FlowCube: Constructing RFID FlowCubes for Multi-Dimensional Analysis of Commodity Flows*

Hector Gonzalez Jiawei Han Xiaolei Li

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

ABSTRACT

With the advent of RFID (Radio Frequency Identification) technology, manufacturers, distributors, and retailers will be able to track the movement of individual objects throughout the supply chain. The volume of data generated by a typical RFID application will be enormous as each item will generate a complete history of all the individual locations that it occupied at every point in time, possibly from a specific production line at a given factory, passing through multiple warehouses, and all the way to a particular checkout counter in a store. The movement trails of such RFID data form gigantic commodity flowgraph representing the locations and durations of the path stages traversed by each item. This commodity flow contains rich multi-dimensional information on the characteristics, trends, changes and outliers of commodity movements.

In this paper, we propose a method to construct a warehouse of commodity flows, called *flowcube*. As in standard OLAP, the model will be composed of cuboids that aggregate item flows at a given abstraction level. The *flowcube* differs from the traditional data cube in two major ways. First, the measure of each cell will not be a scalar aggregate but a commodity *flowgraph* that captures the major movement trends and significant deviations from the superset of objects in the cell. Second, each *flowgraph* itself can be viewed at multiple levels by changing the level of abstraction of path stages. In this paper, we motivate the importance of the model, and present an efficient method to compute it by (1) performing simultaneous aggregation of paths to all interesting abstraction levels, (2) pruning low support path segments along the item and path stage abstraction lattices, and (3) compressing the cube by removing rarely occurring cells, and cells whose commodity flows can be inferred from higher level cells.

*The work was supported in part by the U.S. National Science Foundation NSF IIS-03-08215/05-13678 and NSF BDI-05-15813.

1. INTRODUCTION

With the rapid progress of *radio frequency identification* (RFID) technology, it is expected that in a few years, RFID tags will be placed at the package or individual item level for many products. These tags will be read by a transponder (RFID reader), from a distance and without line of sight. One or more readings for a single tags will be collected at every location that the item visits and therefore enormous amounts of object tracking data will be recorded. This technology can be readily used in applications such as item tracking and inventory management, and thus holds a great promise to streamline supply chain management, facilitate routing and distribution of products, and reduce costs by improving efficiency. However, the enormous amount of data generated in such applications also poses great challenges on efficient analysis.

Let us examine a typical such scenario. Consider a nationwide retailer that has implemented RFID tags at the pallet and item level, and whose managers need to analyze the movement of products through the entire supply chain, from the factories producing items, to international distribution centers, regional warehouses, store backrooms, and shelves, all the way to checkout counters. Each item will leave a trace of readings of the form (EPC, *location*, *time*) as it is scanned by the readers at each distinct location[1]. If we consider that each stores sells tens of thousands of items every day, and that each item may be scanned hundreds of times before being sold, the retail operation may generate several terabytes of RFID data every day. This information can be analyzed from the perspective of paths and the abstraction level at which path stages appear, and from the perspective of items and the abstraction level at which the dimensions that describe an item are studied.

Path view. The set of locations that an item goes through forms a path. Paths are interesting because they provide insights into the patterns that govern the flow of items in the system. A single path can be presented in different ways depending on the person looking at the data. Figure 1 presents a path (seen in the middle of the figure) aggregated to two different abstraction lev-

[1] Electronic Product Code (EPC) is a unique identifier associated with each RFID tag

els, the path at the top of the figure shows the individual locations inside a store, while it collapses locations that belong to transportation. This view may be interesting to a store manager, that requires detailed transition information within the store. 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 transportation manager in the company.

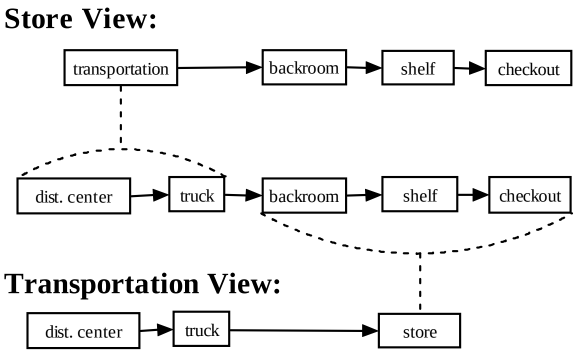

Figure 1: Path views: The same path can be seen at two different abstraction levels.

Item view. 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 dimensions describing its characteristics, *e.g.*, product, brand, manufacturer. Each of these dimensions has an associated concept hierarchy. Figure 2 presents the different levels at which a single item may be looked at along the product dimension. It is possible that a high level manager at a retailer will only look at products at the category level. But that the manager for a particular line of products may look at individual items in that line.

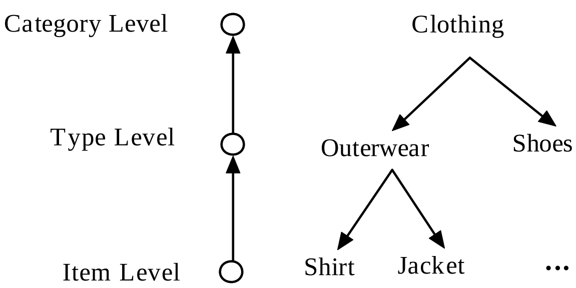

Figure 2: Item view: A product can be seen at different levels of abstraction

The key challenge in constructing a data cube for a database 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. Full materialization of such data cube would be unrealistic as the number of abstraction levels is exponential in the number of dimensions describing an item and describing a path.

In this paper we propose flowcube, a data cube model that summarizes commodity flows at multiple levels of abstraction along the item view and the path view of RFID data. This model will provide answers to questions such as:

1. What are the most typical paths, with average duration at each stage, that shoes manufactured in China take before arriving to the L.A. distribution center, and list the most notable deviations from the typical paths that significantly increase total lead time before arrival?

2. Present a summarized view of the movements of electronic goods in the northeast region and list the possible correlations between the durations spent by items at quality control points in the manufacturing facilities and the probability of being returned by customers.

3. Present a workflow that summarizes the item movement across different transportation facilities for the year 2006 in Illinois, and contrast path durations with historic flow information for the same region in 2005.

The measure of each cell in the flowcube is called a flowgraph, which is a tree shaped probabilistic workflow, where each node records transition probabilities to other nodes, and the distribution of possible durations at the node. Additionally nodes keep information on exceptions to the general transition and duration distributions given a certain path prefix that has a minimum support (occurs frequently in the data set). For example, the flowgraph may have a node for the factory location that says that items can move to either the warehouse or the store locations with probability 60% and 40% respectively. But it may indicate that this rule is violated when items stay for more than 1 week in the factory in which case they move to the warehouse with probability 90%.

Computation of the flowgraph for each cell of the flowcube can be divided into two steps. The first is to collect the necessary counts to find the transition and duration probabilities for each node. This can be done efficiently in a single pass over the paths aggregated in the cell. The second is to compute the flowgraph exceptions, this is a more expensive operation as it requires computing all frequent path segments in the cell, and checking if they cause an exception. In this paper we will focus on the problem of how to compute frequent path segments for every cell in the flowcube in an efficient manner. The technical contribution of the paper can be summarized as follows:

1. **Shared computation.** We explore efficient computation of the flowcube by sharing the computation of frequent cells and frequent path segments simultaneously. Similar to shared computation of multiple cuboids in BUC-like computation [4], we propose to compute frequent cells in the flowcube and

frequent path segments aggregated at every interesting abstraction level simultaneously. For example, in a single scan of the path database we can collect counts for items at the product level and also at the product category level. Furthermore, we can collect counts for path stages with locations at the lowest abstraction level, and also with locations aggregated to higher levels. The concrete cuboids that need to be computed will be determined based on the cube materialization plan derived from application and cardinality analysis. Shared computation minimizes the number of scans of the path database by maximizing the amount of information collected during each scan. In order to efficiently compute frequent cells and frequent path segments we will develop an encoding system that transforms the original path database into a transaction database, where items encode information on their level along the item dimensions, and stages encode information on their level along the path view abstraction levels.

2. **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. Similarly, if the clothing category is not frequent, no particular shirt can be frequent. In our proposed method we do not incur extra scans of the path database for pre-counting, we instead integrate this step with the collection of counts for a given set of candidates of a given length.

3. **Cube compression by removing redundancy and low support counts.** We reduce the size of the flowcube by exploring two strategies. The first is to compute only those cells that contain only a minimum number of paths (iceberg condition). This makes sense as the flowgraph 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 flowgraphs that are non-redundant given higher abstraction level flowgraphs. 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 flowgraph we can infer the 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 in the flowcube.

The rest of the paper is organized as follows. Section 2 presents the structure of the path database. Section 3 introduces the concept of flowgraphs. Section 4, defines the flowcube, and the organization of the cuboids that compose it. Section 5, develops an efficient method to compute frequent patterns for every cell of a flowcube. Section 6, reports on experimental and performance results. We discuss related work in Section 7 and conclude our study in section 8.

2. PATH DATABASE

An RFID implementation usually generates a stream of data of the form $(EPC, location, time)$ where EPC is an electronic product code associated with an item, location is the place were the tag was read by a scanner, and time is when the reading took place. If we look at all the records associated to a particular item and sort them on time, they will form a path. After data cleaning, each path will have stages of the form $(location, time\_in, time\_out)$. In order to study the way patterns flow through locations we can discard absolute time and only focus on relative duration, in this case the stages in each path are of the form $(location, duration)$. Furthermore, duration may not need to be at the precision of seconds, we could discretize the value by aggregating it to a higher abstraction level, clustering, or using any other numerosity reduction method.

A path database is a collection of tuples of the form $\langle d_1, ..., d_m : (l_1, t_1)...(l_k, t_k)\rangle$, where each $d_1, ..., d_m$ are path independent dimensions (the value does not change with the path traversed by the item) that describe an item, *e.g.*, product, manufacturer, price, purchase date. The pair (l_i, t_i) tells us that the item was at location l_i for a duration of t_i time units.

Table 1 presents a path database with 2 path independent dimensions: product and brand. The nomenclature used for stage locations is d for distribution center, t for truck, w for warehouse, s for store shelf, c for store checkout, and f for factory.

id	product	brand	path
1	tennis	nike	$(f,10)(d,2)(t,1)(s,5)(c,0)$
2	tennis	nike	$(f,5)(d,2)(t,1)(s,10)(c,0)$
3	sandals	nike	$(f,10)(d,1)(t,2)(s,5)(c,0)$
4	shirt	nike	$(f,10)(t,1)(s,5)(c,0)$
5	jacket	nike	$(f,10)(t,2)(s,5)(c,1)$
6	jacket	nike	$(f,10)(t,1)(w,5)$
7	tennis	adidas	$(f,5)(d,2)(t,2)(s,20)$
8	tennis	adidas	$(f,5)(d,2)(t,3)(s,10)(d,5)$

Table 1: Path Database

3. FLOWGRAPHS

A duration independent flowgraph 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 flowgraph, the most direct way is to create nodes for every combination of location and duration. This option has the disadvantage of generating very large flowgraphs. A second option is to annotate each node in the duration independent flowgraph with a distribution of possible durations at the node. This approach keeps the size of the flowgraph 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 to the time it spent in the store backroom; and (ii) transition probabilities are independent of duration, *e.g.*, the probability of a box of milk to transition 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 model which we call flowgraph, that not only records duration and transition distributions at each node, but that 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 a flowgraph 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 flowgraph; and the purpose of δ to prevent the exceptions in the flowgraph from being dominated by statistical noise in the path database. Figure 3 presents a flowgraph for the path database in Table 1.

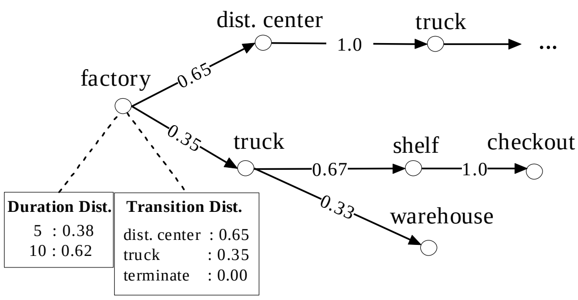

Figure 3: Flowgraph

The flowgraph in Figure 3 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%.

Definition 3.1. *(Flowgraph) A Flowgraph is a tuple (V, D, T, X), where V is the set of nodes, each node corresponds to a unique path prefix in the path database. D is a set of multinomial distributions, one per node, each assigns a probability to each distinct duration at a node. T is a set of multinomial distributions, one per node, each assigns a probability to each possible transition from the node to every other node, including the termination probability. X is the set of exceptions to the transition and duration distributions for each node.*

Computing a flowgraph can be done efficiently by: (1) constructing a prefix tree for the path database (2) annotating each node with duration and transition probabilities (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. FLOWCUBE

The next step we take in our model is to combine flowgraph analysis with the power of OLAP type operations such as drill-down and roll-up. It may be interesting for example to look at the evolution of the flowgraphs for a certain product category over a period of time to detect how a change in suppliers may have affected the probability of returns for a particular item. We could also use multidimensional analysis to compare the speed at which products from two different manufacturers move through the system, and use that information to improve inventory management policies. Furthermore, it may be interesting to look at paths traversed by the items from different perspectives. A transportation manager may want to look at flowgraphs that provide great detail on truck, distribution centers, and sorting facilities while ignoring most other locations. A store manager on the other hand may be more interested in looking at movements from backrooms, to shelfs, checkout counters, and return counters and largely ignore other locations.

In this section we will introduce the concept of a flowcube, which is a data cube computed on an RFID path database, where each cell summarizes commodity flows at at a given abstraction level of the path independent dimensions, and path stages. The measure recorded in each cell of the flowcube is a flowgraph computed on the paths belonging to the cell.

In the next sections we will explore in detail the different components of a flowcube. We will first introduce the concepts of item abstraction lattice and path abstraction lattice, which are important to give a more precise definition of the cuboid structure of a flowcube. We will then study the computational challenges of using flowgraphs as measures. Finally we introduce the concepts of non-redundant flowcubes, and iceberg flowcubes as a way to reduce the size of the model.

4.1 Abstraction Lattice

Each dimension in the flow cube can have an associated concept hierarchy. A concept hierarchy is a tree where nodes correspond to concepts, and edges correspond to *is-a* relationships between concepts. The most concrete concepts reside at the leafs of the tree, while the most general concept, denoted '*', resides at the apex of the tree and represents *any* concept. The level of abstraction of a concept in the hierarchy is the level at which the concept is located in the tree.

Item Lattice. The abstraction level of the items in the path database can be represented by the tuple $(l_1, ..., l_m)$, where l_i is the abstraction level of the path independent dimension d_i. For our running example we can say that the items in the path database presented reside at the lowest abstraction level. The set of all item abstraction levels forms a lattice. A node n_1 is higher in the lattice than a node n_2, denoted $n_1 \preceq n_2$ if the levels all dimensions in n_1 are smaller or equal to the ones in n_2.

Table 2 shows the path independent dimension from Table 1 with the product dimension aggregated one level higher in its concept hierarchy. The "path ids" column lists the paths in the cell, each number corresponds to the path id in Table 1. We can compute a flowgraph on each cell in Table 2. Figure 4 presents the flowgraph for the cell (outerwear, nike).

product	brand	path ids
shoes	nike	1,2,3
shoes	adidas	7,8
outerwear	nike	4,5,6

Table 2: Aggregated Path Database

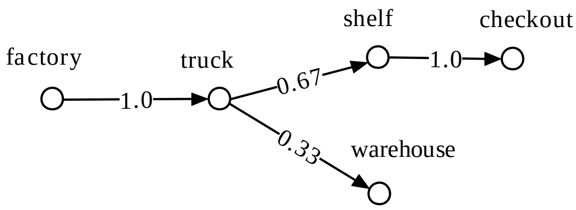

Figure 4: Flowgraph for cell (outerwear, nike, 99)

Path Lattice. In the same way that items can be associated with an abstraction level, path stages will also reside at some level of the location and duration concept hierarchies. Figure 5 presents an example concept hierarchy for the location dimension of the path stages. The shadowed nodes are the concepts that are important for analysis; in this case the data analyst may be a transportation manager that is interesting in seeing the transportation locations at a full level of detail, while aggregating store and factory locations to a higher level. More formally, the path abstraction level is defined by the tuple $(\langle v_1, v_2, ..., v_k \rangle, t_l)$ where each v_i is a node in the location concept hierarchy, and t_l the level in the time concept hierarchy. Analogously to the item abstraction lattice definition, we can define a path abstraction lattice.

In our running example, assuming that time is at the hour level, the path abstraction level corresponding to Figure 5 is (⟨dist. center, truck, warehouse, factory, store⟩ ,hour).

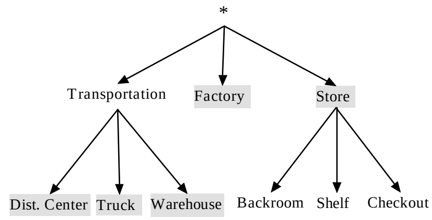

Figure 5: Location Concept Hierarchy

We aggregate a path to abstraction level $(\langle v_1, v_2, ..., v_k \rangle, t_l)$ in two steps. First, we aggregate the location in each stage to its corresponding node v_i, and we aggregate its duration to the level t_l. Second, we merge consecutive locations that have been aggregated to the same concept level. The merging of consecutive locations requires us to define a new duration for the merged stage. The computation of the merged duration would depend on the application, it could be as simple as just adding the individual durations, or it could involve some form of numerosity reduction based on clustering or other well known methods.

Aggregation along the path abstraction lattice is unique to flowcubes and is quite different to 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 which changes. This distinct property requires us to develop new methods to construct a flowcube that has aggregation for both item and path dimensions.

Definition 4.1 (Flowcube). *A flowcube is a collection of cuboids. A cuboid is a grouping of entries in the fact table into cells, such that each cell shares the same values on the item dimensions aggregated to an item abstraction level I_l; and the paths in the cell have been aggregated to a path abstraction level P_l. The measure of a cell is a flowgraph computed on the paths in the cell. A cuboid can be characterized by the pair $\langle I_l, P_l \rangle$.*

4.2 Measure computation

We can divide a flowgraph into two components, the first is the duration and transition probability distributions, the second is the set of exceptions. In this section we will show that while the first component is an algebraic measure, and thus can be computed efficiently, the second component is a holistic measure and requires special treatment.

Assume that we have a dataset S that has been partitioned into k subsets $s_1, ..., s_k$ such that $S = \bigcup_i s_i$ and $s_i \cap s_j = \phi$ for all $i \neq j$. We say that a measure, or function, is algebraic if the measure for S can be computed by collecting M (positive bounded integer) values from each subset s_i. For example, average is distributive as we can collect count() and sum() ($M = 2$) from each subset to compute the global average. A holistic measure on the other hand is one where there is no constant bound on the number of values that need to be collected from each subset in order to compute the function for S. Median is an example of a holistic function, as each subset will need to provide its entire set of values in order to compute the global median.

LEMMA 4.2. *The duration and transition distributions of a flowgraph are algebraic measures.*

Proof Sketch. Each node n in the flowgraph contains a duration probability distribution d and a transition probability distribution t. For each node in the flowgraph, $d(t_i) = count(t_i)/\sum_{i=1}^{k} count(t_i)$, where $d(t_i)$ is the probability of duration t_i, $count(t_i)$ is the number of items that stayed at the node for duration t_i and k is the number of distinct durations at the node. We can compute d for each node in a flowgraph whose dataset has been partitioned into subsets, by collecting the following k values $count(t_1), ..., count(t_k)$. Similarly we can argue that the transition distribution t can be computed by collecting a fixed number of transition counts from each subset. Given that the number of nodes and distinct durations per node in a flowgraph is fixed (after numerosity reduction), we can collect a bounded number of counts from each subset to compute the duration and transition distributions for the flowgraph. ■

The implication of lemma 4.2 is that we can compute a flowcube efficiently by constructing high level flowgraphs from already materialized low level ones without having to go to the path database.

LEMMA 4.3. *The set of exceptions in a flowgraph is a holistic measure.*

Proof Sketch. The flowgraph exceptions are computed on the frequent itemsets in the collection of paths aggregated in the cell, thus proving that a function that returns the frequent itemsets for a cell is not algebraic is sufficient. Assume that the set S is the union of $s_1, ..., s_n$, and that the sets $f_1, ..., f_n$ are the frequent itemsets for each subset s_i. We need to compute F the frequent itemsets for S, assume that f_i^j is frequent pattern j in set i, in order to check if f_i^j is frequent in S we need to collect its count on every subset s_k, and thus we need every subset to provide counts for every frequent pattern on any other subset, this number is clearly unbounded as it depends on the characteristics of each data set. ■

The implication of lemma 4.3 is that we can not compute high level flowgraphs from low level ones by just passing a fixed amount of information between the levels. But we can still mine the frequent patterns required to determine exceptions in each cell in a very efficient way by entirely avoiding the level by level computation approach and instead using a novel shared computation method that simultaneously finds frequent patterns for cuboids at every level of abstraction. In section 5 we will develop the mining method in detail.

4.3 Flowgraph Redundancy

The flowgraph registered for a given cell in a flowcube 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 flowgraph in the cell. For example, if we have a flowgraph G_1 for milk, and a flowgraph G_2 from milk 2% (milk is an ancestor of milk 2% in the item abstraction lattice), and $G_1 = G_2$ we see that G_2 is redundant, as it can be inferred from G_1.

Before we give a more formal definition of redundancy we need a way to determine the similarity of two flowgraphs. A similarity metric between two flowgraphs is a function $\varphi : G_1 \times G_2 \rightarrow \mathbb{R}$. Informally the value of $\varphi(G_1, G_2)$ is large if the G_1 and G_2 are similar and small otherwise. There are many options for φ and the one to use should use depends on the particular RFID application semantics. One possible funtion is to use the KL-Divergence of the probability distributions induced by two flowgraphs. But other similarity metrics, based for example on probabilistic deterministic finite automaton (PDFA) distance could be used. Note that we do not require φ to be a real metric in the mathematical sense, in that the triangle inequality does not necessarily need to hold.

DEFINITION 4.4 (REDUNDANT FLOWGRAPH). *Let G be a flowgraph for cell c, let $p_1, ..., p_n$ be all the cells in the item lattice that are a parent of c and that reside in a cuboid at the same level in the path lattice as c's cuboid. Let $G_1, ..., G_n$ be the flowgraphs for cells $p_1, ..., p_n$, let φ be a flowgraph similarity metric. We say that G is redundant if $\varphi(G, G_i) > \tau$ for all i, where τ is the similarity threshold.*

A flowcube that contains only cells with non-redundant flowgraphs is called a non-redundant flowcube. A non-redundant flowcube can provide significant space savings when compared to a complete flowcube, but more interestingly, it 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. For example, using a non-redundant flowcube we can quickly determine that milk from every manufacturer has very similar flow patterns, except for the milk from farm A which has significant differences. Using this information the data analyst can drill down and slice on farm A to determine what factors make its flowgraph different.

4.4 Iceberg Flowcube

A flowgraph is a statistical model that describes the flow behavior of objects given a collection of paths. If the data set on which the flowgraph is computed is very small, the flowgraph 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 flowcube that contain at least δ paths (minimum support). For example, if we set the minimum support to 2, the cell $(shirt, *)$ from Table 1 will not be materialized as it contains only a single path.

DEFINITION 4.5 (ICEBERG FLOWCUBE). *A flowcube that contains only cells with a path count larger than δ is called an Iceberg flowcube.*

Iceberg flowcubes can be computed efficiently by using apriori pruning of infrequent cells. We can materialize the cube from low abstraction levels to high abstraction ones. 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. The algorithm we develop in the next section will make extensive use of this property to speed up the computation of the flowcube.

5. ALGORITHMS

In this section we will develop a method to compute a non-redundant iceberg flowcube given an input path database. The problem of flowcube construction can be divided into two parts. The first is to compute the flowgraph for each frequent cell in the cube, and the second is to prune uninteresting cells given higher abstraction level cells. The second problem can be solved once the flowcube has been materialized, by traversing the cuboid lattice from low to high abstraction levels, while pruning cells that are found to be redundant given the parents. In the rest of this paper we will focus on solving the first problem.

The key computational challenge in materializing a flowcube is to find the set of frequent path segments, aggregated at every interesting abstraction level, for every cell that appears frequently in the path database. Once we have the counts for every frequent pattern in a cell determining exceptions can be done efficiently by just checking if counts of these patterns change the duration or transition probability distributions for each node. The problem of mining frequent patterns in the flowcube is very expensive as we need to mine frequent paths at every cell, and the number of cells is exponential in the number of item and path dimensions. Flowcube materialization combines two of the most expensive methods in data mining, cube computation, and frequent pattern mining. The method that we develop in this section solves these two problems with a modified version of the Apriori algorithm [3], by collecting frequent pattern counts at every interesting level of the item and path abstraction lattices simultaneously, while exploiting cross-pruning opportunities between these two lattices to reduce the search space as early as possible. To further improve performance our algorithm will use partial materialization to restrict the set of cuboids to compute, to those most useful to each particular application. The algorithm is based on the following key ideas:

Construction of a transaction database. In order to run a frequent pattern algorithm on both the item and path dimensions at every abstraction level, we need to transform the original path database into a transaction database. Values in the path database need to be transformed into items that encode their concept hierarchy information and thus facilitate efficient multi-level mining. For example, the value "jacket" for the product dimension in Table 1, can be encoded as "112", the first digit indicates that it is a value of the first path independent dimension, the second digit indicates that is of type outerwear, and the third digit tells us that it is a jacket (for brevity we omit the encoding for product category as all the products in our example belong to the same category: clothing). Path stages require a slightly different encoding, in addition to recording the concept hierarchy for the location and time dimensions for the stage, each stage should also record the path prefix leading to the stage so that we can do multi-level path aggregation. For example the stage (t,1) in the first path of the path database in Table 1 can be encoded as (fdt,10), to mean that it is the third stage in the path: factory → dist. center → truck, and that it has a duration of 10 time units.

Table 3 presents the transformed database resulting from the path database from Table 1.

TID	Items
1	{121,211,(f,10),(fd,2),(fdt,1),(fdts,5),(fdtsc,0)}
2	{121,211,(f,5),(fd,2),(fdt,1),(fdts,10),(fdtsc,0)}
3	{122,211,(f,10),(fd,1),(fdt,2),(fdts,5),(fdtsc,0)}
4	{111,211,(f,10),(ft,1),(fts,5),(ftsc,0)}
5	{112,211,(f,10),(ft,2),(fts,5),(ftsc,1)}
6	{112,211,(f,10),(ft,1),(ftw,5)}
7	{121,221,(f,5),(fd,2),(fdt,2),(fdts,20)}
8	{121,221,(f,5),(fd,2),(fdt,3),(fdts,10),(fdtsd,5)}

Table 3: Transformed transaction database

Shared counting of frequent patterns. In order to minimize the number of scans of the transformed transaction database we share the counting of frequent patterns at every abstraction level in a single scan. Every item that we encounter in a transaction contributes to the support of all of its ancestors on either the item or path lattices. For example, the item 112 (jacket) contributes to its own support and the support of its ancestors, namely, 11* (outerwear) and 1** (we will later show that this ancestor is not really needed). Similarly an item representing a path stage contributes to the support of all of its ancestors along the path lattice. For example, the path stage (fdts,10) will support its own item and items such as (fdts,*), (fTs,10) and (fTs,*), where f stands for factory, d for distribution center, t for truck, s for shelf, and T for transportation (T is the parent of d, and t, in the location concept hierarchy).

Shared counting processes patterns from short to long. In the first scan of the database we can collect all the patterns of length 1, at every abstraction level in the item and path lattices. In the second scan we check the frequency of candidate patterns of length 2 (formed by joining frequent patterns of length 1). We continue this process until no more frequent patterns are found. Table 4 presents a portion of the frequent patterns of length 1 and length 2 for the transformed path database of Table 3.

Pruning of infrequent candidates. When we generate candidates of length $k+1$ based on frequent patterns of length k we can apply several optimization techniques to prune large portions of the candidate space:

Length 1 frequent

Itemset	Support
{121}	5
{12*}	5
{(f,10)}	5
{(f,*)}	8
{(fd,2)}	4
...	...

Length 2 frequent

Itemset	Support
{12*,211}	3
{12*,21*}	3
{211,(f,10)}	4
{(f,5)(fd,2)}	3
{(f,*),(fd,*)}	3
...	...

Table 4: Frequent Itemsets

- **Precounting of frequent itemsets at high levels of abstraction along the item and path lattices**. We can take advantage of the fact that infrequent itemsets at high abstraction levels of the item and path lattice can not be frequent at low abstraction levels. We can implement this strategy by, for example, counting frequent patterns of length 2 at a high abstraction level while we scan the database to find the frequent patterns of length 1. A more general precounting strategy could be to count high abstraction level patterns of length $k+1$ when counting the support of length k patterns.
- **Pruning of candidates containing two unrelated stages**. Given our stage encoding, we can quickly determine if two stages can really appear in a the same path, and prune all those candidates that contain stages that can not appear together. For example we know that the stages $(fd, 2)$ and $(fts, 5)$ can never appear in the same path, and thus should not be generated as a candidate.
- **Pruning of path independent dimensions aggregated to the highest abstraction level**. We do not need to collect counts for an item such as 1**, this item really mean any value for dimension 1; and its count will always be the same as the size of the transaction table. These type of items can be removed from the transaction database.
- **Pruning of items and their ancestors in the same transaction**. This optimization was introduced in [17] and is useful for our problem. We do not need to count an item and any of its ancestors in the same candidate as we know that the ancestor will always appear with the item. This optimization can be applied to both path independent dimension values, and path stages. For example we should not consider the candidate itemset $\{121, 12*\}$ as its count will be the same of the itemset $\{121\}$.

Partial Materialization. Even after applying all the optimizations outlined above, and removing infrequent and redundant cells the size of of flowcube can still be very large in the cases when we have a high dimensional path database. Under such conditions we can use the techniques of partial materialization developed for traditional data cubes [12, 11, 16].

One strategy that seems especially well suited to our problem is that of partial materialization described in [11], which suggests the computation of a layer of cuboids at a minimum abstraction level that is interesting to users, a layer at an observation level where most analysis will take place, and the materialization of a few cuboids along popular paths in between these two layers.

5.1 Shared algorithm

Based on the optimization techniques introduced in the previous section, we propose algorithm Shared which is a modified version of the Apriori algorithm [3] used to mine frequent itemsets. Shared simultaneously computes the frequent cells, and the frequent path segments aggregated at every interesting abstraction level along the item and path lattices. The output of the algorithm can be used to compute the flowgraph for every cell that passes the minimum support threshold in the flowcube.

Algorithm 1 Shared

Input: A path database D, a minimum support δ
Output: Frequent cells and frequent path segments in every cell
Method:

In one scan of the path database compute the transformed path database into D', collect frequent items of length 1 into L_1, and pre-count patterns of length > 1 at high abstraction levels into P_1.
for $k = 2, L_{k-1} \neq \phi, k{+}{+}$ **do**
    generate C_k by joining frequent patterns in L_{k-1}
    Remove from C_k candidates that are infrequent given the pre-counted set P_{k-1}, remove candidates that include stages that can not be linked, and remove candidates that contain an item and its ancestor.
    for every transaction t in D' **do**
        increment the count of candidates in C_k supported by t, and collect the counts of high abstraction level patterns of length $> k$ into P_k
    end for
    L_k = frequent items in C_k
end for
Return $\bigcup_k L_k$.

5.2 Cubing Based Algorithm

A natural competitor to the Shared algorithm is an iceberg cubing algorithm that computes only cells that pass the iceberg condition on the item dimensions, and that for each such cell calls a frequent pattern mining algorithm to find frequent path segments in the cell. The precise cubing algorithm used in this problem is not critical, as long as the cube computation order is from high abstraction level to low level, because such order enables early pruning of infrequent portions of the cube. Examples of algorithms that fall into this category are BUC [4] and Star Cubing [20].

Algorithm 2 takes advantage of pruning opportunities based on the path independent dimensions, *i.e.*, if it detects that a certain value for a given dimension is infrequent, it will not check that value combined with another dimension because it is necessarily infrequent. What the algorithm misses is the ability to do pruning based on the path abstraction lattice. It will not, for example, detect that a certain path stage is infrequent in the highest abstraction level cuboid and thus will be

Algorithm 2 Cubing

Input: A path database D, a minimum support δ

Output: Frequent cells and frequent path segments in every cell

Method:

1: Divide D into two components D_i, which contains the path independent dimensions, and D_p which contains the paths.

2: Transform D_p into a transaction database by encoding path stages into items, and assign to each transaction a unique identifier.

3: Compute the iceberg cube C on D_i, use as measure the list of transaction identifiers aggregated in the cell.

4: **for** each cell c_i in C **do**

5: c_i^p = read the transactions aggregated in the cell.

6: c_i^f = find frequent patterns in c_i^p by using a frequent pattern mining algorithm

7: **end for**

8: return $\bigcup_i c_i^f$.

infrequent in every other cuboid, the algorithm will repeatedly generate that path stage as a candidate and check its support just to find that it is infrequent every single time. Another disadvantage of the cubing based algorithm is that it has to keep long lists of transaction identifiers as measures for the cells, when the lists are long, the input output costs of reading them can be significant. In our experiments even for moderately sized data sets these lists where much larger than the path database itself. With the algorithm shared, we only record frequent patterns, and thus our input output costs are generally smaller.

6. EXPERIMENTAL EVALUATION

In this section, we perform a thorough analysis our proposed algorithm (shared) and compare its performance against a baseline algorithm (basic), and against the cubing based algorithm (cubing) presented in the previous section. All experiments were implemented using C++ and were conducted on an Intel Pentium IV 2.4GHz System with 1GB of RAM. The system ran Debian Sarge with the 2.6.13.4 kernel and gcc 4.0.2.

6.1 Data Synthesis

The path databases used for our experiments were generated using a synthetic path generator that simulates the movement of items in a retail operation. We first generate the set of all valid sequences of locations that an item can take through the system. Each location in a sequence has an associated concept hierarchy with 2 levels of abstraction. The number of distinct values and skew per level are varied to change the distribution of frequent path segments. The generation of each entry in the path database is done in two steps. We first generate values for the path independent dimensions. Each dimension has a 3 level concept hierarchy. We vary the number of distinct values and the skew for each level to change the distribution of frequent cells. After we have selected the values for the path independent dimensions, we randomly select a valid location sequence from the list of possible ones, and generate a path by assigning a random duration to each location. The values for the levels in the concept hierarchies for path independent dimensions, stage locations, and stage durations, are all drawn from a Zipf distribution [21] with varying α to simulate different degrees of data skew.

For the experiments we compute frequent patterns for every cell at every abstraction level of the path independent dimensions, and for path stages we aggregate locations to the level present in the path database and one level higher, and we aggregate durations to the level present in the path database and to the any (*) level, for a total of 4 path abstraction levels.

In most of the experiments we compare three methods: Shared, Cubing, and Basic. Shared is the algorithm that we propose in section 5.1 and that does simultaneous mining of frequent cells and frequent path segments at all abstraction levels. For shared we implemented pre-counting of frequent patterns of length 2 at abstraction level 2, and pre-counting of path stages with duration aggregated to the '*' level. Cubing is an implementation of the algorithm described in section 5.2, we use a modified version of BUC [4] to compute the iceberg cube on the path independent dimensions and then called Apriori [3] to mine frequent path segments in each cell. Basic is the same algorithm as Shared except that we do not perform any candidate pruning based on the optimizations outlined in the previous section. In the figures we will use the following notation to represent different data set parameters $\mathcal{N}$ for the number of records, δ for minimum support, and d for the number of path independent dimensions.

6.2 Path database size

In this experiment we look at the runtime performance of the three algorithms when varying the size of path database, from 100,000 paths to 1,000,000 paths (disk size of 6 megabytes to 65 megabytes respectively). In Figure 6 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.

6.3 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 7 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 prun-

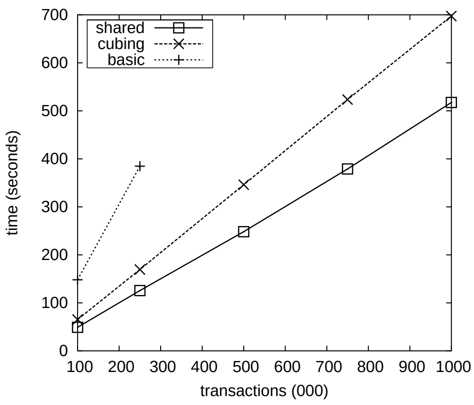

Figure 6: Database Size ($\delta = 0.01$, $d = 5$)

ing 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.

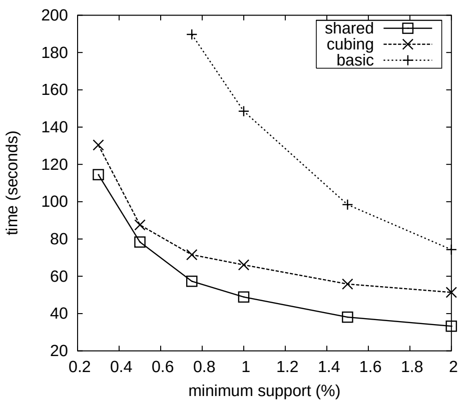

Figure 7: Minimum Support ($\mathcal{N} = 100,000$, $d = 5$)

6.4 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 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 8 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.

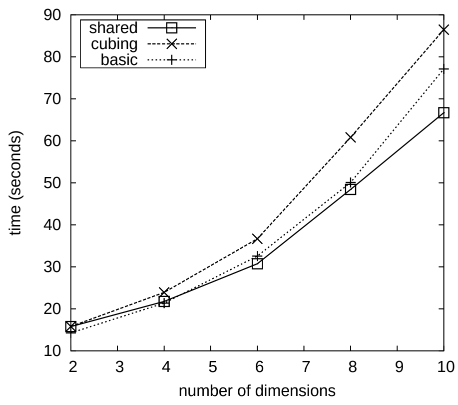

Figure 8: Number of Dimensions. ($\mathcal{N} = 100,000$, $\delta = 0.01$)

6.5 Density of the path independent dimensions

For this experiment we created three datasets with varying numbers of distinct values in 5 path independent dimensions. Dataset a had 2, 2, and 5 distinct values per level in every dimension; dataset b has 4, 4, and 6; dataset c has 5, 5, and 10. In Figure 9 we can see that as we increase the number of distinct items, data sparsity increases, and fewer frequent cells and path segments are found, which significantly improves the performance of all three algorithms. Due to the very large number of candidates we could not run the basic algorithm for dataset a.

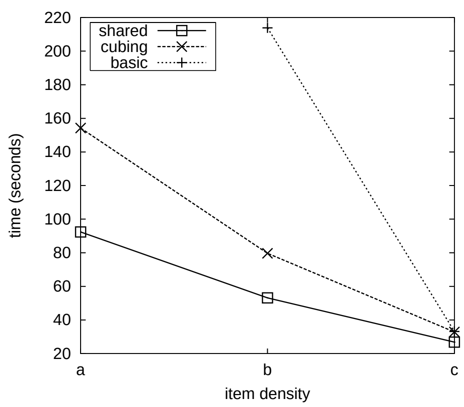

Figure 9: Item density ($\mathcal{N} = 100,000$, $\delta = 0.01$, $d = 5$)

6.6 Density of the paths

In this experiment we kept the density of the path independent dimensions constant and varied the density of the path stages by varying the number of distinct location sequences 10 to 150. We can see in Figure 10 that for small numbers of distinct path sequences, we have many frequent path fragments and thus mining is more expensive. But what is more important is that as the path database becomes denser shared gains a very significant advantage over cubing. The reason is that in a few scans of the database shared is able to detect every frequent path segment at every abstraction level, while cubing needs to do find frequent path segments independently for each frequent cell, and given the high density of paths, mining of frequent path segments is an expensive operation. We could not run the basic algorithm on this experiment as the number of candidates exploded with dense paths.

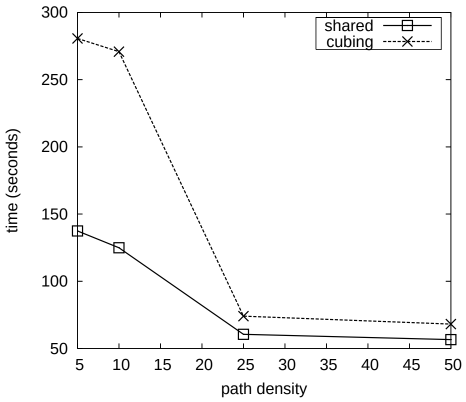

Figure 10: Path Density ($\mathcal{N} = 100,000$, $\delta = 0.01$, $d = 5$)

6.7 Pruning Power

This is experiment we show the effectiveness of the optimizations described in section 5 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 11 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. 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.

In this section we have verified that the ideas of shared computation, simultaneous mining of frequent patterns, and pruning techniques based on both item and path abstraction lattices are effective in practice and provide significant cost savings versus alternative algorithms.

7. RELATED WORK

RFID technology has been extensively studied from mostly two distinct areas: the electronics and radio communication technologies required to construct readers and tags [7]; and the software architecture required to securely collect and manage online information related to tags [14, 15]. More recently a third line of research dealing with mining of RFID data has emerged.

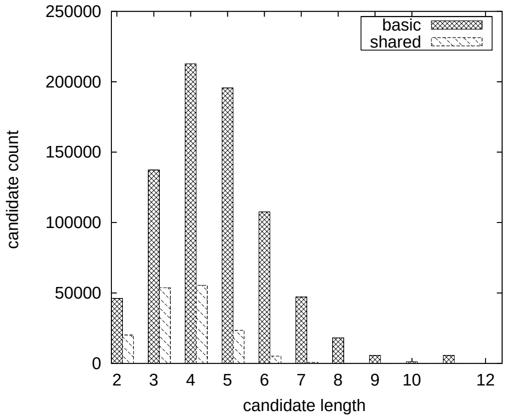

Figure 11: Pruning Power ($\mathcal{N} = 100,000$, $\delta = 0.01$, $d = 5$)

[6] introduces the idea of creating a warehouse for RFID data, but does not go into the data structure or algorithmic details; [8] presents a concrete warehouse architecture for RFID data; their model exploits characteristics of item movements to create a summary of large RFID data sets, but this summary is based on scalar aggregates and does not handle the concept complex measures such as flowgraphs.

Induction of flowgraphs from RFID data sets, shares many characteristics with the problem of process mining [19]. Workflow induction, may be the area closest to our work, it studies the problem of discovering the structure of a workflow from event logs represented by a list of tuples of the form $(case_i, activity_j)$ sorted by the time of occurrence; $case_i$ refers to the instance of the process, and $activity_j$ to the activity executed. [2] 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 [5, 18], 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.

There are several areas of data mining very related to our work. Data cubes were introduced in [1] and techniques to compute iceberg cubes efficiently studied in [4, 10, 20]. Flowcubes differ from this line of research in that our measure is a complex probabilistic model and not just an scalar aggregate such as count or sum, and that our aggregates deal with two interrelated abstraction lattices, one for item dimensions and another for path dimensions. The computation of frequent patterns in large data sets was introduced by [3]; [17] and [9] study the problem of mining multi-level association rules, and [13] deals with mining frequent sequences in a multidimensional space. We borrow ideas from this line of work, such as Apriori pruning and concept hierarchy encoding, but we also differ significantly as we develop techniques that deal with paths, which are not present in their data models, and our algorithms are designed to handle both item and path abstraction lattices.

8. CONCLUSIONS

We introduced the problem of constructing a flowcube for a large collection of paths. The flowcube is data cube 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. We also introduced the flowgraph, a probabilistic workflow model that is used as the cell measure in the flowcube, and that is a concise representation of general flows trends and significant deviations from the trends. Previous work on management of RFID data did not consider the probabilistic workflow view of commodity flows, and did not study how to aggregate such flows in a data cube.

The flowcube is a very useful tool in providing guidance to users in their analysis process. It 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. The flowcube is particularly well suited for the discovery of exceptions in flow trends, as it only stores non-redundant flowgraphs that by definition deviate from their ancestor flowgraphs.

We developed an efficient method to compute the flowcube based on the ideas of shared computation of frequent flow patterns at every level of abstraction of the item and path lattices. Pruning of the search space by taking taking advantage of the relation between the path and the item view on RFID data. Compression of the cube by the removal of infrequent cells, and redundant flowgraphs. And partial materialization of high dimensional flowcubes based on popular cuboids.

Through an empirical study we verify the feasibility of our model and materialization methods. We compared the performance of our proposed algorithm with the performance of two competing algorithms and showed that our solution achieves better performance than those methods under a variety data sizes, data distributions, and minimum support considerations.

9. REFERENCES

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, pages 506–521.

[2] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

[3] 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.

[4] 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)*.

[5] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of state mergin method. In *Second International Colloquium on Grammatical Inference (ICGI'94)*, pages 139–150, 1994.

[6] S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing RFID data. In *Proc. Intl. Conf. on Very Large Databases (VLDB'04)*.

[7] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley and Sons, 2003.

[8] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive RFID data sets. In *International Conference on Data Engineering (ICDE'06)*, 2006.

[9] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 420–431, Zurich, Switzerland, Sept. 1995.

[10] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, pages 1–12.

[11] J. Han, N. Stefanovic, and K. Koperski. Selective materialization: An efficient method for spatial data cube construction. In *Proc. 1998 Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'98) [Lecture Notes in Artificial Intelligence, 1394, Springer Verlag, 1998]*.

[12] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*.

[13] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-dimensional sequential pattern mining. In *Proc. 2001 Int. Conf. Information and Knowledge Management (CIKM'01)*, pages 81–88, Atlanta, GA, Nov. 2001.

[14] S. Sarma, D. L. Brock, and K. Ashton. The networked physical world. 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. White paper, MIT Auto-ID Center, `http://archive.epcglobalinc.org/publishedresearch/MIT-AUTOID-WH-014.pdf`, 2002.

[16] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*.

[17] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 407–419, Zurich, Switzerland, Sept. 1995.

[18] F. Thollard, P. Dupont, and C. dela Higuera. Probabilistic dfa inference using kullback-leibler divergence and minimality. In *Seventeenth International Conference on Machine Learning*, pages 975–982, 2000.

[19] W. van der Aalst and A. Weijters. Process mining: A research agenda. In *Computers in Industry*, pages 231–244, 2004.

[20] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*.

[21] G. K. Zipf. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley: Cambridge, MA, 1949.