Mining Compressed Commodity Workflows From Massive RFID Data Sets *

Hector Gonzalez, Jiawei Han, Xiaolei Li
Department of Computer Science
University of Illinois at Urbana-Champaign
{hagonzal, hanj, xli10}@uiuc.edu

ABSTRACT

Radio Frequency Identification (RFID) technology is fast becoming a prevalent tool in tracking commodities in supply chain management applications. The movement of commodities through the supply chain forms a gigantic workflow that can be mined for the discovery of trends, flow correlations and outlier paths, that in turn can be valuable in understanding and optimizing business processes.

In this paper, we propose a method to construct compressed probabilistic workflows that capture the movement trends and significant exceptions of the overall data sets, but with a size that is substantially smaller than that of the complete RFID workflow. Compression is achieved based on the following observations: (1) *only a relatively small minority of items deviate from the general trend*, (2) *only truly non-redundant deviations, i.e., those that substantially deviate from the previously recorded ones, are interesting*, and (3) *although RFID data is registered at the primitive level, data analysis usually takes place at a higher abstraction level*. Techniques for workflow compression based on non-redundant transition and emission probabilities are derived; and an algorithm for computing approximate path probabilities is developed. Our experiments demonstrate the utility and feasibility of our design, data structure, and algorithms.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Data mining

General Terms: Algorithms

Keywords: RFID, workflow induction

1. INTRODUCTION

Radio Frequency Identification (RFID) technology has received considerable attention from both the hardware and software communities. Hardware research deals with the issues related to building small and reliable tags and readers, while software research addresses the problem of cleaning, summarizing, warehousing and mining RFID data sets. RFID technology is composed of readers and tags. Readers are transponders capable of detecting a tag from a distance and without line of sight. Tags are attached to items, and transmit a unique Electronic Product Code (EPC) when in proximity to a reader. Each tag reading generates a tuple of the form $(EPC, location, time)$, where *location* is the place where the reader is positioned, and *time* is the time when the reading took place.

An RFID data set is a collection of paths, one per distinct tag in the system. Paths can be seen as a probabilistic workflow, where nodes are path stages, and edges are transitions; each edge has a probability that is the fraction of items took the transition. As millions of individual items move through possibly hundreds of locations in the supply chain, such workflows can be enormous.

In [3] we introduced the concept of a FlowCube, which is a data cube computed on an RFID path database. The FlowCube is composed of cuboids aggregated at different levels of abstraction of the path-independent dimensions describing each item (*item view*) such as product, manufacturer, and price; and at different levels of abstraction along the locations and durations of each path stage (*path view*). *Item view* aggregation is similar to regular aggregation in data cubes in which, for example, we may go from individual items, to product type, and to product category. *Path view* is a new kind of aggregation for FlowCubes and it may, for example, collapse all the individual locations within a store to a single "store" location, or all individual trucks into a single "transportation location" to present a summarized view of the paths traversed by items. The measure recorded in each cell of the FlowCube is a probabilistic workflow. [3] presents a framework for efficient construction of FlowCube but does not go into the details of a concrete workflow design. In this paper we address the design and implementation of a *compressed workflow* that can be used as a measure in the FlowCube. Our goal is to design a workflow structure that is manageable in size, but can still be used to identify major flow trends and significant exceptions and provide answers to questions such as:

1. Is there a correlation between the time spent at quality control and the return rate for laptops manufactured in China?

2. What are the most notable characteristics of paths traveled by dairy products that end up being discarded from the stores in Boston?

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

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

3. List containers arriving by ship at New York ports that have spent unusually long periods of time at any intermediate port in Europe?

The gigantic size of an RFID data set and the diversity of queries over flow patterns pose great challenges to traditional workflow induction and analysis technologies since processing may involve retrieval and reasoning over a large number of inter-related tuples through different stages of object movements. Creating a complete workflow that captures the nature of commodity movements and that incorporates time will be prohibitively expensive since there can be billions of different location and time combinations.

In this paper we propose a new probabilistic workflow model that is highly compressed but is still capable of capturing the main characteristics of object movements through locations and time, while keeping track of significant time-location correlations and deviations from the main trend. The workflow is organized in such a way that a wide range of queries can be answered efficiently. Our design is based on the following key observations.

First, it is necessary to eliminate the redundancy present in RFID data and aggregate it to a minimal level of abstraction interesting to users. 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. These tuples can be grouped into a single compressed record: $(EPC, location, time\_in, time\_out)$. For example, if a supermarket has readers on each shelf that scan the items every minute, and items stay on the shelf on average for 1 day, the above compressed record gets a 1,440 to 1 reduction in size without loss of information. One can further compress the data by aggregating the tuples to a level of abstraction higher than the one present in the raw RFID database, for example, representing data at the hour level if users are not interested in infer granularity. Furthermore, one can rewrite the data set as a set of paths of the form $(item_p : (l_1, d_1), \ldots, (l_n, d_n))$ where d_i is the time spent by $item_p$ at the i-th location l_i.

Second, we can do semantic path compression, by merging and collapsing path segments that may be uninteresting to users. For example, a retail store manager may not be interested in item movements through the production lanes in a factory and may like to group all those transitions into a single "factory" location record.

Third, many items have similar flow patterns and only a relatively small number of items significantly deviate from the general trend. Taking this into consideration, one can *construct a highly compressed workflow that represents only the general trends and significant deviations but ignores very low support exceptions*. Such workflow may help uncover correlations present in the data. For example, one may detect that CDs that stayed for more than two months in the factory have a much higher chance of being returned by customers than those that stayed for just one month.

Fourth, for concise representation of the overall flow graphs, it is important to build conditional probabilities on short path segments before long ones. This is because short path segments lead to much less combinations and facilitate concise representation. The exceptions caused by long path segments will need to be expressed only when they are substantially different from that derivable from the corresponding short ones. This will save the overall effort of flow-graph construction and maintenance.

Efficient implementation has been explored with the model proposed above. Our algorithm development and performance study show that such a model, though comprehensive, is clean and concise in comparison with a complete workflow, and the construction of such a model requires only a small number of scans of the RFID data sets.

The rest of the paper is organized as follows. Section 2 presents the structure of the input RFID data. Section 3 presents RFID workflow design and compression methods. Section 4, introduces algorithms for workflow compression. Section 5 reports the experimental and performance results. We discuss the related work in Section 6 and conclude our study in Section 7.

2. COMPRESSED RFID DATA SETS

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. Table 1 is an example of a raw RFID database where a symbol starting with r represents an RFID tag, l a location, and t a time.

RawData
$(r_1, l_1, t_1)\ (r_2, l_1, t_1) \ldots (r_1, l_1, t_2)\ (r_2, l_1, t_2) \ldots (r_1, l_2, t_{10})$ $(r_2, l_1, t_{10}) \ldots (r_2, l_1, t_{15})\ (r_3, l_3, t_{15}) \ldots (r_{10000}, l_3, t_{500})$

Table 1: Raw RFID Records

As shown in our recent study [4], the size of the data can be reduced by eliminating duplicate readings for the same object at the same location and create *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. Table 2 presents a clean RFID database for the raw data in Table 1. The clean database contains stay records for the 10,000 items present in the raw data.

EPC	Stay(EPC, location, time_in, time_out)
r_1	(r_1, l_1, t_1, t_{10})
r_2	$(r_2, l_1, t_{15}, t_{20})(r_2, l_2, t_{21}, t_{29})$
r_3	$(r_3, l_1, t_{10}, t_{15})(r_3, l_3, t_{16}, t_{22})(r_3, l_1, t_{23}, t_{25})$
	...
r_{10000}	$(r_{10000}, l_1, t_{480}, t_{490})(r_{10000}, l_3, t_{491}, t_{500})$

Table 2: A Cleansed RFID Database

It is possible that users are interested in looking at the data at a level of abstraction higher than the one present in the raw data, *e.g.*, users may not be interested in the time spent by items at location at the second level, but at the hour level, and it may not be important to distinguish each item but rather look at the data at the SKU level. We can aggregate the *Stay* table to the level (p_l, l_l, t_l), where p_l is the product level (*e.g.*, SKU, product type, product category), l_l is the location level (*e.g.*, location, locale, location type), and t_l is the time level (*e.g.*, second, minute, hour, day).

For RFID data flow analysis, one could be interested only in the *duration* (*i.e.*, length of time) of object stay or transition, but not the *absolute time*. In such cases, we can rewrite the *Stay* table 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 spends at a location is a continuous attribute. In order to simplify the model even 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*.

Table 3 presents a clustered path-database for the stay data in Table 2. (l_j, i) corresponds to an item that stays for the duration i (where i is a cluster) at location l_j.

Path	*count*
$(l_1, 1)$	700
$(l_1, 2)$	700
$(l_1, 3)$	2400
$(l_1, 1) \rightarrow (l_2, 1)$	240
$(l_1, 2) \rightarrow (l_2, 1)$	240
$(l_1, 3) \rightarrow (l_2, 1)$	800
$(l_1, 3) \rightarrow (l_3, 1)$	2160
$(l_1, 3) \rightarrow (l_3, 2)$	2160
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 1)$	18
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 2)$	18
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 3)$	144
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 1) \rightarrow (l_2, 1)$	6
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 2) \rightarrow (l_2, 1)$	6
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 3) \rightarrow (l_2, 1)$	48
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 1) \rightarrow (l_3, 1)$	36
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 2) \rightarrow (l_3, 1)$	36
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 3) \rightarrow (l_3, 1)$	144
$(l_1, 3) \rightarrow (l_3, 1) \rightarrow (l_1, 3) \rightarrow (l_3, 2)$	144

Table 3: A clustered path-database

3. RFID WORKFLOW

A clustered path-database can be naturally modeled as a probabilistic workflow. Each location corresponds to an activity, and locations are linked according to their order of occurrence. Links between activities q and p have an associated probability that is the percentage of the time that activity p occurred right after activity q. We will illustrate the benefit of such modeling with an example: The US government will require every container arriving at ports in the country to carry an RFID tag in order to detect abnormal paths. We can easily determine the level of abnormality of a path by looking at its transition probabilities in the worflow; you could even look at the current trajectory of a container and predict the set of the most likely next stages, and raise a real time alert when an in-transit container has taken an unexpected transition.

We will state a more formal definition of a probabilistic workflow that is based on the definition of a probabilistic deterministic finite automaton *PDFA* [6]. The only difference is that we associate states with symbols of the alphabet whereas the *PDFA* associates transitions with symbols.

A probabilistic workflow is defined by the tuple $(Q, \Sigma, \eta, \tau, q_0, F)$ where

- Q is a finite set of states
- Σ is the alphabet, in our case it will be locations
- $\eta : Q \rightarrow \Sigma$ is the naming function, that assigns a symbol from Σ to each state
- $\tau : Q \times Q \rightarrow [0, 1]$ is a function that returns the probability associated with a transition
- q_0 is the initial state,
- $F : Q \rightarrow [0, 1]$ is a termination function which returns the probability for a state to be final.

The sum of all the transition probabilities for a state plus its termination probability should add up to 1. Figure 1 presents a probabilistic workflow constructed on all the paths from Table 3. The transition function value for each pair of states $l_i \rightarrow l_j$ is the number placed on the arrow, and it is computed as $count(l_i, l_j)/count(l_i)$, where $count(l_i, l_j)$ is the number of items that took the transition, and $count(l_i)$ is the total number of items that arrived at l_i. The termination function value for each node l_i is the number placed on top of the node and is computed as $count(l_i, \#)/count(l_i)$, where $count(l_i, \#)$ is the number of items that terminate their path at l_i. This workflow is a highly compact summary of the data; we have represented all the paths in the database with a model that has just 6 nodes. But the compression is not lossless, the workflow in Figure 1 does not have any information on the duration spent at each location. For example, we cannot distinguish the paths $(l_1, 1)(l_2, 1)$ and $(l_1, 2)(l_2, 1)$ of Table 3.

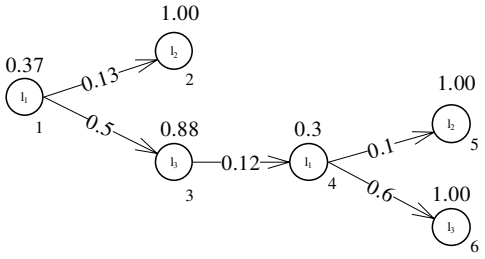

Figure 1: Probabilistic workflow

3.1 Complete workflow

Creating a workflow that incorporates duration information is valuable in the analysis of RFID paths. Duration may for example be very important in inventory management; we can detect path segments that have unusually long durations and use the information to re-route items if possible, or to optimize those segments. In order to incorporate duration into the workflow, the first option is to extend Σ to be not just locations, but the cartesian product of locations $\times$ durations. We refer to a workflow with such nodes as a *complete workflow*. Figure 2 presents the *complete workflow* for Table 3. The problem with this approach is that in many cases we will get a very significant increase in the number of nodes (in our example we went from 6 nodes to 19), with a large percentage of the new nodes being uninteresting. For our running example, we would replace the first node l_1 with 3 nodes $(l_1, 1)$, $(l_1, 2)$, and $(l_1, 3)$, l_2 would have only one node $(l_2, 1)$; if we look at the transition $(l_1, 3) \rightarrow (l_2, 1)$ the transition probability is 0.1 a number close to 0.13 the value recorded in the duration independent workflow, for some applications, explicitly identifying this transition does not add much information. Redundancy becomes more serious when examining all the possible transitions between two nodes with many different possible durations each but with a duration distribution largely independent of each other.

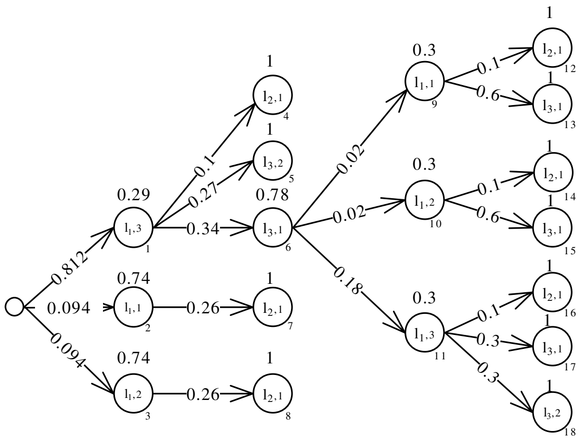

Figure 2: Complete workflow

3.2 Extended probabilistic workflow

In order to solve the inefficiencies presented in the *complete workflow* model we develop an *extended probabilistic workflow* model that incorporates durations by factoring probabilities into conditional transition probabilities, associated with transitions, and conditional emission probabilities associated with durations at nodes. This model is similar to a Hidden Markov Model (HMM) [10] but in our case, we conditioned the emission and transition probabilities at a state on the durations at all previous states, a concept not present in HMMs.

More formally, an *extended probabilistic workflow model* is a tuple $(Q, C, \Sigma, \eta, T, E, q_0)$, where all the components are defined as in a *probabilistic workflow*, and C, T, and E are defined as:

- C, is the set of possible durations at any node, including * to represent any duration.
- $T : \mathcal{P}owerset(Q \times C) \times (Q \bigcup \{\#\}) \rightarrow [0, 1]$, a conditional transition probability function that assigns a probability to each transition going into a state Q or $\#$ (the special termination state symbol) given the time spent at each of the predecessors of Q.
- $E : \mathcal{P}owerset(Q \times C) \times (Q \times C) \rightarrow [0, 1]$, a conditional emission probability function that assigns a probability to each length of stay $c \in C$ at node Q given the time spend at each of the predecessors of Q.

As before, we require that the emission and transition probabilities conditioned on a given path prefix add up to 1.

The ability to compute the probability of a path or a path segment is important, it can be used for example for data cleaning, we can use the workflow to detect unlikely transitions due to false positive readings, or to check the most likely next locations where an item can go right after being at a given location, in order to detect if the reader missed the item or if it really moved. The workflow can also be used to rank paths according to likelihood and allow data analysts to focus their effort in optimizing the top-k most likely paths.

Before we present how to compute probabilities in an *extended probabilistic workflow* we need a few definitions:

- We define a path x to be the sequence of stages x_1 x_2 $\ldots$ x_l, where each x_i a location/duration pair of the form (q, c); $q \in Q$ and $c \in C$. When $c = *$ when we do not care about the particular duration at the location.
- The length of a path x, $length(x)$ is the number of stages in the path.
- We can identify the state and duration of components of the stage i in path x by the functions $state(x, i)$ and $time(x, i)$ respectively.
- We define the prefix of a path $prefix(x, i)$ to be the first i stages in path x. $prefix(x, 0) = \phi$. The prefix of a path, is a path itself. Every path is a prefix of itself.
- The function $count(x)$ returns the number of times that the path x appears in the clustered path database as a complete path or as a prefix of a longer path x'. Note that $(q, c = *)$ matches any duration at location q.
- A path xx' is the concatenation of the stages of the paths x and x'. A path $x(q, c)$ is the concatenation of the stages of path x with the stage (q, c).

We can define the probability of a path as the product of all the conditional emission probabilities for the durations spent at each location in the path, multiplied by the product of all the conditional transition probabilities in the path.

$$P(x) = Emission(x) \times Transition(x) \quad (1)$$

$$Emission(x) = \prod_{i=1}^{n} E(prefix(x, i-1), x_i) \quad (2)$$

$$Transition(x) = \left(\prod_{i=1}^{n} T(prefix(x, i-1), state(x_i)) \right) \times T(x, \#) \quad (3)$$

The computation of an emission probability for c_i $(c_i \neq *)$ in the stage $x_i = (q_i, c_i)$ given the prefix $x_1 \ldots x_{i-1}$ is done as follows:

$$E(prefix(x, i-1), x_i) = \frac{count(prefix(x, i))}{count(prefix(x, i-1)(state(x_i), *))} \quad (4)$$

The computation of a transition probability to location l_i given the prefix $x_1 \ldots x_{i-1}$ is done as follows:

$$T(prefix(x, i-1), l_i)) = \frac{count(prefix(x, i-1)(l_i, *))}{count(prefix(x, i-1))} \quad (5)$$

Example (Emission probability computation). Figure 3 presents each node in the workflow with the conditional emission and transition probabilities. Rows represent durations at the node, and columns the probability of the duration conditioned on different path segments. If we look at the conditional emission table for node 6, we can compute cell 1 (row 1, column 1) as $E((l_1, *)(l_3, *)(l_1, *), (l_3, 1)) = count((l_1, *)(l_3, *)(l_1, *)(l_3, 1))/count(\ (l_1, *)(l_3, *)(l_1, *)(l_3, *))$ $= 216/360 = 0.6$. Similarly we can compute transition probabilities.

3.2.1 Uninteresting conditional probabilities

When computing the conditional transition and emission probabilities for a node, we do not need to consider every possible path that is a prefix to the node, as a conditioning factor. In certain supply chain applications it is possible for some items to have unique path segments, *e.g.*, items moving in a conveyor belt may all have identical paths and durations at each stage. When finding the conditional emission or transition probabilities we can consider each unique path segment as an atomic unit, this offers great benefits in computation costs and workflow compression, *e.g.*, if every item of a certain type goes through the same 20 stages in a conveyor belt, by considering that path segment as a unit we gain a 2^{20} to 1 reduction in the number of possible path prefixes to consider.

We call a conditional emission or transition entry *uninteresting* if it can be derived from another entry without loss of information. For example, if we know that before a certain product gets to the shelf, it always spends 1 day in the warehouse, 2 days in the store backroom, and 4 hours in the truck, there is no need to condition the length of stay at the shelf on all eight combinations of these three stages, conditioning on the single prefix containing all stages is sufficient.

We say that a path x is closed iif you can not find another path y that goes thorough the same stages as x and that is more specific (has less durations set to *) and $count(x) = count(y)$. This is a concept very similar to that of closed frequent patterns [9]. An emission or transition probability entry conditioned on x is *uninteresting* if x is not closed.

Lemma 1. Conditional emission probabilities conditioned on non-closed paths can be derived without loss of information from conditional emission probabilities conditioned on closed paths.

Proof Sketch: Assume that we are looking at node q_i of a workflow and computing the emission probability of duration c_i. Assume that x is a non-closed path leading to q_i and that y is the closed path corresponding to x. If we know $E(y, (q_i, c_i))$, given that $count(x) = count(y)$ we get that $E(x, (q_i, c_i)) = E(y, (q_i, c_i))$. ∎

Lemma 2. Conditional transition probabilities conditioned on non-closed paths can be derived without loss of information from conditional emission probabilities conditioned on closed path.

Proof Sketch: Using an analogous argument to that used in the proof of lemma 1, the transition probability $T(x, q_i) = T(y, q_i)$ when x is not closed and y is the corresponding closed path. ∎

Example (Uninteresting conditional probabilities). Looking up node 3 in the workflow of Figure 3 and computing the conditional transition probability to node 4 given $(l_3, 1)$, we get $T((l_1, *)(l_3, 1), l_1) = 600/2760$. This is exactly the same as $T((l_1, 3)(l_3, 1), l_1) = 600/2760$. The reason is that $(l_1, *)(l_3, 1)$ is not a closed path. Thus the first conditional probability is uninteresting.

3.2.2 Redundant conditional probabilities

The benefit of all conditional emission and transition probabilities is not the same. Intuitively, a conditional probability that deviates significantly from the probability that we would infer from the currently recorded conditional probabilities is more important than the one that deviates very little. Such deviations are important because they are interesting patterns by themselves, *e.g.*, it may be important to discover that the transition, for perishable products, to the return counter increases when they spent too long in transportation locations. Recording these deviations is also important to compute more accurate path probabilities.

When determining if a conditional probability is redundant at a node it is important to *condition on short path prefixes before we condition on long path prefixes*, *i.e.*, we only record deviations given prefixes of length i if they are not redundant given recorded deviations on prefixes of length $i-1$. This optimization is quite natural and effective as it will uncover substantially fewer truly "surprising" deviations, that are really interesting.

Let us define $\hat{E}(x, (q, c))$ to be the expected emission probability of duration c at node q given path x. If we have already computed the emission probabilities $E(a_1, (q, c))$, $\ldots$, $E(a_n, (q, c))$, where each a_i is a direct ancestor of x (same path stages, and just one less duration set to *) and $(a_i, (q, c))$ and $(a_j, (q, c))$ are independent and conditionally independent given (q, c) for every $i \neq j$, we can compute $\hat{E}(x, (q, c))$ as,

$$\hat{E}(x, (q, c)) = \left(\prod_{i=1}^{n} \frac{E(a_i, (q, c))}{E(b, (q, c))} \right) \times E(b, (q, c)) \qquad (6)$$

where $E(b, (q, c))$ is the unconditional probability of observing duration c at state q, b is the path up to q with all durations set to $*$. Intuitively, this equation is adjusting the unconditional probability of observing duration c in stage q, by compounding the effect that each ancestor independently has on the emission probability[1].

We say that a conditional emission probability is *redundant* if $|\hat{E}(x, (q, c)) - E(x, (q, c))| < \epsilon$. And we call ϵ the *minimum probability deviation threshold*[2].

Similarly, we can define *expected transition probability* of $T(x, q)$ as:

$$\hat{T}(x, q) = \left(\prod_{i=1}^{n} \frac{T(a_i, q)}{T(b, q)} \right) \times T(b, q) \qquad (7)$$

where each a_i a direct ancestor of x and $T(b, q)$ is the unconditional transition probability to state q. The intuition behind Eq. (7) is the same as for Eq. (6).

We say that a conditional transition probability is *redundant* if $|\hat{T}(x, q) - T(x, q)| < \epsilon$.

3.2.3 Low support conditional probabilities

Up to this point we have considered all the closed paths that are a prefix of a node in computing its conditional emission and transition probabilities. Conditioning probabilities on every closed path may lead to very large probability tables, with many entries supported by very few observations (paths). The size of our workflow may end up being dominated by noise in the clustered path-database. In order to solve this problem, we restrict our closed paths to only those

[1] Eq. (6) can be better understood with a simple example, assume that you have three events a, b, and c, and b, c independent and conditionally independent given a, you can compute $P(a|bc) = P(a) \times P(a|b)/P(a) \times P(a|c)/P(a)$, which is equivalent to our formula when you have only two ancestors.

[2] Alternative definitions are possible. For example, we could define an emission probability to be redundant if: $max_{a_i}(|E(x, (q, c)) - E(a_i, (q, c))|) < \epsilon$.

that appear in the clustered database more than δ percent of the time. This has the effect that any probability entry will be supported by at least δ percent of the observations and the size of the tables will be significantly smaller. We call δ the minimum support.

3.2.4 Compressed Workflow

We call an *extended probabilistic workflow*, with only interesting, and non-redundant conditional emission and transition probability entries, conditioned on closed paths with minimum support δ and minimum probability deviation ϵ a *compressed workflow*.

The optimal definition of δ and ϵ, such that a good compromise between workflow size, exception interestingness, and information loss is achieved is a very challenging task, and one that depends on the specific application needs. When the main objective is to detect general flow patterns and significant exceptions, the use of larger values of δ and ϵ has two advantages: discovered exception will likely be more interesting, and the total number of exceptions to analyze will be smaller. When the objective is to use the *compressed workflow* to compute path probabilities with high accuracy, we need to be more conservative in the selection of the parameters. For this case a good indicator for parameter selection may be the nature of the paths for which we will estimate probabilities. ϵ can be estimated by checking the error on a representative query set with varying levels of the parameter, we can use the value where error rate has a big jump (see Figure 7). δ could be selected using a top-k frequent path approach such as [5].

Figure 3 shows the *compressed workflow* derived from the clustered path database in Table 3. The number of nodes is 6, the same as the duration independent workflow in Figure 1. Each node contains a conditional emission table, and a conditional transition table. The lightly shaded columns (nodes 1 and 3) are redundant for an $\epsilon = 0.2$ and the darkly shaded columns (node 6) are not supported for a $\delta = 150$ paths. An estimate for the size of the workflow is the number of probability entries, in this case, 30 entries [3].

4. WORKFLOW COMPUTATION METHOD

In this section we will introduce an algorithm to compute a *compressed workflow* from a clustered path database, with minimum support δ and minimum probability deviation ϵ.

4.1 Mining closed paths

From the discussion in the previous section we know that we only need to condition transition and emission probabilities on frequent closed paths. The mining of closed paths can be easily implemented with an efficient closed itemset mining algorithm [16] after associating locations to nodes in a duration independent workflow . If we are computing not a single cell but a complete FlowCube we can compute all the frequent closed paths for all cells at all levels of abstraction in parallel using the method presented in [3].

After we have computed the counts of all closed paths, we need to do one more pass over the path database computing the support of each closed path followed by each duration independent suffix. For example, if we get that $(l_1, 3)(l_3, 1)$ is a frequent closed path, we should compute $(l_1, 3)(l_3, 1)(l_1, *)$, $(l_1, 3)(l_3, 1)(l_2, *)$, and $(l_1, 3)(l_3, 1)(l_3, *)$. These counts will be needed in computing conditional emission and transition probabilities.

[3]In reality the number of probabilities is smaller as we only need to store $n - 1$ entries for each column given the law of total probability.

4.2 Mining conditional probabilities

When computing conditional probabilities given closed frequent paths we follow a short to long principle. We first record unconditional probabilities, then we record probabilities conditioned on paths of length 1 and that deviate from the unconditional probabilities; when we have all paths of length 1, we look at the ones of length 2, and record deviations given the unconditional entries and the length 1 entries; we repeat this process until we have looked at all closed frequent path prefixes for a node. This method is very efficient as the number of distinct short paths is quite small as compared to large paths, and in real RFID applications conditional probabilities usually depend on short path segments, *e.g.*, the transition to the location return counter may depend on the time that an item stayed at quality control, but not on a very long combination of locations and durations.

4.2.1 Computing non-redundant conditional emission probabilities

At this point we have all the information required to compute the conditional emission probabilities given by Eq. (4) for all nodes in the workflow.

When computing $E(prefix(x, i-1), x_i)$ we require the entire path x to be frequent and not only the $prefix(x, i-1)$ part to be frequent. This makes sense as we want to compute deviations in emission probability when enough supporting evidence (including the emission duration itself) is available.

We need to determine which conditional probabilities are redundant and that can be easily done by arranging the closed paths that are a prefix of each node in a lattice, and traversing the lattice from more general paths to more specific paths, storing only probabilities that are found to be non-redundant using Eq. (6).

4.2.2 Computing non-redundant conditional transition probabilities

We can use Eq. (5) to compute the conditional transition probabilities for all nodes in the workflow.

When computing $T(prefix(x, i-1), q_i)$ we require $prefix(x, i-1)$ $(q_i, *)$ to be frequent. This guarantees that we have enough evidence of the conditional transition to q_i.

The determination of redundant conditional transition probabilities follows exactly the same logic used for emission probabilities in the previous section.

4.3 Algorithm

Based on the previous discussion we state algorithm 1, which summarizes the complete set of steps required to output the *compressed workflow*.

Analysis. Algorithm 1 can be executed quite efficiently, in addition to the scans required by the frequent closed itemset mining algorithm, we need just two more scans, one to build the duration independent workflow, and another to collect extra counts needed to compute conditional probability entries.

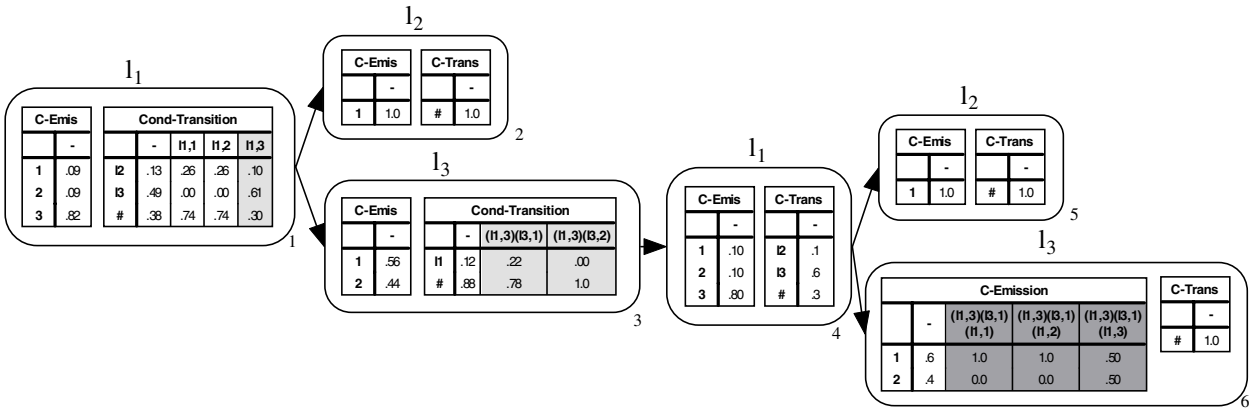

Figure 3: Compressed Workflow

Algorithm 1 Construct *compressed workflow*

Input: Clustered path database D, minimum conditional probability deviation ϵ, minimum support δ
Output: *compressed workflow* W
Method:

1: On a single pass over D construct the duration independent workflow W_1.
2: Using W_1 compute D_1 the path database with locations encoded with nodes in W_1.
3: Call a frequent closed itemset mining algorithm with D_1 as the transaction database and minimum support δ to derive C the set of closed frequent paths.
4: Scan D once to collect the support of the extended frequent closed paths.
5: Construct W by annotating each node in W_1 with the set of non-redundant conditional emission and transition probabilities given the closed paths in C, and the minimum conditional probability deviation ϵ.
6: return W.

4.4 Computing the probability of a path in the compressed workflow

Eq. (1) can be used to compute the probability of a path x when we have all possible conditional emission and transition probabilities. But when we have compressed the workflow by using ϵ as a threshold for redundant probabilities, and δ as the minimum support for recording a conditional probability, we can not use this equation anymore, because many of the required probabilities may be missing.

The solution is to define an approximation of $P(x)$ called $\hat{P}(x)$, that uses Eqs. (6) and (7) for computing conditional emission and transition probabilities respectively, whenever the exact probabilities have not been explicitly recorded in the compressed model. The path prefixes to use are the most specific paths recorded in the model that are still more general than x.

The probability for path x computed by $\hat{P}(x)$ is an approximation to $P(x)$ and is subject to error. The amount of error will increase for larger values of ϵ and δ, and will also depend on the support of x. If $P(x)$, that is the true probability of x is very low, then the error can be larger as the computation of the probability of x may require conditional probability entries that were not stored in the *compressed workflow* because of not having enough support. In the experimental section we explore the conditions under which error increases in more detail.

5. PERFORMANCE STUDY

In this section, we perform a thorough analysis of our model and algorithms. 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 Linux kernel 2.6.13.4 and gcc 4.0.2.

5.1 Data Synthesis

The RFID databases in our experiments were generated using a random path generator that simulates the movement of items through a set of locations in a manner that mimics the operation of a large retailer. Locations have a multinomial transition probability distribution for the likelihood of moving to a given next location or stopping, a multinomial duration probability distribution for the length of stay of items at the location, and a list of correlations with other locations. We also associate a preferred level to locations, to indicate the stage in which they tend to appear. For a given location, transition probabilities are high for locations at the next level and low for locations at any other level, this generates the effect that items tend to move forward in the supply chain but that sometimes can still go backwards or jump to a future location.

As notational convenience, we use the following symbols to denote certain data set parameters. $\mathcal{N}$: Number of paths. ϵ: conditional probability threshold. δ: minimum conditional probability support.

5.2 Workflow Compression

One of the main advantages of using a *compressed workflow* with minimum support δ and minimum conditional probability deviation ϵ is that it captures the essence of the *complete workflow* but requires only a fraction of the space. In this section we compare the size of a *compressed workflow* with that one of a *complete workflow*. The *complete workflow* is constructed by traversing the clustered path database adding each path to a probabilistic prefix tree, while keeping track of the counts of items reaching each node and the transition counts. We will measure the size $\mathcal{S}$ of a workflow

by the total number of probability entries. The size of a *compressed workflow* is the sum of the number of probabilities for the conditional emission, and conditional transition tables for each node. The size of a *complete workflow* is just the sum of the number of transition probabilities for each node.

Figure 4 shows the size of the *compressed workflow* for various levels of ϵ, for a data set containing 100,000 paths, and using a support δ of 0.1%. In the Figure we distinguish the count of conditional and unconditional transition and emission probabilities. For this experiment the size of the *complete workflow* is of 14,095 probability entries. As we decrease ϵ the number of non-redundant conditional transition and emission probabilities increases. For $\epsilon = 0.2$ we do not record any conditional probabilities and the size of the workflow is just the unconditional emission and transition probabilities, as we decrease ϵ the percentage of the total workflow size accounted by conditional probabilities increases from 0% to around 80%. The size of the *compressed workflow* is just 1.5% to 7.5% of the size of the *complete workflow*, we observe very significant space savings even for small values of ϵ.

Compression power is even more dramatic when we increase the number of distinct paths, for this example we generated another data set containing every possible distinct path with a count proportional to its likelihood given the location characteristics. The size of the *complete workflow* increased to 1,190,479 nodes, while the size of the *compressed workflow* increased only marginally. Compression power for this case increased to 10,000 to 1.

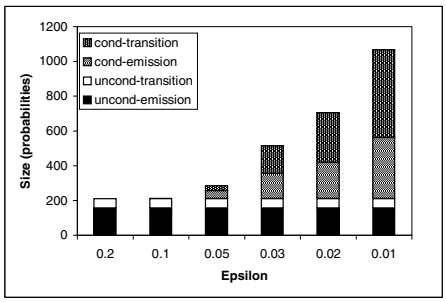

Figure 4: Compression vs. Conditional probability threshold (ϵ). $\mathcal{N} = 100,000$, $\delta = 0.001$

Figure 5 shows the size of the *compressed workflow* for the same data set used for Figure 4; with a fixed ϵ at 5% and with varying support levels. The support parameter determines the number of frequent closed paths that need to be considered in constructing the conditional probability tables of each node, lower support levels tend to create more conditional probability entries. For this experiment we used support levels of 1% to 0.001%; as we decrease support the percentage of the total workflow size corresponding to conditional probabilities increases from 0% to around 95%. As in the previous case compression is very significant, the size of the *compressed workflow* is just 1.5% to 4.5% of the size of the *complete workflow*.

Figure 6 shows the size of the *compressed workflow* and the *complete workflow* for different sizes of the input clustered path database. We can see that while the size of the *compressed workflow* remains largely constant the size of the

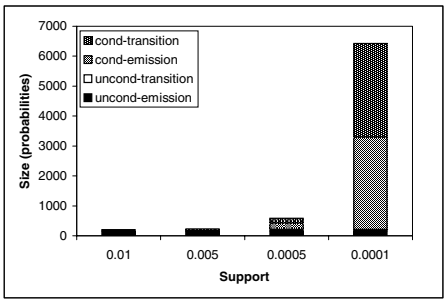

Figure 5: Compression vs. Support (δ). $\mathcal{N} = 100,000$, $\epsilon = 0.05$

complete workflow increases significantly with the number of paths. The reason is that in the *compressed workflow* we only need to update conditional emission and transition probability entries, while in the *complete workflow* new nodes need to be added for each new location/duration observed. In this experiment the error induced by compression remained constant for all database sizes at around 9%.

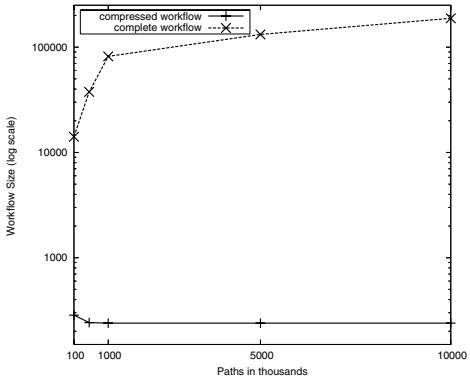

Figure 6: Clustered Path DB size vs. Workflow Size. $\epsilon = 0.05$, $\delta = 0.001$

5.3 Workflow Compression Error

In this section we look at the error incurred in computing the probability of a path using the *compressed workflow* as compared to the *complete workflow*. A *compressed workflow* with probability deviation ϵ and minimum support δ may not be able to assign the exact probability to every possible path, because it only records emission and transition probability deviations if they are supported by more than δ transactions and the deviation is larger than ϵ of the expected probability. The *compressed workflow* will assign probabilities that may be different from the true probability to uncommon paths that deviate from their expected probabilities (in absolute terms the deviation will still be small as the path will have very low probability to begin with, but in percentual terms it can be large). On the other hand, more common paths, or path segments, will be assigned very precise probabilities, because their deviations will have enough supporting transactions. The experiments on this section correspond to the same data sets used to compute *compressed workflow* size in the previous section.

For the experiments in this section we will use the *percentual error* measure[4], which computes the percentual deviation in probability from the *complete workflow*, and is defined as,

$$\frac{|P(path|compressed\_wf) - P(path|complete\_wf)|}{P(path|complete\_wf)}$$

Figure 7 shows the percentual error that a *compressed workflow* will assign to three sets of paths with varying levels of ϵ. The sets of paths used for testing have a probability of occurring (minimum probability) of 0.0001, 0.0002, and 0.0005 respectively. We can see that as we increase ϵ the error increases, and it increases faster for uncommon paths than for more common ones. In this case the common paths have an error of around 9% with every level of ϵ while the other two sets go from 13% to 17% and 16% to 27% respectively. This means that even with a low ϵ we can compute the probability of more common paths and path segments with low error and storing only a small *compressed workflow*.

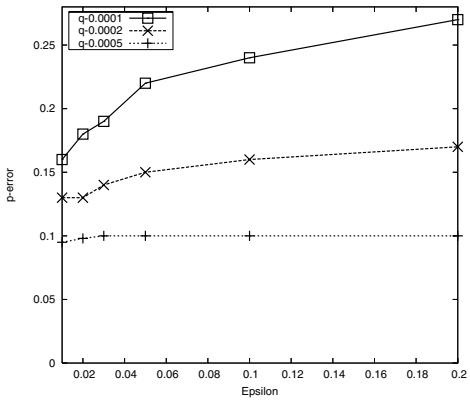

Figure 7: Percentual Error vs. Conditional probability threshold (ϵ). $\mathcal{N} = 100,000$, $\delta = 0.001$

Figure 8 shows the percentual error that a *compressed workflow* will assign to the same three sets of paths as before but with varying the support threshold δ, used to compute the closed paths necessary for recording conditional probability entries. We can see that as we increase support, the percentual error grows; faster for the path data sets with low probability. Common paths have an error of around 10% with every level of δ while the other two sets go from 12% to 17% and 14% to 27% respectively. In this case we can also observe that using a small support threshold is enough to compute the probabilities of common paths, and path segments, with low error. As discussed in section 3.2.4 we can use the slope of the error curves to determine good values for ϵ for a particular type of query load.

5.4 Construction Time

In this section we analyze the efficiency of the algorithm used to construct the *compressed workflow*. The main components of the algorithm are: (1) computation of the unconditional emission and transition probabilities, (2) computation of the closed paths for a given support δ, and (3) computation of the conditional emission and transition entries given the frequent closed paths, and a conditional probability threshold ϵ. In our experiments we observed that the factor with highest influence in computation time was the support threshold, given that it determines how many closed paths will be found, and we need to test each node for conditional dependencies against each of the closed paths.

[4]We also use KL-Divergence of the computed probabilities and the true probabilities and this measure mimics percentual error.

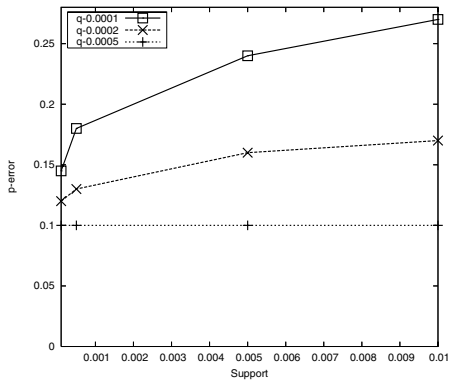

Figure 8: Error vs. Support (δ). $\mathcal{N} = 100,000$, $\epsilon = 0.05$

Figure 9 shows the computation of the conditional emission and transition probabilities for a clustered path database with 1,000,000 paths. We can see that as we increase support, the computation time decreases. For this experiment the time to learn the unconditional probability entries of the workflow was unaffected by the support threshold and remained constant at around 4 seconds. For mining closed patterns we used the program closet+ [16], and the run time was around 2 seconds for each support level.

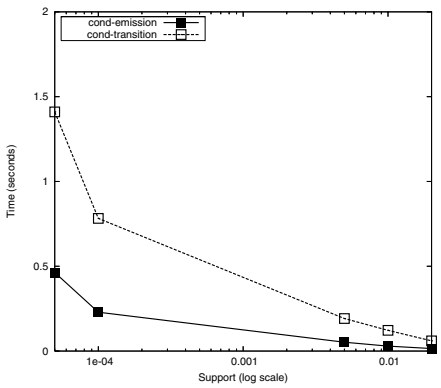

Figure 9: Construction Time vs Support (δ). $\mathcal{N} = 1,000,000$, $\epsilon = 0.05$

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 [11, 12]. The second is cleaning of errors present in RFID data due to error inaccuracies, [7, 8]. The third is related

to the creation of a multi-dimensional warehouses capable of providing OLAP operations over large RFID data sets [4, 3]. Our work is an extension of [3]. That paper introduces the concept of a FlowCube which is a data cube constructed on a path database where each cell records as its measure a workflow, which the authors call FlowGraph. In this paper we explore in detail the design of a concrete probabilistic workflow that can be used in the FlowCube.

Workflow induction is an area or research close to our work [15], it studies the problem of learning the structure of a workflow from even 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, and does not take activity duration into account. [14] deals with time in workflow learning; it first learns a workflow without time, and then replays the log, computing statistics such as average, minimum, and standard deviation for each node. This approach is not well suited for RFID flows, where there may be strong interdependencies between the time distributions at different nodes not captured by summary statistics at a single node.

Another area of research closed to RFID workflow mining is that of grammar induction [2, 13], the idea is to take as input a set of strings and infer the probabilistic deterministic finite state automaton (PDFA) [6] that generated the strings. This approach could be applied to a clustered path database, by looking at each path as a string formed by an alphabet composed of every possible combination of location and duration; but as we have shown in the experimental section this approach is not well suited for RFID data as it tends to generate huge workflows.

7. CONCLUSIONS

We have proposed a novel model for the design and construction of a highly compressed RFID workflow that captures the main trends and important deviations in item movements in an RFID application. The *compressed workflow* records for each node, conditional emission and transition probability tables that have an entry for each non-redundant, and sufficiently supported deviation from the main movement trend. This design takes advantage of the fact that in many RFID applications items tend to move according to a general trend, and have a relatively small number of deviations from it. Our performance study shows that the size of the *compressed workflow* is only a fraction of that of a *complete workflow*, that it can be used to compute the probabilities of paths and path segments with high precision, and that it can be computed efficiently.

The *compressed workflow* presented in our study can be a very useful in providing guidance to users in their analysis process as it highlights general trends and exceptions which may not be apparent from the raw data. This knowledge can be used to better understand the way objects move through the supply chain, and optimize the logistics process governing the most common flow trends; while exception flow information can be used to uncover events that may for example trigger a higher than expected return rate.

Notice that our proposal for workflow compression is based on the assumption that commodities flow according to a general trend and only few and significant deviations are present in the data. 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 workflow models for such applications.

8. 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. 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.

[3] 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.

[4] 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.

[5] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 211–218, Maebashi, Japan, Dec. 2002.

[6] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, Reading, Massachusetts, 1979.

[7] 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.

[8] 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.

[9] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, Jan. 1999.

[10] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77:257–286, 1989.

[11] 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.

[12] 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.

[13] 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.

[14] W. van der Aalst and B. van Dongen. Discovering workflow performance models from timed logs. In *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, pages 45–63, 2002.

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

[16] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 236–245, Washington, DC, Aug. 2003.