

# Real-time Acquisition of Buyer Behaviour Data – The Smart Shop Floor Scenario

Bo Yuan, Maria Orłowska, Shazia Sadiq

School of Information Technology and Electrical Engineering  
The University of Queensland, St Lucia QLD 4072  
Brisbane, Australia  
{boyuan, maria, shazia}@itee.uq.edu.au

**Abstract.** The emergence of a range of new technologies like auto-identification devices, active tags, and smart items has impacted profoundly on business software solutions such as supply chain management, logistics, and inventory control. Integration of automatic data acquisition with enterprise applications as well as potential to provide real-time analytic functionality is opening new avenues for business process automation. In this paper, we propose a novel application of these technologies in a retailing environment leading to the vision of a *smart shop floor*. We firstly present the infrastructure for the smart shop floor. We then demonstrate how the proposed infrastructure is feasible and conducive to real-time acquisition of buyer behaviour data through three selected queries. Complete algorithmic solutions to each query are presented to provide proof of concept, and further deliberations on analytic potential of the proposed infrastructure are also provided.

**Topic:** Data Capture in Real-time      **Category:** Regular Paper

## 1 Introduction

It is evident from mandates from retailing giants such as Wal-Mart and Metro, manufacturers like Proctor & Gamble, and several initiatives within public sector, that technologies built on smart identification devices and wireless communication protocols will, and have already, become an inherent aspect of business operations. These technologies are providing unprecedented potential for business process automation, changing profoundly the expectations of data scale and quality and system responsiveness.

The notion of *smart items* has been proposed [1] to provide an umbrella concept to several approaches in this regard. Essentially a smart item is a device that can provide data about itself and/or the object it is associated with, and in addition has the ability to communicate this data. Examples include RFID tags [2], and sensor devices.

The deployment environment for smart item technologies clearly requires a number of additional components such as device readers, communication networks, and back-end servers. Many challenges relating to reader accuracy, noise management, variety of tag data standards (e.g. GTIN – Global Trade Identification Number, SSCC- Serial

Shipping Container Code, GRAI – Global Returnable Asset Identifier and more recently EPCGlobal [4]) impact on the integration of smart item generated data with enterprise applications.

Business solution providers have made tremendous progress in the last few years to integrate smart item technologies within enterprise application frameworks, see for example SAP Auto-ID Infrastructure [3]. Most of these developments have focused on streamlining the processes that involve packing, receiving, distribution of goods, and related services on stock management and logistics.

In this paper, we propose to extend these developments into a retailing environment, wherein a smart item infrastructure is combined with a *smart shop floor*. The smart shop floor provide additional components that go beyond product/item tagging, and introduces buyer profiles and behaviour characterizations into the environment.

In the next section we will present our vision for the smart shop floor. We envisage that the proposed environment holds the potential to generate unprecedented insights into buyer behaviour in large retail markets. These insights, made possible through the proposed infrastructure, can be supported through fully automated functions. In the remaining paper, we will provide as proof of concept three possible queries (insights) that can be provided using the proposed infrastructure. Complete algorithmic solutions are provided for each of the queries. We will conclude the paper by providing further deliberations on analytic potential of the proposed setup thereby identifying a number of interesting extensions to this work.

## 2 Smart Shop Floor

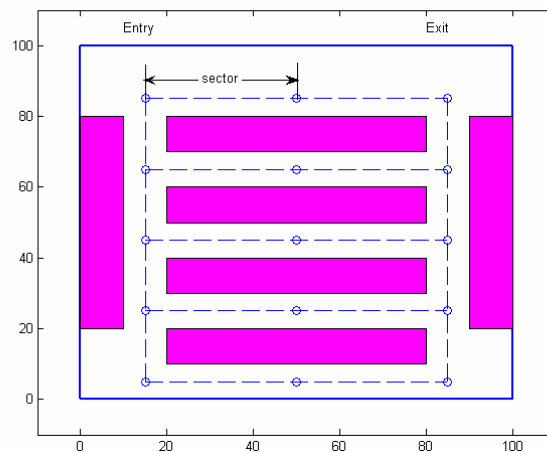
The motivation of smart shop floor is to increase the quality of service in retailing environments by providing shoppers personalized, valuable shopping advices and addressing common issues such as confusing price labels, long queue/waiting time at check-outs as well as the frustration of locating products. The successful deployment of the smart short floors is expected to not only offer retailers crucial advantages over competitors but also make the shopping experience more enjoyable.

The technical components in the infrastructure of such smart shop floors have been made available, as demonstrated by some prototypes currently under trial such as the U-Scan smart trolley of Fujitsu and the Metro Group Future Store Initiative ([www.future-store.org](http://www.future-store.org)). The key components related to the queries in this paper are:

- Active/Smart Trolley
  - o Capture buyer profile (e.g., swipe card).
  - o Read product item tags (i.e., it can recognize items being put in and maintain an ongoing shopping record).
  - o Receive location data from sector indicators.
  
- Sector Indicators
  - o Sense the existence of a trolley and send it the unique sector identification. An example is given in Figure 1 showing a regular shop floor.

In practice, they could be arranged in a much more flexible manner, basically reflecting the floor's layout of the shop. The precise definition of a sector will be introduced in Section 3.1.

- Secure Wireless Network
  - o Provide the communication between smart trolleys, sector indicators and back-end servers to collect real-time path and shopping data.



**Fig. 1.** A high-level illustration of a smart shop floor set-up with 18 sectors. It shows the entry/exit, product shelves and the segmentation of the shop floor in terms of sectors.

### 3 Characterizing Buyer Behaviour

Understanding buyer behaviour is one of the most complex endeavours in business development and marketing. Indeed it is a question that goes well beyond what computational techniques can offer. In the context of this discussion, we limit the notion of buyer behaviour to *shopping profile* within a well defined shop floor space and within a given shopping session. The shopping profile is characterized by three aspects:

1. The Buyer – Buyer properties such as age, gender, postcode, credit limit etc.
2. The Products – Product data and purchase list within a given session.
3. The Navigation – Navigation path in relation to defined sectors in the shop floor space in a given session.

Based on the smart shop floor infrastructure proposed in the previous section, the data relating to all aspects of the shopping profile can be acquired in real-time. Subsequently, there are a number of queries that can be conducted to characterize various

aspects of buyer behaviour through the shopping profiles. In this section, we present three typical queries and the corresponding solutions as example and proof of concept for the proposed infrastructure. These are:

- **Query 1.** To discover the path of a given length (defined by the number of sectors) shared by the largest portion of buyers.
- **Query 2.** To find out the path with as many sectors as possible, subject to a pre-defined threshold of support.
- **Query 3.** To find out sectors where buyers visit frequently but seldom purchase any products in these sectors.

### 3.1 Shop Floor Specification

The basic element in the proposed three queries is called “sector”. Note that in most realistic shop floors, product shelves are naturally divided into sectors according to the types of products. The reasons of conducting queries based on sectors instead of precise physical locations are listed as follows:

- o The data of physical locations do not have intuitive meaning and can become totally meaningless if the layout of the shop floor is changed.
- o The data of physical locations need more storage space and usually require the process of data cleaning.

In the proposed shop floor described in Section 2, sector indicators are deployed along the product shelves. Each time a smart trolley passes by a sector, it will communicate with the corresponding sector indicator, which will send back its unique identification to the trolley. An ongoing list of sectors visited by a buyer is maintained in the trolley, which can be transferred to back-end servers in real-time via the secure wireless network in the shop. In general, sector indicators can be used to capture the general information of shopping paths, by processing only a limited amount of data. Certainly, the level of precision of the path information can be easily controlled by varying the number of sector indicators as well as their specific locations.

For the purpose of algorithmic analysis, let  $S=\{s_1, s_2, \dots, s_n\}$  be a set of literals representing sectors. Define  $G$  as an undirected graph on the sectors (i.e., each vertex represents a unique sector). An edge in  $G$  connecting  $s_i$  and  $s_j$  indicates the neighbourhood of the two sectors (i.e., a buyer can move directly from one sector to another). A *path* is an ordered list of sectors (i.e., minimum one sector), which must satisfy the relationship among sectors as defined in  $G$  (see Figure 2 for an example). In other words, a path is created by starting from a certain vertex in  $G$  and travelling through the graph following the edges. Here, we use the most flexible definition of paths and impose no restrictions on the possible forms of paths, which means that paths can be of different lengths and a certain sector can occur multiple times in a path.

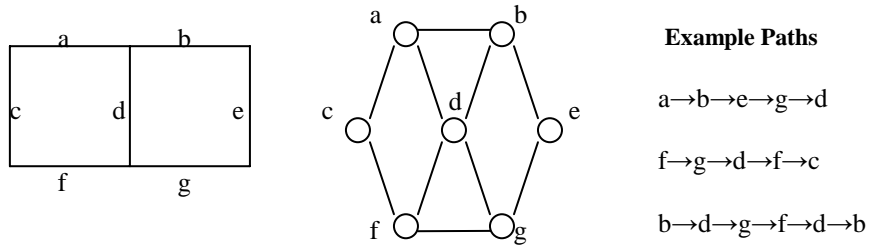


Fig.2. A shop floor (left), its graphical notation (middle) and some example paths (right).

### 3.2 Finding the Most Popular Common Path with a Given Length

The objective of this query is to discover the path of a given length (defined by the number of sectors) shared by the largest portion of buyers.

#### 3.2.1 Algorithm Framework

Given a database  $D$  of paths defined as above (i.e., each record is created by a buyer's single visit to the shop), the question is how to find out the most frequent path shared by buyers. Here, we assume that the length of the path is given in advance. In classical data mining algorithms for shopping basket problems such as algorithm *Apriori* [5] and its extensions and variations [6, 7], all candidates have to be enumerated in advance. During the scan of the database, there is a need to sequentially check which candidates are supported by each record. Suppose that there are totally  $P$  sectors in the shop floor with each sector directly connected to  $K$  other sectors. It is easy to see that the number of all possible candidate paths with  $N$  sectors is  $P \cdot K^{N-1}$ , which is usually quite large given realistic values of  $N$ ,  $K$  and  $P$ . For example, in the shop floor shown in Figure 1, there are 18 sectors each of which is connected to 2 to 4 other sectors. In this case, there are hundreds of thousands candidate paths for  $N=10$ . Furthermore, it is also time-consuming to check whether these candidates are supported by a certain record in the database.

The approach proposed here is able to handle the query through a single scan of the database without the need to generate candidates in advance, thanks to the special features of the path mining problem. The basic idea is to sequentially process each record and use a sliding window to generate all candidate paths that it supports on the fly. In other words, only candidate paths supported by at least one record will be generated and there is no need to explicitly check whether a candidate is supported by a certain record. Our technique is different from traditional sequence mining algorithms in that the number of candidate paths supported by a record is very limited, which is due to the fact that sectors in a candidate path must be directly connected. It is easy to see that a path record containing  $M$  sectors supports at most  $M-N+1$  unique candidate paths with  $N$  sectors. For example, a record with eight sectors {a b c d e f g h} supports the following four candidate paths with five sectors:

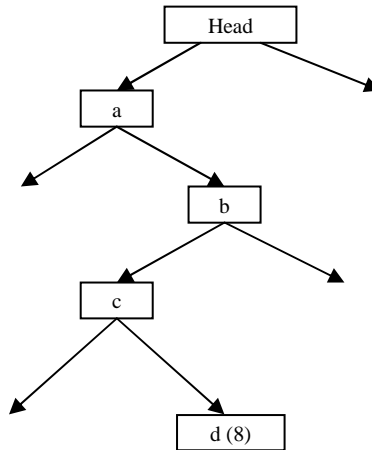
$$\{a b c d e\}, \{b c d e f\}, \{c d e f g\}, \{d e f g h\}$$

The general procedure of the proposed algorithm is given below:

- Step 1:** Select a new record from the database. If all records have been processed, go to Step 5.
- Step 2:** Find out all N-sector candidate paths supported by this record.
- Step 3:** For each candidate path generated in Step 2 that has not been met before, assign a new *id* to it.
- Step 4:** Increase the counters of the above candidate paths. Go to Step 1.
- Step 5:** Return the *id* of the counter with the maximum value.

### 3.2.2 Data Structure

In order to implement the above algorithm, a data structure is required to store candidate paths together with their unique *ids*. Each time a candidate path is found in a record, if it has been seen before, its *id* will be retrieved. Otherwise, it will be added into the data structure and a new *id* is created. For this purpose, a tree structure *T* is adopted where each node corresponds to a sector and *ids* are stored in the leaf nodes.



**Fig. 3.** An example of the tree structure storing candidate paths with four sectors.

An example of *T* is shown in Figure 3 where the path {a b c d} is stored in the tree and assigned an *id* 8. For convenience, all *ids* are integer numbers starting from 1 sequentially. The depth of the tree is equal to the number of sectors in the candidate paths and the number of branches below each node is determined by the number of sectors connected to it in *G*, which is usually between 2 and 4 in a typical shop floor. Note that in order to retrieve the *id* of a candidate path, only *N* steps are required. Inserting a new candidate path requires the same number of operations.

The second data structure is an array  $C$  of counters recording the frequency of each candidate path. Each time a new candidate path (with a new  $id$ ) is generated, a new counter is created, increasing the size of  $C$  by one. Since we are only interested in the number of records that support a certain candidate path, instead of the number of times that it appears in the database, only counters corresponding to candidate paths found for the first time in a record will be updated.

### 3.2.3 Time Complexity

Suppose that the average length of records in the database is  $M$ . On average, there are  $(M-N+1)$  candidate paths to be generated in each record, with possible duplicates. For a database with  $K$  records, there are  $(M-N+1) \cdot K$  candidate paths to be processed. As shown in the last section, for each candidate path, retrieving or creating its  $id$  requires searching through  $T$  for  $N$  steps. Other operations such as increasing the values of counters require constant time. The overall time complexity of the proposed algorithm is given by:  $O((M-N+1) \cdot N \cdot K) \approx O(M \cdot N \cdot K)$  for  $M \gg N$ .

Note that once the  $id$  of the most frequent path is identified, this path can be then retrieved through a depth-first search through  $T$ . Alternatively, all candidate paths found during the scan of the database could be stored together with their  $ids$  so that the most frequent path can be directly retrieved once its  $id$  is known.

### 3.2.4 Examples

To better understand how the proposed algorithm works, suppose that the first record in the database contains eight sectors: {a b c d e b c d}. There are six candidate paths with three sectors supported by this record (Table 1).

**Table 1.** A demo of the algorithm working on the first record {a b c d e b c d} with  $N=3$ .

No.	Candidate	ID	Counters
1	{a b c}	1	[1]
2	{b c d}	2	[1, 1]
3	{c d e}	3	[1,1,1]
4	{d e b}	4	[1,1,1,1]
5	{e b c}	5	[1,1,1,1,1]
6	{b c d}	2	[1,1,1,1,1]

In Table 1, the first five candidates have never been found before. As a result, each of them is inserted into  $T$  and assigned a unique  $id$  and new counters are created for each of them. However, for the last candidate {b c d}, it is already available in  $T$  and its previously assigned  $id$  is returned and no new counter is created. Also, since this  $id$  has already been counted in the same record, the existing value of the corresponding counter is kept unchanged.

Next, the algorithm moves to the second record. Suppose that the first candidate in the second record is {c d e}, which is not a new candidate (No.3 in Table 1) and no new  $id$  is assigned to it. However, since it is the first time that this candidate is found in the second record, its counter will be increased by one and the values of counters

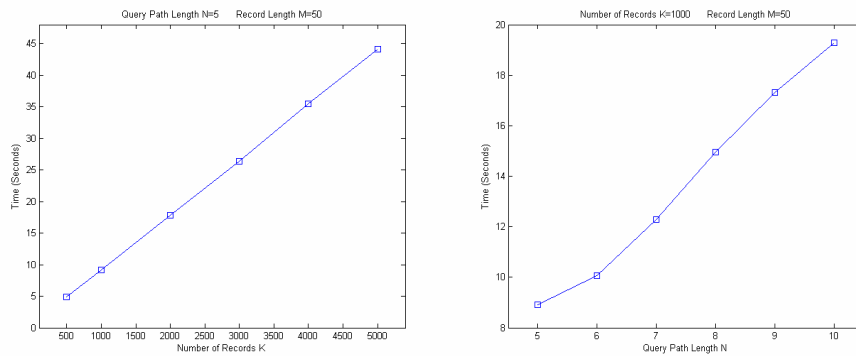
are [1, 1, 2, 1, 1] now. Similarly, if the second candidate is {d e b}, the fourth element in the counter array  $C$  will also be increased by one (i.e.,  $C$  is [1, 1, 2, 2, 1]).

### 3.2.5 Simulations

A test database was randomly generated with 5000 path records, each containing 50 sectors. The layout of the shop floor is based on Figure 1, which consists of 18 sectors with each sector directly connected to 2 to 4 other sectors.

The first experiment was aimed at showing the running time of the algorithm with regard to the number of records in the database with  $K=500, 1000, 2000, 3000, 4000$  and 5000. The second experiment was conducted to show the running time of the algorithm with regard to the length of the candidate path with  $N=5, 6, 7, 8, 9$  and 10.

Experiments were conducted on a PIII 800MHz PC with Matlab 7. Experimental results shown in Figure 4 suggest that the algorithm scaled approximately linearly.



**Fig. 4.** The scalability of the proposed algorithm with regard to the number of records (left) and the length of the candidate paths (right).

## 3.3 Finding the Longest Common Path with a Given Support

The objective of this query is to find out the path with as many sectors as possible, subject to a predefined threshold of support.

### 3.3.1 Problem Analysis

Since the length of candidate paths are not known in advance, the solution to this query is to iteratively apply the algorithm proposed in Section 3.2 with sequentially increasing  $N$  values (i.e.,  $N=1,2, \dots$ ) until the support of the most popular candidate path falls below the threshold. However, the major difficulty is that the cost of the algorithm will increase as the value of  $N$  grows due to its  $O(M \cdot N \cdot K)$  time complexity. In the meantime, the storage space required may also increase greatly.

An important fact is that the support of a path is no more than the minimum support of any of its sub-paths (i.e., one or more sequentially connected sectors contained in the original path). In other words, if it is known that a certain sub-path does not have

the minimum support from the database, it is always true that any longer path that contains this sub-path will not have the minimum support either.

This feature makes it possible to do some pruning of the search space based on the information gained in earlier passes. Consequently, the computational cost of handling longer candidate paths may be substantially reduced.

Note that the number of candidate paths of length  $N$  to be processed in a record with  $M$  sectors is  $M-N+1$ . The key point here is how to reduce this number so that only a much smaller set of candidate paths will need to be processed.

For example, consider a record with eight sectors:

{a b c d e f g h}

It is easy to see that it supports totally six candidate paths with three sectors:

{a b c}, {b c d}, {c d e}, {d e f}, {e f g}, {f g h}

Suppose, based on previous passes, it is known that sectors  $d$  and  $f$  do not meet the minimum support requirement. As a result, any candidate path containing one or both of them is deemed to be infrequent. In this case, five out of six candidates are such examples and only the first candidate {a b c} needs to be processed.

### 3.3.2 Algorithm Framework

According to the above analysis, the algorithm for finding the longest common path with a given support works as follows:

**Step 1:**  $N=1$

**Step 2:** Scan the database to find out the support of all  $N$ -sector candidate paths that are not deemed to be infrequent.

**Step 3:** If no candidate paths have the minimum support, STOP.

**Step 4:** Update the database  $D$  by marking infrequent candidate paths.

**Step 5:**  $N=N+1$ , go to Step 2.

In this framework, the algorithm starts by scanning the database once to check the support of each single sector ( $N=1$ ) following the procedure described in Section 3.2. All sectors that are found infrequent are marked, which is used to prune the search space. In the next pass, the algorithm only tries to find out the support of all candidate paths with two sectors ( $N=2$ ) that are not deemed to be infrequent. This process is repeated until no more candidate paths meet the support threshold.

### 3.3.3 Algorithm Details

In order to mark infrequent candidates in the database, we need the location information of each candidate in the database, which is stored in three arrays during the scan. The first array stores the *id* of each candidate path while the second and third arrays store the No. of the record where this candidate is found as well as the position of the candidate in the record. Note that we sequentially store the information of all candidate paths and the maximum length of the above three arrays is  $(M-N+1) \cdot K$ .

Suppose that {a b c d e f g h} is the 8<sup>th</sup> record in the database and the candidate path {c d e} is assigned an *id* 5. In this case, the values of the corresponding elements in the three arrays are {5}, {8} and {3} respectively, due to the fact that {c d e} has an *id* 5 and is the third candidate path in the 8<sup>th</sup> record.

After each pass, in order to mark sectors in infrequent candidates, we go through the array with *ids* and check the support of each *id* in its counter  $C(id)$ . If it is below the support threshold, its location information stored in the other two arrays will be retrieved to locate this candidate path in the database.

A data structure called *mask* is maintained with the same dimensions as the original database. Each element in *mask* corresponds to a sector in the database. Initially, all elements are set to 0 (i.e., all sectors are available). At the end of the Nth pass, when a candidate path is found infrequent, the element in *mask* corresponding to the first sector in this candidate path is set to N indicating that the path of length N starting with this sector is infrequent (i.e., longer paths are also infrequent).

Recall that in Section 3.2, each of the first  $M-N+1$  sectors in a record of length M is used as the head sector of a candidate path with N sectors. With the information in *mask*, it is now possible to reduce the number of candidate paths to be generated by excluding those that are known to be infrequent. Each time a head sector is selected, the elements in *mask* corresponding to it as well as N-1 consecutive sectors are checked sequentially. For  $1 \leq i \leq N$ , if the value  $x$  of the element in *mask* corresponding to the  $i^{\text{th}}$  sector in the candidate path is greater than zero but no more than  $N-i+1$ , it can be determined that this candidate path is infrequent because it contains at least one infrequent sub-path of length  $x$ . Note that the element corresponding to the head sector of such a candidate path should also be set to N if its original value is 0.

For example, suppose that a record is:

{a b c d e f g h}

The group of elements corresponding to this record in *mask* have initial values:

{0 0 0 0 0 0 0}

After the first pass with  $N=1$ , suppose that only d is found to be infrequent. The values of the elements are updated to:

{0 0 0 1 0 0 0}

In the second pass with  $N=2$ , the algorithm will not generate the candidates {c d} and {d e} because both of them contain d, which is an infrequent sub-path of length one, and are known to be infrequent. The element corresponding to c in *mask* is set to 2 indicating that the path of length two starting with c is infrequent. In the meantime, if candidate path {f g} is also found to be infrequent at the end of the second pass, the element corresponding to f in *mask* will be set to 2. The values of the elements in *mask* at the end of the second pass are updated to:

{0 0 2 1 0 2 0}

As a result, in the third pass with  $N=3$ , instead of having six candidate paths in play, only a single candidate path {a b c} will be generated while others can all be determined to be infrequent based on the existing information in *mask*.

In general, the number of nonzero elements in *mask* is expected to increase after each pass, which will make it more likely for a candidate path to be infrequent. In a word, *mask* is used to avoid generating candidate paths that are already known to be infrequent and thus saves a large amount of computational effort.

### 3.4 Finding Sectors below a Given Purchase Level

The objective of this query is to find out sectors where buyers visit frequently but seldom purchase any products in these sectors.

#### 3.4.1 Problem Analysis

The purchase level of a sector is defined as the ratio between the number of records in which at least a product in that sector is purchased and the number of records in which that sector is visited. A sector with low purchase level means that most customers visiting it are not interested in products in that sector. The reason that it is visited may be simply due to the fact that it is close to the entry/exit of the shop or it is on the way to other sectors with popular products.

Note that we are only interested in knowing that during each single visit to the shop whether a buyer visited a certain sector and whether this buyer purchased any products from that sector. In practice, a buyer may visit a sector multiple times and purchase nothing or multiple products during a single visit to the sector or in different times.

In addition to the database containing records of paths, we assume that there is a database with records of transactions (i.e., lists of products purchased). Also, it is assumed that there is a table specifying the relationship between products and sectors. A condition is that any product is only available in a specific sector (i.e., given any product *id*, there is a unique corresponding sector *id*).

#### 3.4.2 Algorithm Details

The basic idea is to transform the transaction database into a sector database using the product-sector table so that each record in the new database is a set of sectors where shopping activities happened. Note that, unlike the path database, sectors in this new database are likely to be disconnected from each other.

The general procedure of the algorithm is as follows:

- Step 1:** Apply the algorithm in Section 3.2 on the path database ( $N=1$ ) to find out the frequency of each sector.
- Step 2:** Transform the transaction database into a database of sectors using the product-sector table.
- Step 3:** Apply the algorithm in Section 3.2 on the sector database ( $N=1$ ) to find out the frequency of each sector with purchasing activity.
- Step 4:** Calculate the purchase level of each sector.

In the above algorithm, Step 1 is used to find out the frequency of each sector being visited while Step 3 is used to find out the frequency of each sector appearing in the transaction records. Note that the tree structure  $T$  generated in Step 1 is used in Step 3 to make sure that sectors are associated with consistent *ids*. In Step 4, the purchase level of each sector is calculated by the ratio between its frequency in the transaction records and its frequency in the path records.

## 4 Conclusions

In this paper, a smart shop floor setup was proposed to capture real-time buyer behaviour data. We are particularly interested in discovering patterns embedded in the shopping paths of buyers, defined in terms of product shelf sectors. For this purpose, three queries have been identified together with detailed algorithmic solutions. In the meantime, there are some other queries that can be conducted on the data collected from the proposed smart shop floor. For example, it would be interesting to analyse the relationship among sectors to find out the set of sectors that are often visited during the same shopping trip. It may also be of interest to see, if a buyer visited a certain sector, which other sectors this buyer is likely to visit. These queries belong to classical data mining tasks such as mining association rules and sequences, which could be tackled by existing techniques [8, 9].

Furthermore, real-time path planning function can be incorporated into the smart trolley to guide shoppers through the shop floor, according to their specific shopping lists. Also, more advanced queries can be supported by introducing additional functionality into the proposed infrastructure. For example, sector indicators can be designed to transfer time stamp information in addition to the sector identification to the smart trolley. By doing so, it is possible to conduct time-related queries such as finding the sectors where buyers spend most of their time.

## References

- [1] Alexander, K., Gillian, T., Gramling, K., Kindy, M., Moogimane, D., Schultz, M., Woods, M.: "IBM Business Consulting Services – Focus on the Supply Chain: Applying Auto-ID within the Distribution Center", Auto-ID Center, White paper IBM-AUTOID-BC-002, Sep. 2003
- [2] Finkenzeller, K.: RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification. John Wiley & Sons, 2nd Edition, May 2003
- [3] Bornhovd, C., Lin, T., Haller, H., Schaper, J.: Integrating Automatic Data Acquisition with Business Processes – Experiences with SAP's Auto-ID Infrastructure. Proceedings of the 30<sup>th</sup> VLDB Conference, Toronto, Canada, 2004
- [4] EPCGlobal: EPC Tag Data Standards Version 1.1 Rev. 1.24, EPCGlobal, Standards Specification, Apr. 2004, <http://www.epcglobalinc.org>.
- [5] Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. In the 20<sup>th</sup> International Conference on Very Large Data Bases, pp. 487-499, 1994
- [6] Agrawal R., Srikant R.: Mining Sequential Patterns. In the 11<sup>th</sup> International Conference on Data Engineering, pp. 3-14, 1995
- [7] Srikant R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In the 5<sup>th</sup> International Conference on Extending Database Technology, pp. 3-17, 1996
- [8] Luo, C., Chung, S.: A Scalable Algorithm for Mining Maximal Frequent Sequences Using Sampling. In the 16<sup>th</sup> International Conference on Tools with Artificial Intelligence, pp. 156-165, 2004
- [9] Maimon, O., Rokach, L.: The Data Mining and Knowledge Discovery Handbook. Springer, 2005