Warehousing and Mining Massive RFID Data Sets

Jiawei Han, Hector Gonzalez, Xiaolei Li, and Diego Klabjan

University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{hanj, hagonzal, xli10, klabjan}@uiuc.edu

Abstract. Radio Frequency Identification (RFID) applications are set to play an essential role in object tracking and supply chain management systems. In the near future, it is expected that every major retailer will use RFID systems to track the movement of products from suppliers to warehouses, store backrooms and eventually to points of sale. The volume of information generated by such systems can be enormous as each individual item (a pallet, a case, or an SKU) will leave a trail of data as it moves through different locations. We propose two data models for the management of this data. The first is a *path cube* that preserves object transition information while allowing muti-dimensional analysis of path dependent aggregates. The second is a *workflow cube* that summarizes the major patterns and significant exceptions in the flow of items through the system. The design of our models is based on the following observations: (1) items usually move together in large groups through early stages in the system (*e.g.*, distribution centers) and only in later stages (*e.g.*, stores) do they move in smaller groups, (2) although RFID data is registered at the primitive level, data analysis usually takes place at a higher abstraction level, (3) many items have similar flow patterns and only a relatively small number of them truly deviate from the general trend, and (4) only non-redundant flow deviations with respect to previously recorded deviations are interesting. These observations facilitate the construction of highly compressed RFID data warehouses and the exploration of such data warehouses by scalable data mining. In this study we give a general overview of the principles driving the design of our framework. We believe warehousing and mining RFID data presents an interesting application for advanced data mining.

1 Introduction

Radio Frequency Identification (RFID) is a technology that allows a sensor (RFID reader) to read, from a distance and without line of sight, a unique identifier that is provided (via a radio signal) by an "inexpensive" tag attached to an item. RFID offers a possible alternative to bar code identification systems and it facilitates applications like item tracking and inventory management in the supply chain. The technology holds the promise to streamline supply chain management, facilitate routing and distribution of products, and reduce costs by improving efficiency.

Large retailers like Walmart, Target, and Albertsons have already begun implementing RFID systems in their warehouses and distribution centers, and are requiring their suppliers to tag products at the pallet and case levels. Individual tag prices are expected to fall from around 25 cents per unit to 5 cents per unit by 2007. At that price level, we

X. Li, O.R. Zaiane, and Z. Li (Eds.): ADMA 2006, LNAI 4093, pp. 1–18, 2006.

can expect tags to be placed at the individual item level for many products. The main challenge then becomes how can companies handle and interpret the enormous volume of data that an RFID application will generate. Venture Development Corporation [11], a research firm, predicts that when tags are used at the item level, Walmart will generate around 7 terabytes of data every day. Database vendors like Oracle, IBM, Teradata, and some startups are starting to provide solutions to integrate RFID information into enterprise data warehouses.

Example. Suppose a retailer with 3,000 stores sells 10,000 items a day per store. Assume that we record each item movement with a tuple of the form: $(EPC, location, time)$, where EPC is an Electronic Product Code which uniquely identifies each item[1]. If each item leaves only 10 traces before leaving the store by going through different locations, this application will generate at least 300 million tuples per day. A manager may ask queries on the duration of paths like (Q_1): "*List the average shelf life of dairy products in 2003 by manufacturer*", or on the structure of the paths like (Q_2): "*What is the average time that it took coffee-makers to move from the warehouse to the shelf and finally to the checkout counter in January of 2004?*", or on the major flow characteristics like (Q_3): "*Is there a correlation between the time spent at quality control and the probability of returns for laptops manufactured in Asia?*".

Such enormous amount of low-level data and flexible high-level queries pose great challenges to traditional relational and data warehouse technologies since the processing may involve retrieval and reasoning over a large number of inter-related tuples through different stages of object movements. No matter how the objects are sorted and clustered, it is difficult to support various kinds of high-level queries in a uniform and efficient way. A nontrivial number of queries may even require a full scan of the entire RFID database.

1.1 Path Cube

The *path cube* compresses and aggregates the paths traversed by items in the system along time, location, and product related dimensions. This cube will allow a wide range of OLAP queries to be answered efficiently. Our design is based on the following key observations.

- We need to eliminate the redundancy present in RFID data. Each reader provides tuples of the form $(EPC, location, time)$ at fixed time intervals. When an item stays at the same location, for a period of time, multiple tuples will be generated. We can group these tuples into a single one of the form $(EPC, location, time\_in, time\_out)$.
- Items tend to move and stay together through different locations. For example, a pallet with 500 cases of CDs may arrive at the warehouse; from there cases of 50 CDs may move to the shelf; and from there packs of 5 CDs may move to the checkout counter. We can register a single *stay* tuple of the form $(EPC\ list, location, time\_in, time\_out)$ for the CDs that arrive in the same pallet and stay together in the warehouse, and thus generate an 80% space saving.

[1] We will use the terms EPC and RFID tag interchangeably throughout the paper.

- We can gain further compression by reducing the size of the EPC lists in the *stay* records by grouping items that move to the same locations. For example, if we have a *stay* record for the 50 CDs that stayed together at the warehouse, and that the CDs moved in two groups to shelf and truck locations. We can replace the list of 50 EPCs in the stay record for just two *generalized identifiers* (*gids*) which in turn point to the concrete EPCs. In addition to the compression benefits, we can gain query processing speedup by assigning path-dependent names to the gids.
- Most search or mining queries are likely to be at a high level of abstraction, and will only be interested in the low-level individual items if they are associated with some interesting patterns discovered at a high level.

1.2 Workflow Cube

The *workflow cube* aggregates item flows at different levels of abstraction of the item-related dimensions and the location-related dimensions. The measure of the *workflow cube* is a compressed probabilistic workflow, *i.e.*, each cell in the cube will contain a workflow computed on the paths aggregated in the cell. It differs from the *path cube* in two major ways: (1) it does not consider absolute time, and only relative durations, and (2) the measure of each cell is a workflow and not a scalar aggregate such as sum or count, which would be typical in the *path cube*.

Commodity flows can be analyzed from the perspective of paths (*path view*) and the abstraction level at which path stages appear or from the perspective of item related dimensions (*item view*) and their abstraction levels. Figure 1 presents a path (seen in the middle of the figure) aggregated to two different abstraction levels. The path at the top of the figure shows the individual locations inside a store, while it collapses locations that belong to transportation; this may be interesting to a store manager. The path at the bottom of the figure on the other hand, collapses locations that belong to stores, and keeps individual locations that belong to transportation; this view may be interesting to a transportation manager in the company. An orthogonal view into RFID commodity flows is related to items themselves. This is a view much closer to traditional data cubes. An item can have a set of dimension describing its characteristics, *e.g.*, product, brand, manufacturer. Each of these dimensions has an associated concept hierarchy.

The key challenge in constructing a *workflow cube* based on a set of RFID paths is to devise an efficient method to compute summaries of commodity flows for those item views and path views that are interesting to the different data analysts utilizing the application. The proposed construction method is based on the following optimizations:

- Reduction of the size of the workflow cube by exploring two strategies. The first is to compute only those cells that contain a minimum number of paths (iceberg condition). This makes sense as each workflow is a probabilistic model that can be used to conduct statistically significant analysis only if there is enough data to support it. The second strategy is to compute only workflows that are non-redundant given higher abstraction level workflows. For example, if the flow patterns of 2% milk are similar to those of milk (under certain threshold), then by registering just the high level workflow we can infer that one for 2% milk , *i.e.*, we expect any low level concept to behave in a similar way to its parents, and only when this behavior is truly different, we register such information.

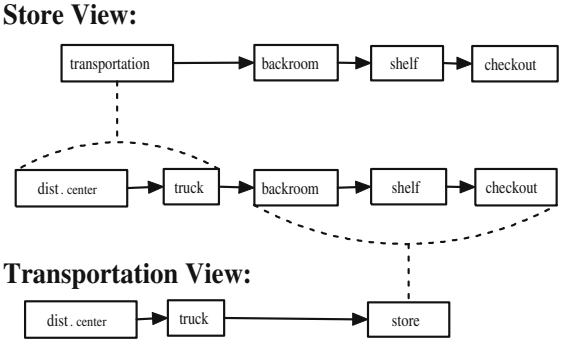

Fig. 1. Path views: The same path can be seen at two different abstraction levels

- Shared computation. We explore efficient computation of the *workflow cube* by sharing the computation of frequent cells and frequent path segments simultaneously. Similar to shared computation of multiple cuboids in BUC-like computation [3], we propose to compute frequent cells in the *workflow cube* and frequent path segments aggregated at every interesting abstraction level simultaneously. Shared computation minimizes the number of scans of the path database by maximizing the amount of information collected during each scan.
- Pruning of the search space using both the path and item views. To speed up cube computation, we use pre-counting of high abstraction level itemsets that will help us prune a large portion of the candidate space without having to collect their counts. For example if we detect that the stage shelf is not frequent in general, we know that for no particular duration it can be frequent; or if a store location is not frequent, no individual location within the store can be frequent.

The rest of the paper is organized as follows. Section 2 presents the structure of the input RFID data. Section 3 presents the architecture of the *path cube*. Section 4 Gives an overview of the architecture of the *workflow cube*. Section 5 reports experimental results. We discuss the related work in Section 6, outline some promising research directions on mining RFID data in Section 7, and conclude our study in Section 8.

2 RFID Data

Data generated from an RFID application can be seen as a stream of RFID tuples of the form $(EPC, location, time)$, where *EPC* is the unique identifier read by an RFID reader, *location* is the place where the RFID reader scanned the item, and *time* is the time when the reading took place. Tuples are usually stored according to a time sequence. A single EPC may have multiple readings at the same location, each reading is generated by the RFID reader scanning for tags at fixed time intervals or on a continuous basis.

In order to reduce the large amount of redundancy in the raw data, data cleaning should be performed. The output after data cleaning is a set of clean stay records of the form $(EPC, location, time\_in, time\_out)$ where *time_in* is the time when the object enters the location, and *time_out* is the time when the object leaves the location.

Data cleaning of stay records can be accomplished by sorting the raw data on EPC and time, and generating *time_in* and *time_out* for each location by merging consecutive records for the same object staying at the same location.

Furthermore, for the purpose of constructing the *workflow cube* we are not interested in absolute time but only relative durations and thus we can rewrite the cleansed RFID database as a set of paths of the form $(EPC : (l_1, d_1)\ (l_2, d_2) \ldots (l_k, d_k))$, were l_i is the i-th location in the path traveled by the item identified by EPC, and d_i is the total time that the item stayed at l_i.

The duration that an item spent at a location is a continuous attribute. In order to simplify the model further we can discretize all the distinct durations for each location into a fixed number of clusters. We call such a path database with discretized durations *clustered path-database*.

3 Architecture of the Path Cube

Before we describe our proposed architecture for the *path cube*, it is important to describe why a traditional data cube model would fail on such data. Suppose we view the cleansed RFID data as the fact table with dimensions $(EPC, location, time\_in, time\_out : measure)$. The data cube will compute all possible group-bys on this fact table by aggregating records that share the same values (or any *) at all possible combinations of dimensions. If we use count as measure, we can get for example the number of items that stayed at a given location for a given month. The problem with this form of aggregation is that it does not consider links between the records. For example, if we want to get the number of items of type "dairy product" that traveled from the distribution center in Chicago to stores in Urbana, we cannot get this information. We have the count of "dairy products" for each location but we do not know how many of those items went from the first location to the second. We need a more powerful model capable of aggregating data while preserving its path-like structure.

We propose an RFID warehouse architecture that contains a fact table, *stay*, composed of cleansed RFID records; an information table, *info*, that stores path-independent information for each item, *i.e.*, SKU information that is constant regardless of the location of the item, such as manufacturer, lot number, and color; and a *map* table that links together different records in the fact table that form a path. We call the *stay*, *info*, and *map* tables aggregated at a given abstraction level an *RFID-Cuboid.*

The main difference between the RFID warehouse and a traditional warehouse is the presence of the map table linking records from the fact table (*stay*) in order to preserve the original structure of the data.

The computation of cuboids in the *path cube* (*RFID-Cuboids*) is more complex than that of regular cuboids as we will need to aggregate the data while preserving the structure of the paths at different abstraction levels.

From the data storage and query processing point of view the RFID warehouse can be viewed as a multi-level database. The raw RFID repository resides at the lowest level, on its top are the cleansed RFID database, the minimum abstraction level *RFID-Cuboids* and a sparse subset of the full cuboid lattice composed of frequently queried (popular) *RFID-Cuboids.*

3.1 Key Ideas of RFID Data Compression

Even with the removal of data redundancy from RFID raw data, the cleansed RFID database is usually still enormous. Here we explore several general ideas for constructing a highly compact RFID data warehouse.

Taking advantage of bulky object movements
Since a large number of items travel and stay together through several stages, it is important to represent such a collective movement by a single record no matter how many items were originally collected. As an example, if 1,000 boxes of milk stayed in location loc_A between time t_1 (time_in) and t_2 (time_out), it would be advantageous if only one record is registered in the database rather than 1,000 individual RFID records. The record would have the form: $(gid, prod, loc_A, t_1, t_2, 1000)$, where 1,000 is the count, $prod$ is the product id, and gid is a generalized id which will not point to the 1,000 original EPCs but instead point to the set of new gids which the current set of objects move to. For example, if this current set of objects were split into 10 partitions, each moving to one distinct location, gid will point to 10 distinct new gids, each representing a record. The process iterates until the end of the object movement where the concrete EPCs will be registered. By doing so, no information is lost but the number of records to store such information is substantially reduced.

Taking advantage of data generalization
Since many users are only interested in data at a relatively high abstraction level, data compression can be explored to group, merge, and compress data records. For example, if the minimal granularity of time is hour, then objects moving within the same hour can be seen as moving together and be merged into one movement. Similarly, if the granularity of the location is shelf, objects moving to the different layers of a shelf can be seen as moving to the same *shelf* and be merged into one. Similar generalization can be performed for products (*e.g.*, merging different sized milk packages) and other data as well.

Taking advantage of the merge and/or collapse of path segments
In many analysis tasks, certain path segments can be ignored or merged for simplicity of analysis. For example, some non-essential object movements (*e.g.*, from one shelf to another in a store) can be completely ignored in certain data analysis. Some path segments can be merged without affecting the analysis results. For store managers, merging all the movements before the object reaches the store could be desirable. Such merging and collapsing of path segments may substantially reduce the total size of the data and speed-up the analysis process.

3.2 RFID-CUBOID

With the data compression principles in mind, we propose *RFID-Cuboid*, a data structure for storing aggregated data in the RFID warehouse. Our design ensures that the data are disk-resident, summarizing the contents of a cleansed RFID database in a compact yet complete manner while allowing efficient execution of both OLAP and tag-specific queries.

The *RFID-Cuboid* consists of three tables: (1) *Info*, which stores product information for each RFID tag, (2) *Stay*, which stores information on items that stay together at

a location, and (3) *Map*, which stores path information necessary to link multiple stay records.

Information Table
The information table stores path-independent dimensions, such as product name, manufacturer, product price and product category. Each dimension can have an associated concept hierarchy. All traditional OLAP operations can be performed on these dimensions in conjunction with various RFID-specific analysis. For example, one could drill-down on the product category dimension from "clothing" to "shirts" and retrieve shipment information only on shirts.

Entries in *Info* are records of the form: $\langle (EPC\_list), (d_1, \ldots, d_m) : (m_1, \ldots, m_i) \rangle$, where the code list contains a set of items that share the same values for dimensions $d_1, \ldots, d_m$, and $m_1, \ldots, m_i$ are measures of the given items, *e.g.*, price.

Stay Table
As mentioned in the introduction, items tend to move and stay together through different locations. Compressing multiple items that stay together at a single location is vital in order to reduce the enormous size of the cleansed RFID database. In real applications items tend to move in large groups. At a distribution center there may be tens of pallets staying together, and then they are broken into individual pallets at the warehouse level. Even if products finally move at the individual item level from a shelf to the checkout counter, our stay compression will save space for all previous steps taken by the item.

Each entry in *Stay* is a record of the form: $\langle (gids, location, time\_in, time\_out) : (m_1, \ldots, m_k) \rangle$, where $gids$ is a set of generalized record ids each pointing to a list of RFID tags or lower level $gids$, $location$ is the location where the items stayed together, $time\_in$ is the time when the items entered the location, and $time\_out$ the time when they left. If the items did not leave the location, $time\_out$ is $NULL$. Finally, $m_1, \ldots, m_n$ are the measures recorded for the stay, *e.g.*, count, average time at $location$, and the maximal time at $location$.

Map Table
The *Map* table is an efficient structure that allows query processing to link together stages that belong to the same path in order to perform structure-aware analysis, which could not be answered by a traditional data warehouse. There are two main reasons for using a *Map* table instead of recording the complete EPC lists at each stage: (1) data compression, by reduction of the size of the EPC lists size at each location; and (2) query processing efficiency, by encoding the path of a group of items in their generalized identifier.

The map table contains mappings from higher level $gids$ to lower level ones or EPCs. Each entry in *Map* is a record of the form: $\langle gid, (gid_1, \ldots, gid_n) \rangle$, meaning that, gid is composed of all the EPCs pointed to by $gid_1, \ldots, gid_n$. The lowest level $gids$ will point directly to individual items.

In order to facilitate query processing we will assign path-dependent labels to high level $gids$. The label will contain one identifier per location traveled by the items in the gid.

3.3 Path Cuboid Lattice

In order to provide fast response to queries specified at various levels of abstraction, it is important to pre-compute some *RFID-Cuboids* at different levels of the concept hierarchies for the dimensions of the *Info* and *Stay* tables. It is obviously too expensive to compute all the possible generalizations, and partial materialization is a preferred choice. This problem is analogous to determining which set of cuboids in a data cube to materialize in order to answer OLAP queries efficiently given the limitations on storage space and precomputation time. This issue has been studied extensively in the data cube research [5,10] and the principles are generally applicable to the selective materialization of *RFID-Cuboids.*

In our design, we suggest to compute a set of *RFID-Cuboids* at the minimal interesting level at which users will be interested in inquiring the database, and a small set of higher level structures that are frequently requested and that can be used to quickly compute non-materialized *RFID-Cuboids.*

An *RFID-Cuboid* residing at the minimal interesting level will be computed directly from the cleansed RFID database and will be the lowest cuboid that can be queried unless one has to dig directly into the detail cleansed data in some very special cases.

3.4 Query Processing

In this section we discuss the implementation of the basic OLAP operations, *i.e.*, drill-down, roll-up, slice, and dice, applied to the *path cube*, and introduce a new operation, *path selection*, relevant to the paths traveled by items.

Given the very large size and high dimensionality of the RFID warehouse we can only materialize a small fraction of the total number of *RFID-Cuboids.* We will compute the *RFID-Cuboid* that resides at the minimum abstraction layer that is interesting to users, and those *RFID-Cuboids* that are frequently requested. When a roll-up or drill-down operation requires an *RFID-Cuboid* that has not yet been materialized, it would have to be computed on the fly from an existing *RFID-Cuboid* that is close to the required one but at a lower abstraction level. The slice and dice operations can be implemented efficiently by using relational query execution and optimization techniques.

Path Selection

Path queries, which ask about information related to the structure of object traversal paths, are unique to the RFID warehouse since the concept of object movements is not modeled in traditional data warehouses. It is essential to allow users to inquire about an aggregate measure computed based on a predefined sequence of locations (path). One such example could be: "*What is the average time for milk to go from farms to stores in Illinois?*".

Queries on the paths traveled by items are fundamental to many RFID applications and will be the building block on top of which more complex data mining operators can be implemented. We will illustrate this point with a real example. The United States government is currently in the process of requiring the containers arriving into the country, by ship, to carry an RFID tag. The information can be used to determine if the path traveled by a given container has deviated from its historic path. This application may need to first execute a path-selection query across different time periods, and then use outlier detection and clustering to analyze the relevant paths.

In order to answer a path query we first select the $gids$ for the stay records that match the conditions for the initial and final stages of the query expression. For example, g_{start} may look like $\langle 1.2, 8.3.1, 3.4\rangle$ and g_{end} may look like $\langle 1.2.4.3, 4.3, 3.4.3\rangle$. We then compute the pairs of gids from g_{start} that are a prefix of a gid in g_{end}. We get the pairs $\langle (1.2, 1.2.4.3), (3.4, 3.4.3)\rangle$. For each pair we then retrieve all the stay records. The pair $(1.2, 1.2.4.3)$ would require us to retrieve stay records that include gids 1.2, 1.2.4, and 1.2.4.3. Finally, we verify that each of these records matches the selection conditions for each $stage_i$ and for *Info*, and add those paths to the answer set.

4 Architecture of the Workflow Cube

The *workflow cube*, as in standard OLAP, will be composed of cuboids that aggregate item flows at a given abstraction level. The *workflow cube* differs from the traditional data cube in two major ways. First, the measure of each cell will not be a simple aggregate but a commodity *workflow* that captures the major movement trends and significant deviations for the subset of objects in the cell. Second, each *workflow* itself can be viewed at multiple levels by changing the level of abstraction path stages. The *workflow cube* also differs from the *path cube* in that it only considers relative duration and not absolute time in its analysis of paths. This distinction is useful in building a statistical model of object flows.

4.1 Probabilistic Workflow

A duration independent workflow is a tree where each node represents a location and edges correspond to transitions between locations. All common path prefixes appear in the same branch of the tree. Each transition has an associated probability, which is the percentage of items that took the transition represented by the edge. For every node we also record a termination probability, which is the percentage of paths that terminate at the location associated with the node.

We have several options to incorporate duration information into a duration independent workflow, the most direct way is to create nodes for every combination of location and duration. This option has the disadvantage of generating very large workflows. A second option is to annotate each node in the duration independent workflow with a distribution of possible durations at the node. This approach keeps the size of the workflow manageable and captures duration information for the case when (i) the duration distribution between locations is independent, *e.g.*, the time that milk spends at the shelf is independent of the time it spent in the store backroom; and (ii) transitions probabilities are independent of duration, *e.g.*, the probability of a box of milk which transits from the shelf to the checkout counter does not depend on the time it spent at the backroom.

There are cases when conditions (i) and (ii) do not hold, *e.g.*, a product that spends a long time at a quality control station may increase its probability of moving to the return counter location at a retail store. In order to cover these cases we propose to use a new model that not only records duration and transition distributions at each node, but also stores information on significant deviations in duration and transition probabilities given frequent path prefixes to the node. A prefix to a node is a sequence of $(location, duration)$ pairs that appear in the same branch as the node but before

it. The construction of such workflow requires two parameters, ϵ that is the minimum deviation of a duration or transition probability required to record an exception, and δ the minimum support required to record a deviation. The purpose of ϵ is to record only deviations that are truly interesting in that they significantly affect the probability distribution induced by the workflow; and the purpose of δ to prevent the exceptions in the workflow to be dominated by statistical noise in the path database.

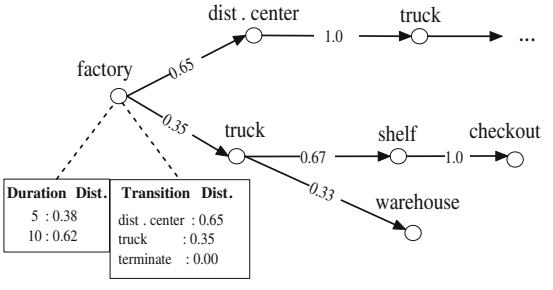

Fig. 2. Workflow

The workflow in Figure 2 also registers significant exceptions to duration and transition probabilities (not shown in the figure), *e.g.*, the transition probability from the truck to the warehouse, coming from the factory, is in general 33%, but that probability is 50% when we stay for just 1 hour at the truck location. Similarly we can register exceptions for the distribution of durations at a location given previous durations, *e.g.*, items in the distribution center spend 1 hour with probability 20% and 2 hours with probability 80%, but if an item spent 5 hours at the factory, the distribution changes and the probability of staying for 2 hours in the distribution center becomes 100%.

Computing a workflow can be done efficiently by (1) constructing a prefix tree for the path database, (2) annotating each node with duration and transition probabilities, and (3) mining the path database for frequent paths with minimum support δ, and checking if those paths create exceptions that deviate by more than ϵ from the general probability. Steps (1) and (2) can be done with a single scan of the path database, and for step (3) we can use any existing frequent pattern mining algorithm.

4.2 Workflow Cuboid Lattice

Each cuboid in the *workflow cube* will reside at a certain level of abstraction in the *Item Lattice* and *Path Lattice*. The *Item Lattice* represents all the possible levels of abstraction of the dimensions used to describe items, *e.g.*, product, manufacturer, and color. This lattice is the same that one would encounter on traditional data cubes. The *Path Lattice* represents all possible levels of abstraction of the locations and durations of the paths in the database, and it is helpful in expanding only those portions of the paths that are interesting to data analysts while collapsing the rest of the locations.

Aggregation along the path abstraction lattice is unique to *workflow cubes* and is quite different from the type of aggregation performed in a regular data cube. In a data cube, an aggregated cell contains a measure on the subset of tuples from the fact

table that share the same values on every aggregated dimension. When we do path aggregation, the dimensions from the fact table remain unchanged, but it is the measure of the cell itself that changes. This distinct property requires us to develop new methods to construct a *workflow cube* that has aggregation on both item and path dimensions.

4.3 Workflow Redundancy

The workflow registered for a given cell in a *workflow cube* may not provide new information on the characteristics of the data in the cell, if the cells at a higher abstraction level on the item lattice, and the same abstraction level on the path lattice, can be used to derive the workflow in the cell. For example, if we have a workflow G_1 for milk, and a workflow G_2 from milk 2% (milk is an ancestor of milk 2% in the item abstraction lattice), and $G_1 = G_2$, we view that G_2 is redundant, as it can be inferred from G_1. Registering only non-redundant cells not only allows significant compression but also provides important insight into the relationship of flow patterns from high to low levels of abstraction, and can facilitate the discovery of exceptions in multi-dimensional space.

4.4 Iceberg Workflow Cube

A workflow is a statistical model that describes the flow behavior of objects given a collection of paths. If the data set on which the workflow is computed is very small, the workflow may not be useful in conducting data analysis. Each probability in the model will be supported by such a small number of observations and it may not be an accurate estimate of the true probability. In order to minimize this problem, we will materialize only cells in the *workflow cube* that contain at least δ paths.

Iceberg cubes can be computed efficiently by using apriori pruning of infrequent cells. We can materialize the cube from low dimensionality to high dimensionality. If at some point a low level cell is not frequent, we do not need to check the frequency of any specialization of the cell.

5 Performance Study

In this section, we perform an experimental evaluation of the compression power and query processing performance of the *path cube*; we also report on the speedup gained by using the proposed techniques to compute the *workflow cube*.

5.1 Experimental Setup

The path databases used for our experiments were generated using a synthetic path generator that simulates the movement of items through the supply chain of a large retailer. The simulator varies the level of path bulkiness ($\mathcal{B}$) by changing the number of items that move together at each stage in the supply chain. In general we assume that items move in larger groups near the start of the supply chain (*e.g.* factories) and smaller groups near the end (*e.g.* shelves and checkout counters). Each path dependent dimension (*i.e.*locations and time), and path independent dimension (*e.g.*product, manufacturer, price) has an associated concept hierarchy; we vary the number of distinct

values and skew at each level of the concept hierarchies to change the distribution of frequent cells in the *workflow cube*.

For the *path cube* experiments we compare three distinct methods to represent a cuboid: (1) *clean*, which uses the cleansed path database, (2) *nomap*, which uses the stay and information tables as described in the paper but instead of using gids it directly stores EPC lists, and (3) *map* which uses the stay, information, and map tables. For the *workflow cube* experiments we compare three competing techniques used to compute the frequent cells and frequent path segments necessary to construct the *workflow cube*: (1) *shared* which is an algorithm proposed in the paper and that implements simultaneous mining of frequent cells and frequent path segments at all abstraction levels while performing apriori pruning, (2) *cubing* which is a modified version of of BUC [3] to compute the iceberg cube on the path independent dimensions and then called Apriori [2] to mine frequent path segments in each cell, and (3) *basic* is the same algorithm as *shared* except that we do not perform any candidate pruning.

As a notational convenience, we use the following symbols to denote certain data set parameters. For the *path cube* we use: $\mathcal{B} = (s_1, \ldots, s_k)$ for path bulkiness, $\mathcal{P}$ for the number of products, and k for the average path length. For the *workflow cube* we use: $\mathcal{N}$ for the number of paths, δ for minimum support (iceberg condition), and d for the number of path independent dimensions.

5.2 Path Cube Compression

The *RFID-Cuboids* form the basis for future query processing and analysis. As mentioned previously, the advantage of these data structures is that they aggregate and collapse many records in the cleansed RFID database. Here, we examine the effects of this compression on different data sets.

Figure 3 shows the size of the cleansed RFID database (clean) compared with the map and nomap *RFID-Cuboids*. The data sets contains 1,000 distinct products, traveling in groups of 500, 150, 40, 8, and 1 through 5 path stages, and 500 thousand to 10 million cleansed RFID records. The *RFID-Cuboid* is computed at the same level of abstraction of the cleansed RFID data, and thus the compression is lossless. As it can be seen from Figure 3 the *RFID-Cuboid* that uses *map* has a compression power of around 80% while the one that uses EPC lists has a compression power of around 65%.

5.3 Path Cube Query Processing

A major contribution of the *path cube* is the ability to efficiently answer many types of queries at various levels of aggregation. We assume that for each of the scenarios we have a B+Tree on each of the dimensions. In the case of the map cuboid the index points to a list of gids matching the index entry. In the case of the nomap cuboid and the cleansed database the index points to the tuple $(RFID\ tag, record\ id)$. This is necessary as each RFID tag can be present in multiple records. The query answering strategy used for the map cuboid is the one presented in Section 3.4. The strategy for the other two cases is to retrieve the $(RFID\ tag, record\ id)$ pairs matching each component of the query, intersecting them, and finally retrieving the relevant records.

Figure 4 shows the effect of different cleansed database sizes on query processing. The map cuboid outperforms the cleansed database by several orders of magnitude, and

most importantly the query answer time is independent of database size. The nomap cuboid is significantly faster than the cleansed data but it suffers from having to retrieve very long RFID lists for each stage. The map cuboid benefits from using very short gid lists, and using the path-dependent gid naming scheme that facilitates determining if two stay records form a path without retrieving all intermediate stages.

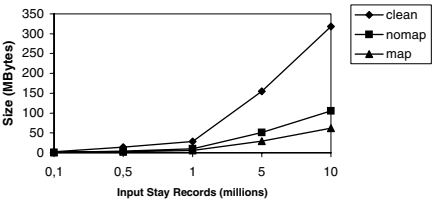

Fig. 3. Compression vs. Cleansed Data Size. $\mathcal{P} = 1000$, $\mathcal{B} = (500, 150, 40, 8, 1)$, $k = 5$.

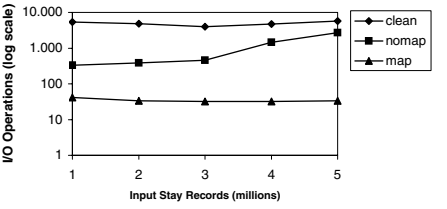

Fig. 4. Query Processing. $\mathcal{P} = 1000$, $\mathcal{B} = (500, 150, 40, 8, 1)$, $k = 5$.

5.4 Workflow Cube Construction Time vs. Database Size

In this experiment we vary the size of path database, from 100,000 paths to 1,000,000 paths. In Figure 5 we can see that the performance of shared and cubing is quite close for smaller data sets but as we increase the number of paths the runtime of shared increases with a smaller slope than that of cubing. This may be due to the fact that as we increase the number of paths the data set becomes denser BUC slows down. Another influencing factor in the difference in slopes is that as the data sets become denser cubing needs to invoke the frequent pattern mining algorithm for many more cells, each with a larger number of paths. We were able to run the basic algorithm for 100,000 and 200,000 paths, for other values the number of candidates was so large that they could not fit into memory. This highlights the importance of the candidate pruning optimizations.

5.5 Workflow Cube Construction Time vs. Minimum Support

In this experiment we constructed a path database with 100,000 paths and 5 path independent dimensions. We varied the minimum support from 0.3% to 2.0%. In Figure 6 we can see that shared outperforms cubing and basic. As we increase minimum support the performance of all the algorithms improves as expected. Basic improves faster that the other two, this is due to the fact that fewer candidates are generated at higher support levels, and thus optimizations based on candidate pruning become less critical. For every support level we can see that shared outperforms cubing, but what is more important we see that shared improves its performance faster than cubing. The reason is that as we increase support shared will quickly prune large portions of the path space, while cubing will repeatedly check this portions for every cell it finds to be frequent.

5.6 Workflow Cube Construction Time vs. Number of Dimensions

In this experiment we kept the number of paths constant at 100,000 and the support at 1%, and varied the number of dimensions from 2 to 10. The datasets used for this

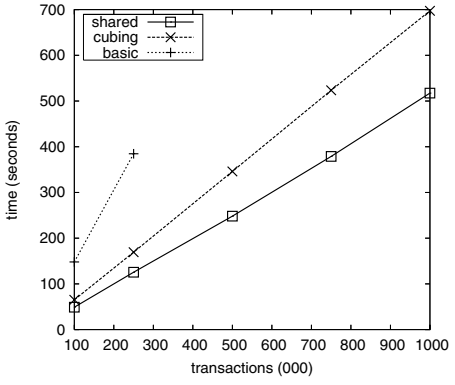

Fig. 5. Time vs. Database Size ($\delta = 0.01$, $d = 5$)

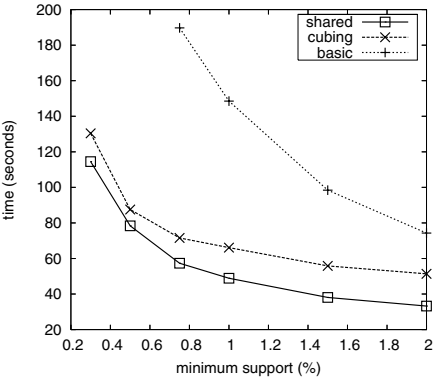

Fig. 6. Time vs. Minimum Support ($\mathcal{N} = 100,000$, $d = 5$)

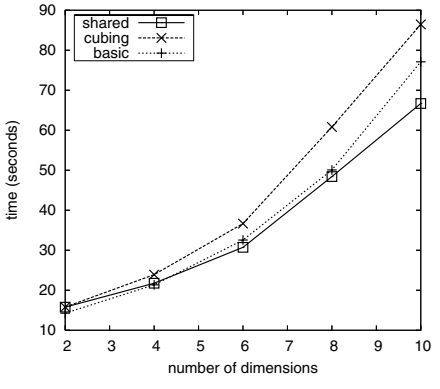

Fig. 7. Time vs. Number of Dimensions ($\mathcal{N} = 100,000$, $\delta = 0.01$)

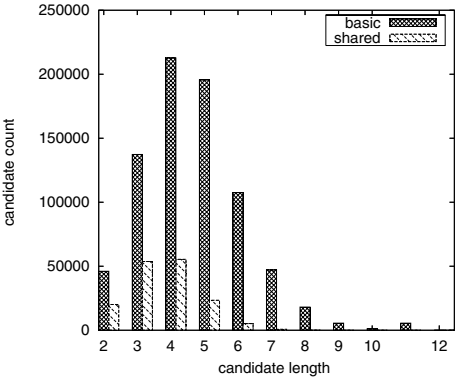

Fig. 8. Pruning Power ($\mathcal{N} = 100,000$, $\delta = 0.01$, $d = 5$)

experiment were quite sparse to prevent the number of frequent cells to explode at higher dimension cuboids. The sparse nature of the datasets makes all the algorithms achieve a similar performance level. We can see in Figure 7 that both shared and cubing are able to prune large portions of the cube space very soon, and thus performance was comparable. Similarly basic was quite efficient as the number of candidates was small and optimizations based on candidate pruning did not make a big difference given that the number of candidates was small to begin with.

5.7 Workflow Cube Construction Pruning Power

This is experiment shows the effectiveness of the proposed optimizations to prune unpromising candidates from consideration in the mining process. We compare the number of candidates that the basic and shared algorithms need to count for each pattern

length. We can see in figure 8 that shared is able to prune a very significant number of candidates from consideration. Basic on the other hand has to collect counts for a very large number of patterns that end up being infrequent, this increases the memory usage and slows down the algorithm. This issue was evidenced in other experiments when the number of candidates was so large that basic could not fit them into memory. We can also see in the figure that shared considers patterns only up to length 8, while basic considers patterns all the way to length 12. This is because basic is considering long transactions that include items and their ancestors.

6 Related Work

Software research into management of RFID data is divided into three areas. The first is concerned with secure collection and management of online tag related information [14,15]. The second is cleaning of errors present in RFID data due to error inaccuracies, [13,12]. The third is related to the creation of multi-dimensional warehouses capable of providing OLAP operations over a large RFID data set [7,6].

The design of the *path cube* and the *workflow cube* shares many common principles with the traditional data cube [8]. They both aggregate data at different levels of abstraction in multi-dimensional space. Since each dimension has an associated concept hierarchy, both can be (at least partially) modeled by a *Star* schema. The problem of deciding which cuboids to construct in order to provide efficient answers to a variety of queries specified at different abstraction levels is analogous to the problem of partial data cube materialization studied in [10,16]. However, the *path cube* differs from a traditional data cube in that it also models *object transitions* in multi-dimensional space.

The *workflow cube* also shares common principles with the above lines of research but it differs from it in that our measure is a complex probabilistic model and not just an aggregate such as count or sum, and that our aggregates deal with two interrelated abstraction lattices, one for item dimensions and the other for path dimensions. Induction of workflows from RFID data sets shares many characteristics with the problem of process mining [18]. Workflow induction, the area possibly closest to our work, studies the problem of discovering the structure of a workflow from event logs. [1] first introduced the problem of process mining and proposed a method to discover workflow structure, but for the most part their methods assumes no duplicate activities in the workflow and does not take activity duration into account, which is a very important aspect of RFID data. Another area of research very closed to flowgraph construction is that of grammar induction [4,17], the idea is to take as input a set of strings and infer the probabilistic deterministic finite state automaton (PDFA) that generated the strings. This approach differs from ours in that it does not consider exceptions to transition probability distributions, or duration distributions at the nodes.

7 Towards Fruitful Research on Mining RFID Data Warehouses

The systematic data cleaning, integration, compression and aggregation in the construction of RFID data warehouses provide a valuable source of integrated and compressed data as well as a powerful infrastructure for mining RFID data. As indicated in our

scattered discussions in the previous sections, frequent pattern analysis, data flow aggregation and analysis, and exception analysis have been integrated in the process of construction of such data warehouses. With the construction and incremental maintenance of such an integrated data warehouse, it is possible to systematically develop scalable and versatile data mining functions in such an RFID data warehouse system.

Almost all kinds of data mining functions that can be associated with traditional data warehouses [9] can find their applications at mining RFID data warehouses. However, due to the tremendous amount of FRID data and its special analysis tasks, there are some particular mining tasks that may play more important roles in RFID data mining than others. Since RFID data mining is just an emerging research direction, here we only briefly outline a few such tasks. Interested researchers may likely find exciting new directions by their own research. First, due to the hardly avoidable inaccuracy or errors in RFID reading at various circumstances, RFID data likely contain noise, missing data, or erroneous readings. Data mining will help build cleansed and integrated data warehouse by finding regularities of data movements, cross-checking exceptions and outliers, and performing inference based on expert-provided or discovered rules. Second, with the bulky RFID data, one may like to know summary, statistics, and the characteristic and discriminant features of the FRID data in multi-dimensional space, especially those related to time, location, product category, and so on. A data cube-based aggregation and statistical analysis will be an important component in RFID data analysis. Third, although it is important to understand the general characteristics and trends of RFID data, it is important to detect outliers and exceptions, especially those related to data flow and object movements. Fourth, due to the incremental updates of bulky RFID data, the mining for changes of data in multi-dimensional space in an incremental manner could be another importance theme of study. Finally, due to the multi-dimensional nature of FRID data warehouse, it is critical to develop scalable and OLAP-based multi-dimensional data mining methods so that the analysis can be performed in an online analytical processing manner, either by human interaction or by automated processes.

Based on the above analysis, we believe the tremendous amount of RFID data provides an exciting research frontier in data mining as well as a fertile ground for further research across multiple disciplines.

8 Conclusions

We have proposed two novel models to warehouse RFID data that allow high-level analysis to be performed efficiently and flexibly in multi-dimensional space. The first model, the *path cube* takes advantage of bulky object movements to construct a highly compressed representation of RFID data that can be used to compute a variety of path dependent aggregates. The second model, the *workflow cube* summarizes major flows and significant flow exceptions in an RFID data set, and can be used to discover interesting patterns in the movement of commodities through an RFID application.

The *path cube* is composed of a hierarchy of compact summaries (*RFID-Cuboids*) of the RFID data aggregated at different abstraction levels where data analysis can take place. Each *RFID-Cuboid* records item movements in the *Stay*, *Info*, and *Map* tables that take advantage of the fact that individual items tend to move and stay together

(especially at higher abstraction levels) to collapse multiple movements into a single record without loss of information.

The *workflow cube* is a model useful in analyzing item flows in an RFID application by summarizing item paths along the dimensions that describe the items, and the dimensions that describe the path stages. Each cell has a probabilistic workflow as measure, this workflow is a concise representation of general flow trends and significant deviations from the trends. The *workflow cube* facilitates the discovery of trends in the movement of items at different abstraction levels. It also provides views of the data that are tailored to the needs of each user.

Notice that both of these models work well when our underlying assumption of bulky object movements, and major flow patterns with a small number of exceptions are valid. This fits a good number of RFID applications, such as supply chain management. However, there are also other applications where RFID data may not have such characteristics. We believe that further research is needed to construct efficient models for such applications. Finally, we view mining patterns, rules and outliers from RFID data in RFID data warehouses as a promising research frontier with broad applications.

Acknowledgement. The work was supported in part by the U.S. National Science Foundation NSF IIS-03-08215/05-13678 and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proc. 1998 Int. Conf. Extending Database Technology (EDBT'98)*, pages 469–483, Valencia, Spain, Mar. 1998.
2. R. Agrawal and R. Srikant. Fast algorithm for mining association rules in large databases. In *Research Report RJ 9839*, IBM Almaden Research Center, San Jose, CA, June 1994.
3. K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 359–370, Philadelphia, PA, June 1999.
4. R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proc. 1994 Int. Col. Grammatical Inference (ICGI'94)*, pages 139–152, Alicante, Spain, Sept. 1994.
5. S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26:65–74, 1997.
6. H. Gonzalez, J. Han, and X. Li. Flowcube: Constructuing RFID flowcubes for multi-dimensional analysis of commodity flows. In *Proc. 2006 Int. Conf. Very Large Data Bases (VLDB'06)*, Seoul, Korea, Sept. 2006.
7. H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analysis of massive RFID data sets. In *Proc. 2006 Int. Conf. Data Engineering (ICDE'06)*, Atlanta, Georgia, April 2006.
8. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54, 1997.
9. J. Han and M. Kamber. *Data Mining: Concepts and Techniques* (2nd ed.). Morgan Kaufmann, 2006.

10. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, pages 205–216, Montreal, Canada, June 1996.
11. Venture development corporation (vdc). In *http://www.vdc-corp.com/*.
12. S. R. Jeffery, G. Alonso, and M. J. Franklin. Adaptive cleaning for RFID data streams. In *Technical Report UCB/EECS-2006-29*, EECS Department, University of California, Berkeley, March 2006.
13. S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *Proc. 2006 Int. Conf. Data Engineering (ICDE'06)*, Atlanta, Georgia, April 2006.
14. S. Sarma, D. L. Brock, and K. Ashton. The networked physical world. In *White paper, MIT Auto-ID Center*, http://archive.epcglobalinc.org/publishedresearch/MIT-AUTOID-WH-001.pdf, 2000.
15. S. E. Sarma, S. A. Weis, and D. W. Engels. RFID systems, security & privacy implications. In *White paper, MIT Auto-ID Center*, http://archive.epcglobalinc.org/publishedresearch/MIT-AUTOID-WH-014.pdf, 2002.
16. A. Shukla, P. M. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 488–499, New York, NY, Aug. 1998.
17. F. Thollard, P. Dupont, and C. dela Higuera. Probabilistic DFA inference using kullback-leibler divergence and minimality. In *Proc. 2000 Int. Conf. Machine Learning (ICML'00)*, pages 975–982, Stanford, CA, June 2000.
18. W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng*, 16:1128–1142, 2004.