Hong, and D. H. Kim, "Time parameterized interval R-tree for tracing tags in RFID systems," in *Proceedings of the 16th International Conference on Database and Expert Systems Applications*, Vol. 3588, 2005, pp. 503-513.
9. A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, Vol. 31, 1999, pp. 264-323.
10. T. Brinkhoff and H. P. Kriegel, "The impact of global clustering on spatial database systems," in *Proceedings of International Conference on Very Large Data Bases*, 1994, pp. 168-179.
11. I. Kamel and C. Faloutsos, "Hilbert R-tree: An improved R-tree using fractals," in *Proceedings of International Conference on Very Large Data Bases*, 1994, pp. 500-509.
12. D. S. Cho and B. H. Hong, "Optimal page ordering for region queries in static spatial databases," in *Proceedings of International Conference on Database and Expert Systems Applications*, Vol. 1873, 2000, pp. 366-375.
13. H. V. Jagadish, "Linear clustering of objects with multiple attributes," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Vol. 19, 1990, pp. 332-342.
14. Y. Theodoridis, M. Vazirgiannis, and T. Sellis, "Spatio-temporal indexing for large multimedia applications," in *Proceedings of International Conference on Multimedia Computing and Systems*, 1996, pp. 441-448.
15. D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of

- moving object trajectories,” in *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000, pp. 395-406.
16. T. Yufei and D. Papadias, “MV3R-Tree: A spatio-temporal access method for time-stamp and interval queries,” in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 431-440.
 17. H. Zhu, J. Su, and O. H. Ibarra, “Trajectory queries and octagons in moving object databases,” in *Proceedings of the 11th International Conference on Information and Knowledge Management*, 2002, pp. 413-421.
 18. A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47-54.
 19. N. Beckmann and H. P. Kriegel, “The R*-tree: An efficient and robust access method for points and rectangles,” in *Proceedings ACM SIGMOD International Conference on Management of Data*, 1990, pp. 322-331.
 20. M. A. Nascimento and J. R. O. Silva, “Towards historical R-trees,” in *Proceedings of the ACM Symposium on Applied Computing*, 1998, pp. 235-240.
 21. D. Whitley, “A genetic algorithm tutorial,” *Statistics and Computing*, Vol. 4, 1994, pp. 65-85.
 22. D. Whitley and J. Dzebera, “Advance correlation analysis of operators for the traveling salesman problems,” *Parallel Problem Solving from Nature*, Vol. 866, 1994, pp. 68-77.
 23. S. S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, New York, Berlin, Heidelberg, 1998.
 24. M. F. Worboys, *GIS: A Computing Perspective*, Taylor & Francis, London, 1995.



Sungwoo Ahn (安星佑) received the M.S. and Ph.D. degrees in Computer Engineering from Pusan National University, Busan, Korea, in 2001 and 2009. He is currently a Post Doctoral Fellow at Department of Computer Engineering in Pusan National University, Busan, Korea. He has developed RFID middleware with Institute of Logistics Information Technology. His research interests include RFID system, RFID middleware, moving object database, RTLS middleware and sensor network system.



Bonghee Hong (洪鳳憲) received the M.S. and Ph.D. degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1984 and 1988. In 1987, he joined the faculty of Computer Engineering of the Pusan National University (PNU). He is working as a Professor of Database in the Department of Computer Engineering at the PNU. He is a Director of Institute of Logistics Information Technology. His research interests include theory of database systems, moving object databases, spatial databases, RFID system and RFID middleware.