Sensors **2015**, *15*, 5344-5375; doi:10.3390/s150305344

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

A Framework for Mining Actionable Navigation Patterns from In-Store RFID Datasets via Indoor Mapping

Bin Shen [1], Qiuhua Zheng [2], Xingsen Li [1] and Libo Xu [1,*]

[1] Ningbo Institute of Technology, Zhejiang University, Ningbo 315100, China;
E-Mails: tsingbin@nit.zju.edu.cn (B.S.); lixs@nit.zju.edu.cn (X.L.)

[2] School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China; E-Mail: zheng_qiuhua@163.com (Q.Z.)

***** Author to whom correspondence should be addressed; E-Mail: xu_libo@163.com;
Tel./Fax: +86-574-8813-0015.

External Editors: Kourosh Khoshelham and Sisi Zlatanova

Received: 11 December 2014 / Accepted: 27 February 2015 / Published: 5 March 2015

Abstract: With the quick development of RFID technology and the decreasing prices of RFID devices, RFID is becoming widely used in various intelligent services. Especially in the retail application domain, RFID is increasingly adopted to capture the shopping tracks and behavior of in-store customers. To further enhance the potential of this promising application, in this paper, we propose a unified framework for RFID-based path analytics, which uses both in-store shopping paths and RFID-based purchasing data to mine actionable navigation patterns. Four modules of this framework are discussed, which are: (1) mapping from the physical space to the cyber space, (2) data preprocessing, (3) pattern mining and (4) knowledge understanding and utilization. In the data preprocessing module, the critical problem of how to capture the mainstream shopping path sequences while wiping out unnecessary redundant and repeated details is addressed in detail. To solve this problem, two types of redundant patterns, *i.e.*, loop repeat pattern and palindrome-contained pattern are recognized and the corresponding processing algorithms are proposed. The experimental results show that the redundant pattern filtering functions are effective and scalable. Overall, this work builds a bridge between indoor positioning and advanced data mining technologies, and provides a feasible way to study customers' shopping behaviors via multi-source RFID data.

Keywords: RFID; indoor mapping; shopping transaction path mining; data preprocessing; filtering redundant patterns; framework

1. Introduction

Nowadays, various types of sensors, such as Kinect sensor [1], inertial motion units [2], ultrasound range sensor [2], GPS [3], Radio Frequency Identification (RFID), laser scanners [4] and remote sensor networks [5], have been used to perceive the physical environment, and many promising solutions [2,6,7] have been proposed to realize the effective mapping from the physical world to the cyberspace. Among them, RFID technology is one of the most promising technologies. Based on geometric mapping, it has been increasingly used in various application scenarios, such as intelligent exhibition halls, goods tracking and congestion detection in logistics and distribution [8], abnormal activity detection and insecurity factor detection in access control, and so on.

In the grocery industry, when various items, not only returnable transport carts, trolleys and kegs, but also valuable products, are equipped with RFID tags, item-level RFID infrastructures are established [9]. They can be utilized to realize a wide range of smart applications, e.g., auto check-outs [9], item-level valuable merchandise tracking [10], vendor managed inventory [11], smart price tags [9], *etc.* [12,13]. Among them, one of the more attractive applications is tracking customers' shopping paths. These paths can be captured based on identifying the moving trajectories of smart shopping carts/trolleys/keys which are tagged with RFID [14]. Besides that, shopping carts/trolleys/keys featuring RFID readers can recognize valuable products put into the carts/trolleys/keys, if each valuable product is tagged with an RFID label. As a result, both the walking trajectories of customers and the corresponding purchase behaviors are automatically recorded in the RFID datasets, which are quite precious for mining in-depth knowledge about the shopping behaviors of customers.

Clearly, in the competitive retail climate, discovering insights from the shopping behaviors of customers and then turning these insights into promotion and customer care actions are crucial for enhancing retail business and service quality [15]. Towards this vision, one fundamental work is to study customers' shopping paths in conjunction with their purchasing behaviors. The quick development of in-door positioning [6,7,16–23] and data mining [24–32] technologies sheds light on the above problem, and motivates us to consider building a bridge between RFID-based indoor mapping and advanced data mining techniques to explore customer's shopping behavior in depth.

Consequently, in this study, we propose a framework for mining actionable navigation patterns using multi-source RFID data, *i.e.*, shopping path data and RFID-supported customers' purchasing behavior data. Actionable navigation pattern [24,25] is very useful for understanding behaviors of customers and can be applied to various applications, such as customer navigation, active advertising and recommendations, *etc.* In this framework, we first use the path graph to map the problem in the physical space to a problem in the cyber space, where shopping paths are represented by sequences of path segments. After the mapping between the physical space and the cyber space, the problem of RFID-based shopping path analytics is converted to sequential pattern analysis [26], which has plenty of research in data mining field [27–32] for further reference.

This paper is organized as follows: first, we introduce indoor mapping technologies and related terms in Sections 2 and 3, respectively. Then, in Section 4, the framework for mining multi-source in-door RFID data is presented, and four modules are discussed in detail, which are: (1) mapping from the physical space to the cyber space; (2) data preprocessing; (3) pattern mining and (4) knowledge understanding and utilization. In Section 5, we address a key problem existing in the data preprocessing module, which is how to identify the mainstream shopping transaction paths while wiping out unnecessary redundant and repeated details. An algorithm which can filter two types of redundant patterns is also proposed. Then, a simulated shopping path generator is discussed in Section 6, and the experimental evaluation of the algorithm is given in Section 7. Finally, we discuss the contributions towards a real supermarket scenario and conclude our work in Sections 8 and 9, respectively.

2. Indoor Mapping

To easily comprehend our proposed framework, we provide below a broad overview of indoor mapping technologies.

2.1. Overview

With the progress in sensor technology, many promising indoor-mapping solutions [1–3,6,7,16,17], which can provide precise (or proximity), reliable and robust positioning services, have been proposed. Commonly, an indoor-mapping solution contains two components: (1) a physical-layer for sensing and (2) a software-realized data processing and location positioning, where the sensing capability is based on various available technologies, such as ultra-wideband (UWB), RFID, wireless local area network (WLAN), Bluetooth, ultrasound and video cameras, or the combination of these technologies [6,16,17]. On the basis of sensing, location positioning can be achieved using positioning algorithms, which can be mainly divided into three categories: triangulation, scene analysis and proximity [17]. Triangulation schemes employ geometrical property-based techniques, which are typically time of arrival (TOA), time difference of arrival (TDOA), round-trip time-of-flight (RTOF), angle of arrival (AOA) and received signal strength (RSS) [6]. Scene analysis approaches commonly involve two phases: an offline phase of training and an online phase of positioning. In the offline phase, fingerprints of scenes are collected and stored; during the online recognition phase, machine learning methods (e.g., extreme learning machine [18]) are adopted to compare the observed fingerprints with pre-measured fingerprints for position determination [17].

2.2. RFID-Based Indoor Positioning

Among the above technologies, RFID is an attractive option for coarse grained localization which provides proximity position information, because it is relatively cost-effective and is quite suitable for tracking a large number of items. Therefore, RFID technology is selected in our application of tracking shopping carts and purchased items in a supermarket.

Non-contact RFID positioning systems include three components: RFID readers, tags and servers, where tags can be active or passive. Active RFID tags equipped with internal batteries can broadcast their signals initiatively, and provide a much longer signal transmission range than passive tags; while passive tags are powered by signals transmitted from RFID readers [19,22]. Several basic frequency bands are employed by RFID systems, which include low frequency (LF), high frequency (HF), very high frequency (VHF), ultra-high frequency (UHF) and microwave frequency. Different frequency bands offer different read ranges which normally vary from 10 cm to 12 m, and are suited for different applications [20]. Representative RFID-based precise location sensing systems are SpotON [21] and LANDMARC [22], where reference tags are employed as landmarks. Typical work towards RFID- based proximity positioning includes tracking materials on construction job sites by combining proximity reads from a discrete range [23].

Fault tolerance is another important issue for RFID-based positioning system. The faults (false positive/negative readings) may be caused by many factors, such as hardware failures (e.g., malfunction, running out of battery energy), multipath interference, or complex radio propagation [33]. Countermeasures can be divided into two categories: physical solutions that are based on hardware performance improvement [34], and intelligent software solutions that are based on spatial-temporal correlations/redundancy [33,35].

3. Materials for the Study

In this section, related concepts are defined, and the notations used in this study are summarized in Table 1.

Definition 1. A *path segment s* is a directed edge associated with a direction symbol (*s.dir*), two terminal points (one is the start terminal point *s.b* and the other is the end terminal point *s.e*), and its length (*s.l*). The path segment only can be travelled from *s.b* to *s.e*. The reverse-order path segment of *s* is the path segment sharing the same edge with *s* but reverse direction, *i.e.*, $s_{reverse}$, where $s_{reverse}.dir$ and *s.dir* are reverse, $s_{reverse}.b$ equals *s.e*, $s_{reverse}.e$ equals *s.b*, and $s_{reverse}.l$ and *s.l* are equal.

Definition 2. A *path graph G* is a directed graph, *i.e.*, *G* = (*V*, *E*), where *V* is the set of terminal points of path segments, and *E* is the set of path segments. Path graph *G* is an abstraction of the connections of path segments in a real field.

Definition 3. A *shopping path SP* is a sequence of path segments, *SP* = $< s_1, s_2, \cdots, s_n >$, where $s_{i+1}.b = s_i.e$ and $s_j \in E$, ($1 \le i \le n-1$, $1 \le j \le n$). The beginning point and the ending point of *SP* can be represented as $SP.b = s_1.b$ and $SP.e = s_n.e$.

Definition 4. There are several concepts related to shopping paths as given below:

(1) Given two shopping paths, *i.e.*, *SP* = $< s_1, s_2, \cdots, s_n >$ and $SP' = < s'_1, s'_2, \cdots, s'_l >$ ($l \le n$), if there exists *i*, such that $s'_1 = s_i$, $s'_2 = s_{i+1}$, …, $s'_l = s_{i+l-1}$, then *SP* is a super-sequence of SP', and SP' is a subsequence of *SP* (denoted as $SP' \subset SP$). We also call that SP' is contained in *SP*.

(2) A navigation pattern *NP* means a subsequence of a shopping path.

(3) For a shopping path *SP* = $< s_1, s_2, ..., s_k, s_{k+1}, ..., s_n >$, the reverse-order path of *SP* is $SP_{reverse} = < s_{n,reverse}, \cdots, s_{k+1,reverse}, s_{k,reverse}, ..., s_{2,reverse}, s_{1,reverse} >$, where $s_{i,reverse}$ ($1 \le i \le n$) is the reverse-order path segment of s_i.

(4) Given a shopping path *SP* = $< s_1, s_2, ..., s_k, s_{k+1}, ..., s_n >$, $SP_{prefix} = < s_1, s_2, ..., s_k >$, is called a **prefix** of *SP*, and $SP_{suffix} = < s_{k+1}, s_2, ..., s_n >$ is called a suffix of *SP*, where $1 \le k \le n$.

(5) Given *n* shopping paths, *i.e.*, SP_1, SP_2, …, SP_{n-1} and SP_n, if $SP_i.e = SP_{i+1}.b$ $(1 \le i \le n-1)$ is satisfied, these shopping paths can be connected one after another, and the connection can be marked as $SP_1 \to SP_2 \to \ldots \to SP_n$.

Table 1. Notations.

Notation	Description
s, s_i	A path segment
$s_{reverse}$	The reverse-order path segment of s
$s_i.b$, $s_i.e$	The start terminal point, the end terminal point of s_i respectively
t_i	Unit time per unit length spent in s_i
v_i	A terminal point
T_i	The itemset purchased in s_i
G	A path graph
SP, SP', SQ	A shopping path
$SP' \subset SP$	SP is a super-sequence of SP', and SP' is a subsequence of SP
STP, STP_{prefix}, STP_{suffix}	A shopping transaction path
Trans(STP)	Transforming STP to a shopping path
i_{item}	An item
$i_{item} \lhd STP$	i_{item} is purchased in STP
D	A shopping transaction path database
S_{item}, $\|S_{item}\|$	A set of items, and the number of its elements respectively
Γ_s	The itemset sold in s
SIT	The Segment-Item Table
E_{item}	The set of path segments that sell i_{item}
IST, LT, PT	The Item-segment table, the Length table, and the Path-set table respectively

Definition 5. Given a shopping path *SP*, the connection between *SP* and its reverse-order path $SP_{reverse}$ (*i.e.*, $SP \to SP_{reverse}$) forms a symmetric pattern. If $SP.b = SP.e$, *SP* is called a loop pattern. Given a loop pattern *SP*, if *SP* repeats *n* ($n \ge 2$) times successively, *i.e.*, $SP \to SP \to \ldots \to SP$, we call it a loop repeat pattern. Given a shopping paths, *i.e.*, *SP*, we call the pattern $SP \to SP_{reverse} \to SP$ a *palindrome-contained pattern*.

Definition 6. A *shopping transaction path* is a sequence of triples, $STP = <(s_1,t_1,T_1), (s_2,t_2,T_2), \ldots, (s_n,t_n,T_n)>$, where (s_i, t_i, T_i) means that a shopper purchases the itemset T_i and spends t_i unit time per unit length in the path segment s_i $(1 \le i \le n)$.

Definition 7. Given a shopping transaction path, *i.e.*, $STP = <(s_1,t_1,T_1), (s_2,t_2,T_2), \ldots, (s_n,t_n,T_n)>$, there are several concepts, which are relevant to shopping transaction paths, are given below:

(1) For simplicity, all s_i $(1 \le i \le n)$ are called the path segments of shopping transaction path *STP*.

(2) For a given item i_{item}, if $i_{item} \in T_1 \cup T_2 \cup \ldots \cup T_n$, we call i_{item} is purchased in *STP*, *i.e.*, $\mathrm{i_{item}} \lhd \mathrm{STP}$.

(3) Given a fragment of *STP*, *i.e.*, $STP' = <(s_k,t_k,T_k), (s_{k+1},t_{k+1},T_{k+1}), \ldots, (s_l,t_l,T_l)>$ $(1 \le k \le l \le n)$, we called STP' a subsequence of *STP*, and STP' is contained in *STP*.

(4) $STP_{prefix} = <(s_1,t_1,T_1), (s_2,t_2,T_2), \ldots, (s_k,t_k,T_k)>$, is called a prefix of *STP*, where $1 \le k \le n$. $STP_{suffix} = <(s_{k+1},t_{k+1},T_{k+1}), (s_{k+2},t_{k+2},T_{k+2}), \ldots, (s_n,t_n,T_n)>$ is called a suffix of *STP*.

(5) Given a shopping transaction path, $STP = <(s_1,t_1,T_1), (s_2,t_2,T_2), \ldots, (s_n,t_n,T_n)>$, if only path segments appear in STP, we transform it to a shopping path, $SP = <s_1, s_2,\ldots, s_n>$, which is called the shopping path of STP and denoted as $SP = \text{Trans}(STP)$.

(6) Given n shopping transaction paths, *i.e.*, STP_1, STP_2, …, STP_{n-1} and STP_n, if $\text{Trans}(STP_i).e = \text{Trans}(STP_{i+1}).b$ ($1 \le i \le n-1$) is satisfied, these shopping transaction paths can be connected one after another, and the connection can be marked as $STP_1 \rightarrow STP_2 \rightarrow \ldots \rightarrow STP_n$.

For example, $STP = <(s_1,1,\varnothing),(s_2,0.8,\varnothing),(s_3,8,\{i_1,i_2\}),(s_4,0.8,\varnothing),(s_5,5,\{i_3\})>$ denotes that when the shopper visits $s_1, s_2,\ldots, s_5$ consecutively, he/she spends 1, 0.8, …, 5 unit time per unit length in these path segments respectively. Meanwhile, the shopper purchases $\{i_1, i_2\}$ in s_3 and purchases $\{i_3\}$ in s_5. $s_1, s_2,\ldots, s_5$ are all path segments of STP. Among them, s_1 is the first path segment of STP, s_2 is the second one, …, and s_5 is the last one. Because $i_1 \in \varnothing \cup \varnothing \cup \{i_1, i_2\} \cup \varnothing \cup \{i_3\}$, we have $i_1 \triangleleft STP$. STP can be transformed to a shopping path, that is to say $\text{Trans}(STP) = <s_1, s_2,\ldots, s_5>$.

Definition 8. A *mainstream of shopping path* is a shopping path without containing any loop repeat or palindrome-contained subsequence pattern. A shopping transaction path STP is a *mainstream of shopping transaction path*, if $\text{Trans}(STP)$ is a mainstream of shopping path.

Definition 9. A *Segment-Item Table* (*SIT*), maintaining the information of items sold in each path segment, is denoted as below:

$$SIT = \{(s_1, \Gamma_{s_1}), (s_2, \Gamma_{s_2}), \ldots, (s_W, \Gamma_{s_W})\} \tag{1}$$

where s_i is a path segment, Γ_{s_i} is the itemset sold in s_i ($1 \le i \le W$), and W is the total number of path segments.

Definition 10. An *Item-Segment Table* (*IST*), maintaining the information about the segments where each item is sold, is denoted as below:

$$IST = \{(i_{item,1}, E_{item,1}), (i_{item,2}, E_{item,2}), \ldots, (i_{item,U}, E_{item,U})\} \tag{2}$$

where $i_{item,j}$ is an item, $E_{item,j}$ is the set of path segments that sell $i_{item,j}$ ($1 \le j \le U$), and U is the total number of items.

Definition 11. A *Length Table* (*LT*), maintaining the length information of path segments, is denoted as below:

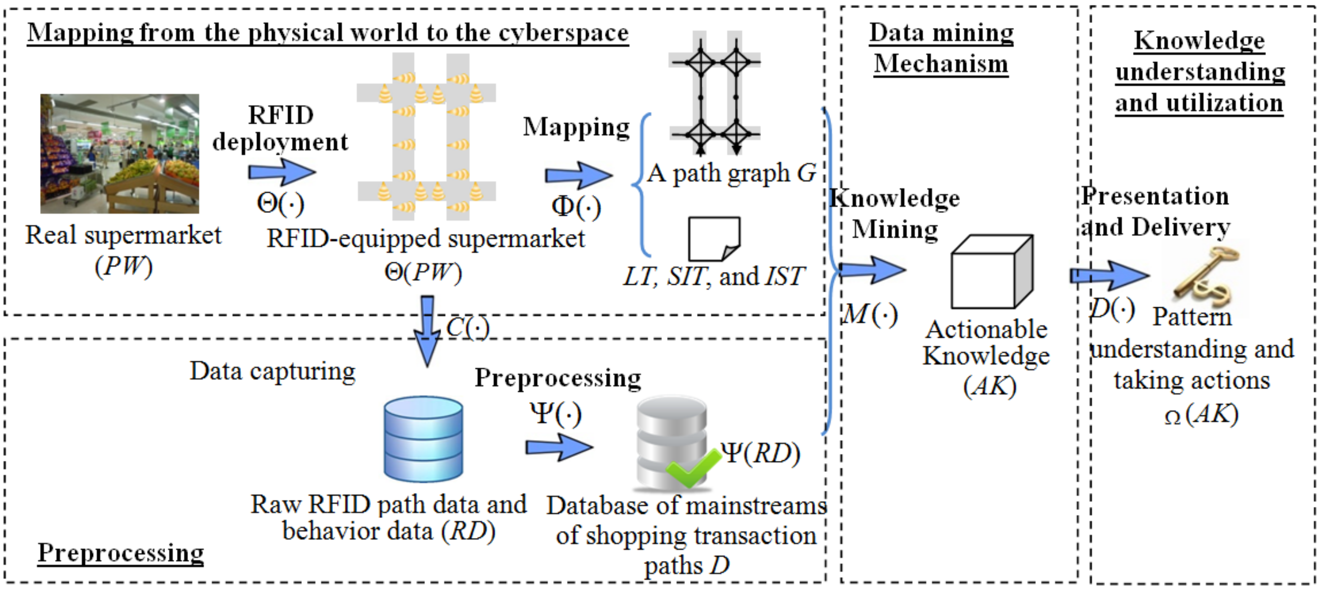
$$LT = \{(s_1, s_1.l), (s_2, s_2.l), \ldots, (s_W, s_W.l)\} \tag{3}$$

where s_i is a path segment, and $s_i.l$ is the length of s_i which can be obtained according to the length of normal trajectory of s_i ($1 \le i \le W$).

4. System Framework of RFID Path Explorer

In this section, we describe the framework of the RFID supported paths and behaviors mining, called *RFID Path Explorer*.

The proposed framework consists of four modules: (1) mapping from the physical world to the cyber space; (2) data preprocessing; (3) a data mining mechanism; and (4) knowledge understanding and utilization (see Figure 1). Table 2 shows an example of shopping transaction path database which contains five shopping transaction paths. Below, we explain them in detail.



Mapping from the physical world to the cyberspace
RFID deployment
Θ(·)
Real supermarket (*PW*)
RFID-equipped supermarket Θ(*PW*)
Mapping
Φ(·)
A path graph *G*
LT, *SIT*, and *IST*
C(·)
Data capturing
Preprocessing
Ψ(·)
Ψ(*RD*)
Raw RFID path data and behavior data (*RD*)
Database of mainstreams of shopping transaction paths *D*
Preprocessing
Data mining Mechanism
Knowledge Mining
M(·)
Actionable Knowledge (*AK*)
Knowledge understanding and utilization
Presentation and Delivery
D(·)
Pattern understanding and taking actions
Ω(*AK*)

Figure 1. The framework for RFID based shopping transaction path mining.

Table 2. An example of shopping transaction path database.

STP_{id}	**Shopping Transaction Path**
1	(AB, 0.8, Ø), (BC, 1, Ø), (CD, 4, {i_1, i_2}), (DE, 3, {i_3}), (EF, 0.8, Ø), (FD, 0.8, Ø), (DK, 0.8, Ø)
2	(AB, 0.9, Ø), (BC, 1, Ø), (CD, 5, {i_1}), (DK, 0.8, Ø)
3	(DK, 0.9, Ø), (KC, 0.8, Ø), (CD, 5, {i_2}), (DE, 5, {i_3})
4	(BC, 0.8, Ø), (CD, 4, {i_1}), (DK, 1, Ø), (KA, 0.8, Ø), (AD, 1, Ø), (DE, 4, {i_3}), (EF, 1, Ø)
5	(DK, 0.9, Ø), (KC, 1, Ø), (CD, 6, {i_2, i_4}), (DK, 1, Ø)

4.1. Indoor Mapping from the Physical World to the Cyberspace

The module of indoor mapping from the physical world (*PW*) to the cyber space consists of two steps as shown in Figure 1.

4.1.1. Step 1. RFID Deployment ($\Theta(\cdot)$)

According to the task of application domain (*i.e.*, finding actionable navigation patterns for purchasing an item), suitable RFID devices should be chosen and deployed in a real field (*i.e.*, real supermarket). For instance, in our application, a RFID tag, which has a unique Electronic Product Code (*EPC*), is attached to each shopping trolley. RFID readers are located at various places of a supermarket, such as the entrance, the checkout, the gathering place for shopping carts, aisles and thoroughfares *etc.*, and used to identify shopping trolleys passing by. At the same time, when valuable items attached with RFID tags are put into a shopping trolley, they also can be recognized by this RFID-reader-equipped shopping trolley. Thus, both shoppers' path and behaviors can be captured ($C(\cdot)$) and recorded. For the sake of robust, redundant multiple readers/tags [23] and received signal strength functions of RFID devices [21] can be added to promote the reliability of proximity location determination.

4.1.2. Step 2. Mapping ($\Phi(\cdot)$)

Recorded raw RFID data can't be understood without the support of semantic information about these data. Therefore, attributes and features related to the analysis task should be abstracted from the physical world and mapped to the cyber-space. In the context of our application, a path graph *G* is used to abstract the connections of path segments, after RFID devices are deployed in a supermarket. An illustration of path graph is shown in Figure 2c, which is mapping from Figure 2b. A segment-item table (*SIT*) and an item-segment table (*IST*) are also extracted to reflect the items sold in each segment and the segments where each item is sold respectively. An example of *SIT* and *IST* is shown in Table 3.

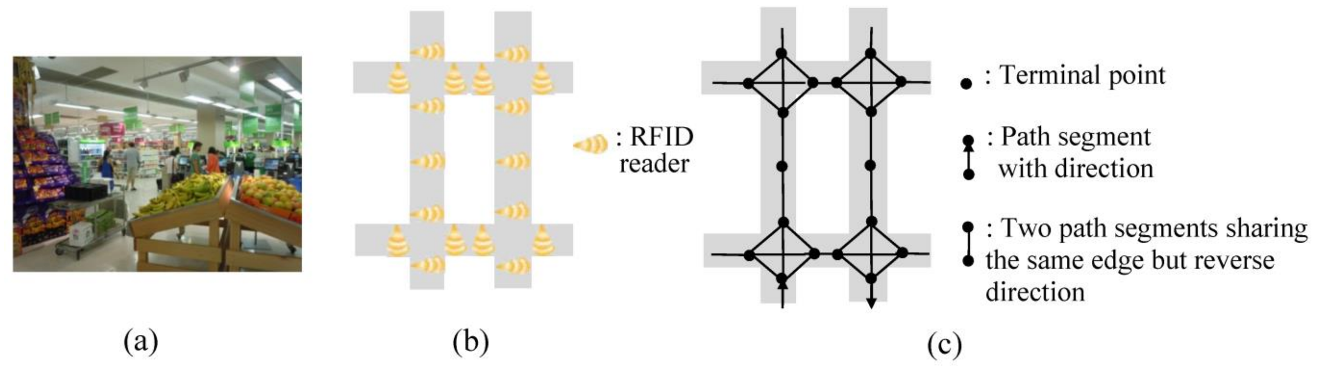

(a)
(b)
: RFID reader
(c)
: Terminal point
: Path segment with direction
: Two path segments sharing the same edge but reverse direction

Figure 2. An illustration of path graph (**a**) A photo of Real supermarket. (**b**) An illustration of a part of supermarket after RFID deployment. (**c**) An illustration of a part of path graph after mapping.

Table 3. An example of *SIT* and *IST*.

Path Segment	Itemset	Itemset	Path Segment
AB	NULL	i_1	{CD, PZ}
BC	NULL	i_2	{CD, DE}
CD	{ i_1, i_2, i_6, i_{11}, i_{12}}	i_3	{GH, PQ, PZ}
DE	{i_2, i_{15}}	i_4	{GJ}
…	…	…	…
PZ	{ i_1, i_3, i_{20}, i_{5000}}	i_{5000}	{PZ}

4.2. Preprocessing

After the raw RFID path and behavior data is captured, the preprocessing is shown in Figure 3.

4.2.1. Step 1. Data Ordering and Data Compression

Raw RFID path data has the form (*EPC*, *Loc*, *Time_stamp*), where *EPC* is the Electronic Product Code of the tag that uniquely represent a shopping cart, *Loc* is the identification location whose reader finds the tag, and *Time_stamp* is the time when the RFID reading takes place [36]. These raw data firstly need to be sorted on *EPC* and *time*, and then be transformed to the form of stay record, *i.e.*, (*EPC*, *Loc*, *T_in*, *T_out*), where *T_in* is the time when the RFID tag enters the identification area, and *T_out* is the leave time [37].

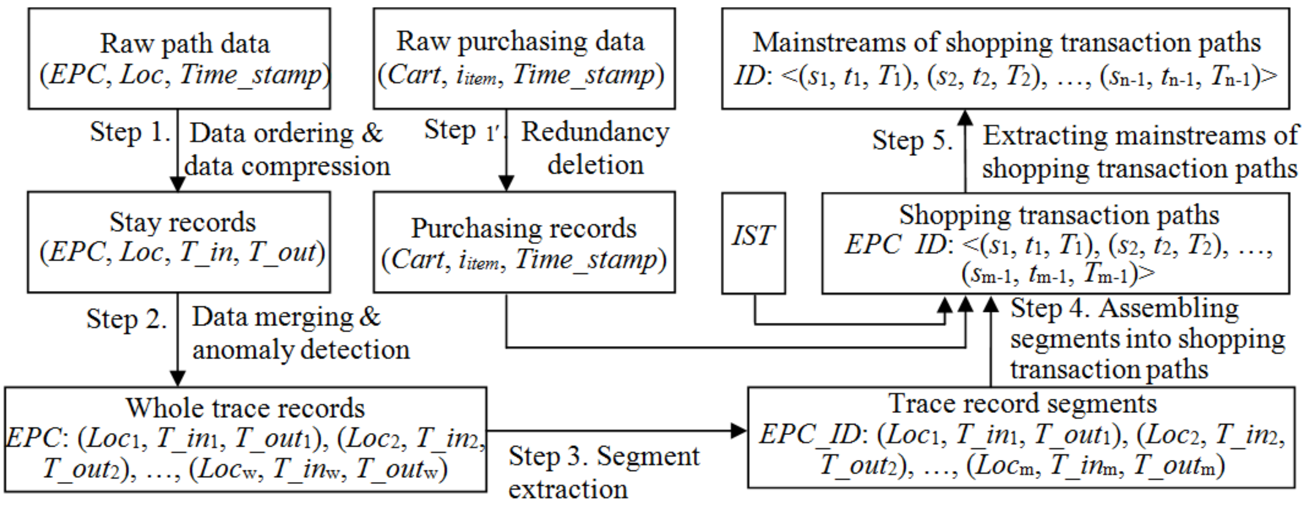

Raw path data (EPC, Loc, Time_stamp)
Raw purchasing data (Cart, i_item, Time_stamp)
Mainstreams of shopping transaction paths ID: <(s1, t1, T1), (s2, t2, T2), …, (sn-1, tn-1, Tn-1)>
Step 1. Data ordering & data compression
Step 1′. Redundancy deletion
Step 5. Extracting mainstreams of shopping transaction paths
Stay records (EPC, Loc, T_in, T_out)
Purchasing records (Cart, i_item, Time_stamp)
IST
Shopping transaction paths EPC_ID: <(s1, t1, T1), (s2, t2, T2), …, (sm-1, tm-1, Tm-1)>
Step 2. Data merging & anomaly detection
Step 4. Assembling segments into shopping transaction paths
Whole trace records EPC: (Loc1, T_in1, T_out1), (Loc2, T_in2, T_out2), …, (Locw, T_inw, T_outw)
Step 3. Segment extraction
Trace record segments EPC_ID: (Loc1, T_in1, T_out1), (Loc2, T_in2, T_out2), …, (Locm, T_inm, T_outm)

Figure 3. RFID data preprocessing.

When an item (*i.e.*, i_{item}) is put into a shopping cart (*i.e.*, *Cart*), a raw purchasing data (*i.e.*, (*Cart*, i_{item}, *Time_stamp*)) is also generated, where *Time_stamp* is the time of detecting the item. And then, raw purchasing data is continuously produced, until the item is picked out of the shopping cart. Therefore, for an item, only the record of first reading needs to be saved, which marks purchasing of the item happens.

4.2.2. Step 2. Data Merging and Anomaly Detection

For a *EPC*, from the stay records, a whole trace record can be constructed, which has the form *EPC*: (Loc_1, T_in_1, T_out_1), (Loc_2, T_in_2, T_out_2), …, (Loc_w, T_in_w, T_out_w), where Loc_i is the location where the tag is detected, T_in_i and T_out_i are the entering time and the leaving time of Loc_i respectively, and (Loc_i, T_in_i, T_out_i) is ordered by T_in_i ($1 \leq i \leq W$) [38]. For a whole trace record, if there are two same successive locations (*i.e.*, $Loc_i = Loc_{i+1}$), then (Loc_i, T_in_i, T_out_i) and (Loc_{i+1}, T_in_{i+1}, T_out_{i+1}) can be merged to (Loc_i, T_in_i, T_out_{i+1}). Thus we have a whole trace record where any two successive locations are different.

Here, any two consecutive locations are required to meet spatial constraint [35] that these two locations should be directly connected in the path graph. Two successive locations which cannot satisfy this constraint are labelled as an anomaly, and the anomaly should be checked further to infer/determine whether missed readings or false positive readings occur. If permanent/intermittent/transient faults leading to this anomaly can be identified, missed readings are filled in and false positive readings are discarded to make the whole trace record smoothly connected; otherwise, suspicious readings are removed and the remaining parts of the trace record are kept separately. This prior-knowledge based validation mechanism can further promote the reliability of sensing.

4.2.3. Step 3. Segment Extraction

In the supermark scenario, shopping carts are recycled and used by different shoppers at different shopping times. Thus, the whole trace record of a certain *EPC*, which is the proxy of a cart, commonly contains multiple shopping trips of various shoppers. Besides, path sequences of supermarket staff collecting shopping carts may also be contained. Therefore, in order to study customers' shopping behavior, it is necessary to extract individual shopping trips from the whole trace records.

For the above purpose, we develop a finite state machine model [39] for shopping carts (see Figure 4) by referring to the real situation in supermarkets. In Figure 4, there are four states for a shopping cart: "idle", "shopping", "discarded" and "end" states. The "idle" state implies that the cart stays in the gathering place for shopping carts and is currently available. The "shopping" state indicates that the cart is in use by a shopper. The "discarded" state means that the cart is discarded midway by a shopper, and the "end" state implies the cart arriving at the checkout and the end of a shopping trip.

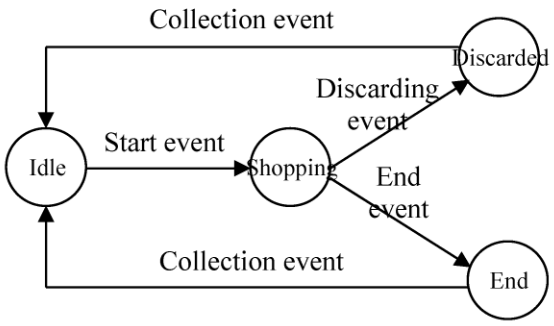


Figure 4. The finite state machine model for shopping carts.

In the model, a state will be transformed to another one, if a certain event [40] is triggered. The initial state of a cart may be "idle". If a "start" event happens, the "idle" state will become the "shopping" state, where the "start" event can be defined as the observation that a shopping cart leaving the gathering place and entering the main entrance. Then, the "shopping" state will be changed to the state of "discarded" if the "discarding" event happens or to the "end" state if the "end" event takes place. Both the "discarded" and the "end" states are followed by the "collection" event, and will be transformed to the "idle" state again. Here, the "discarding", the "end" and the "collection" events also should be defined according to the real situation. For example, we may define the "discarding" and the followed "collection" events as a long stay in a certain location and then moving to the gathering place directly, and the "end" event as a cart arriving checkout with items taken out of the shopping cart. Thus trace record segments can be extracted from a whole trace record, where each segment represents a single shopping trip.

4.2.4. Step 4. Assembling Segments into Shopping Transaction Paths

In order to analyze shopping paths, terminal-points focused trace record segments need to be further transformed to path-segments focused shopping transaction paths, which also combine the information of purchased items. The transform process is shown below.

First, given a trace record segment of a shopping cart (*Cart*), *i.e.*, *EPC_ID*: (Loc_1, T_in_1, T_out_1), (Loc_2, T_in_2, T_out_2), …, (Loc_m, T_in_m, T_out_m), and a length table (*LT*), the trace record segment can be converted to a shopping path with time information in the form of *EPC_ID*: (s_1, t_1), (s_2, t_2), …, (s_{m-1}, t_{m-1}), where s_i is the path segment connecting two successive locations (*i.e.*, Loc_i and Loc_{i+1}), and t_i represents the time spent in s_i per unit length of s_i ($1 \le i \le m-1$). That is to say, t_i equals $time_i / s_i.l$, where $time_i$ representing the time spent in s_i is assumed as $(T\_in_{i+1}+T\_out_{i+1})/2-(T\_in_i+T\_out_i)/2$ for simplicity ($1 \leq i \leq m$-1), and $s_i.l$ is the length of s_i. Second, suppose the corresponding set of purchasing records of *Cart* is $\{(Cart, i_{item,1}, Time\_stamp_{item,1}), \ldots, (Cart, i_{item,n}, Time\_stamp_{item,n})\}$, where $T\_in_1 \leq Time\_stamp_{item,j} \leq T\_out_m$ ($1 \leq j \leq n$). Then for each $(Cart, i_{item,j}, Time\_stamp_{item,j})$, given $IST = \{(i_{item,1}, E_{item,1}),\ldots,(i_{item,U}, E_{item,U})\}$, there are four cases to decide which path segment $i_{item,j}$ is purchased in.

Case 1. If $\exists\, 2 \leq i \leq m$-1, $T\_out_i \leq Time\_stamp_{item,j} \leq T\_in_{i+1}$, then obviously $i_{item,j}$ is purchased in s_i.

Case 2. If $T\_in_1 \leq Time\_stamp_{item,j} < T\_out_1$, then obviously $i_{item,j}$ is purchased in s_1; if $T\_in_m < Time\_stamp_{item,j} \leq T\_out_m$, clearly, $i_{item,j}$ is purchased in s_{m-1}.

Case 3. If $\exists\, 2 \leq i \leq m$-1, $T\_in_i < Time\_stamp_{item,j} < T\_out_i$, $s_i \in E_{item,j}$ and $s_{i-1} \notin E_{item,j}$, then it can be deduced that $i_{item,j}$ is purchased in s_i. If $\exists\, 2 \leq i \leq m$-1, $T\_in_i < Time\_stamp_{item,j} < T\_out_i$, $s_i \notin E_{item,j}$ and $s_{i-1} \in E_{item,j}$, then it can be derived that $i_{item,j}$ is purchased in s_{i-1}.

Case 4. If $\exists\, 2 \leq i \leq m$-1, $T\_in_i < Time\_stamp_{item,j} < T\_out_i$, $s_i \in E_{item,j}$ and $s_{i-1} \in E_{item,j}$, we can't judge which path segment $i_{item,j}$ is purchased in among s_{i-1} and s_i. In this case, besides the above conditions are met, if $T\_in_i < Time\_stamp_{item,j} \leq (T\_in_i + T\_out_i)/2$, $i_{item,j}$ is most probably purchased in s_{i-1}; otherwise, if $(T\_in_i + T\_out_i)/2 < Time\_stamp_{item,j} < T\_out$, $i_{item,j}$ is most likely purchased in s_i.

Thus, after all items (*i.e.*, $i_{item,j}$ ($1 \leq j \leq n$)) are added to the corresponding item set of path segment where this item is purchased, we can obtain a shopping transaction path, *i.e.*, *EPC_ID*: $<(s_1, t_1, T_1), (s_2, t_2, T_2), \ldots, (s_{m-1}, t_{m-1}, T_{m-1})>$, where T_i ($1 \leq I \leq m$-1) is the item set purchased in s_i.

4.2.5. Step 5. Extracting Mainstreams of Shopping Transaction Paths

In the context of web browsing, Chen *et al.* [27] first introduced the concept of maximal forward reference, and proposed a method of breaking a user session down into several maximal forward references if backward references appear in this session. However, several limitations still exist in the scheme of extracting maximal forward reference.

First, their extraction method assumes that backward references are all for easy of travelling, and not for browsing. But this assumption fails in the context of a real supermarket. There are two intentions for a shopper choosing to go backward. One is trying to explore and purchase in the backward reference, and the other is going through the backward reference to other interested sections, so the method of Chen *et al.* [27] can't be applied here.

Second, their method throws away all backward references, which might provide important clues on shoppers' purchasing and navigation behaviors.

Third, after applying their method, the frequency of prefix sequence in front of symmetric pattern is increased unexpectedly. For instance, suppose there is a shopping path < AB, BC, CD, DC, CE, EF, FG, GH, HG, GI > shown in Figure 5, where two symmetric patterns exist, *i.e.*, < CD, DC > and < GH, HG >. After applying the method of Chen *et al.* [27], the set of maximal forward shopping path is < AB, BC, CD >, < AB, BC, CE, EF, FG, GH >, < AB, BC, CE, EF, FG, GI >. We can find that the frequency of prefix sequence < AB, BC > unexpectedly becomes 3, while it is 1 in the original shopping path. The frequency of prefix sequence < AB, BC, CE, EF, FG > is converted to 2, while it is 1 in the original shopping path.

Fourth, a maximal forward reference terminates if a backward reference appears. Thus, a maximal forward reference will not contain any symmetric pattern. This may lead to an unexpected loss of important knowledge on symmetric pattern. For example, from the shopping path < AB, BC, CD, DC, CE, EF, FG, GH, HG, GI > shown in Figure 5, we know that this customer would like to go to the aisle of CD first, turn back to the main thoroughfare of EF, go forward to the aisle of GH, and then be back to the main thoroughfare again. The knowledge on customer's turning back disappears in the set of maximal forward shopping path.

Therefore, instead of finding maximal forward references, we present another new scheme called *extracting mainstream of shopping transaction path*, which reserves necessary symmetric patterns, but discards redundant and repeated details. This scheme is discussed in the next subsection in detail.

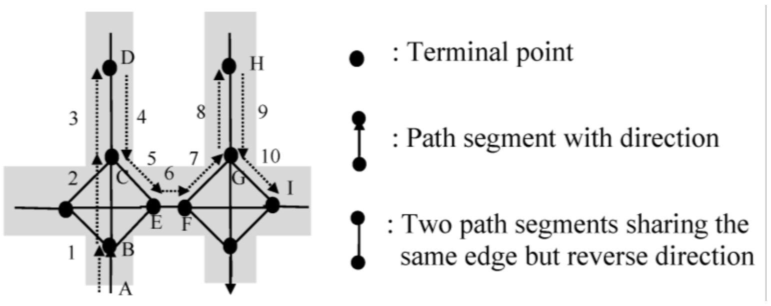

D
H
3
4
8
9
5
7
10
2
C
6
G
I
E
F
1
B
A
● : Terminal point
: Path segment with direction
: Two path segments sharing the same edge but reverse direction

Figure 5. An example for identifying maximal forward reference.

4.3. Mainstreams of Shopping Transaction Paths

In the environment of a real supermarket, in order to choose items of interest, shoppers are inclined to push/pull shopping carts forward and backward. Symmetric patterns, loop patterns and redundant details may appear in shopping paths. In order to catch the mainstream of path sequences while discarding unnecessary redundant and repeated details, we put forward a scheme for identifying mainstream shopping transaction paths. In this scheme, we recognize that two types of redundant patterns need to be simplified, *i.e.*, loop repeat patterns and palindrome-contained patterns. .

4.3.1. Processing of Loop Repeat Patterns

Successive repeated path sequence loops actually reflect the same shopping interest and share the same behavior pattern, so these loops can be combined into one loop. For a shopping path, we can compress several repeat loops into one directly. For a shopping transaction path with a loop repeat pattern, *i.e.*, $STP_{prefix} \rightarrow STP_1 \rightarrow STP_2 \rightarrow \ldots \rightarrow STP_n \rightarrow STP_{suffix}$, where STP_i ($i = 1,\ldots,n$) shares the same navigation pattern, these STP_i ($i = 1,\ldots,n$) also can be combined into a single $STP_{combine} = <(s_1, t_1, T_1), (s_2, t_2, T_2),\ldots, (s_m, t_m, T_m)>$, so we also need to consider specifying values of t_j and T_j in $STP_{combine}$ ($j = 1,\ldots,m$). The time spent in a path segment is normally comprised of two parts: walking time and time for exploring and purchasing. We consider that if a shopper tries to complete the task of exploring and purchasing in one loop (say $STP_{combine}$), which is previously done in multiple loops (say STP_i ($i = 1,\ldots,n$)), time spent in the same path segment of loop should be cumulated, and itemsets purchased in the same path segment but in different loops also should be combined.

Therefore, we have the following definition for simplification of loop repeat pattern:

Definition 12. A shopping path containing loop repeat pattern, *i.e.*, $SP_{prefix} \rightarrow SP \xrightarrow{n} SP \rightarrow SP_{suffix}$, can be simplified as $SP_{prefix} \rightarrow SP \rightarrow SP_{suffix}$. A shopping transaction path containing loop repeat pattern, *i.e.*, $STP_{prefix} \rightarrow STP_1 \rightarrow STP_2 \rightarrow \ldots \rightarrow STP_n \rightarrow STP_{suffix}$, where $STP_i = <(s_1, t_{1,i}, T_{1,i}), (s_2, t_{2,i}, T_{2,i}), \ldots, (s_m, t_{m,i}, T_{m,i})>$, all STP_i share the same navigation pattern (say Trans(TSP) = $<s_1, s_2, \ldots, s_m>$), and m is the number of path segments in STP_i ($i = 1,\ldots,n$), can be simplified as $STP_{prefix} \rightarrow STP_{combine} \rightarrow STP_{suffix}$, where $STP_{combine} = <(s_1, t_1, T_1), (s_2, t_2, T_2), \ldots, (s_m, t_m, T_m)>$, t_{walking} is the smallest value of time spent per unit length in this shopping transaction path, t_j and T_j ($j = 1,\ldots,m$) are defined as below:

$$t_j = t_{\text{walking}} + \sum_{i=1}^{n} t_{\text{purchasing}}(j,i) \quad (j = 1,\ldots,m) \tag{4}$$

$$t_{\text{purchasing}}(j,i) = t_{j,i} - t_{\text{walking}} \quad (i = 1,\ldots,n, j = 1,\ldots,m) \tag{5}$$

$$T_j = \bigcup_{i=1}^{m} T_{j,i} \quad (j = 1,\ldots,m) \tag{6}$$

For instance, for the shopping path <AB, BC, CD, DE, EB, BC, CD, DE, EB, BC, CF> shown in Figure 6, loop <BC, CD, DE, EB> appears two times continuously and forms a loop repeat pattern, so this shopping path can be simplified as a mainstream of shopping path <AB, BC, CD, DE, EB, BC, CF>.

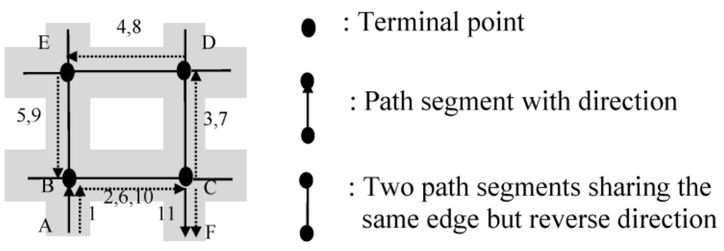

4,8
E
D
5,9
3,7
B
C
2,6,10
1
11
A
F
● : Terminal point
: Path segment with direction
: Two path segments sharing the same edge but reverse direction

Figure 6. An illustrative example of loop repeat pattern.

For a shopping transaction path <(AB, 1, Ø), (BC, 2, Ø), (CD, 3, {i_1}), (DE, 2, Ø), (EB, 6, {i_2, i_3}, (BC, 2, {i_4}), (CD, 5, {i_6, i_7}), (DE, 1, Ø), (EB, 1, Ø), (BC, 1.1, Ø), (CF, 1, Ø)>, a loop repeat pattern <BC, CD, DE, EB>→<BC, CD, DE, EB> appears. Triple elements having the same path segment but different loops should be combined, *i.e.*, (BC, 2, Ø) and (BC, 2, {i_4}), (CD, 3, {i_1}) and (CD, 5, {i_6, i_7}), *etc.* We take the combination of (BC, 2, Ø) and (BC, 2, {i_4}) as an example. First, we obtain that t_{walking} equals 1, which is the smallest one in the set of time spent per unit length in this shopping transaction path. Then, time spent in BC per unit length (t_{BC}) is computed as 1 + (2−1) + (2−1) = 3, and itemset in BC (T_{BC}) is Ø∪{i_4} = {i_4}. Therefore, the result of the combination of (BC, 2, Ø) and (BC, 2, {i_4}) is (BC, 3, {i_4}). Thus this shopping transaction path can be simplified as a mainstream of shopping transaction path < (AB, 1, Ø), (BC, 3, {i_4}), (CD, 7, {i_1, i_6, i_7}), (DE, 2, Ø), (EB, 6, {i_2, i_3}), (BC, 1.1, Ø), (CF, 1, Ø) >. This mainstream shopping transaction path retains the main movement of a shopper and prunes unnecessary details.

4.3.2. Handling Palindrome-Contained Pattern

A palindrome-contained pattern, *i.e.*, $SP \rightarrow SP_{reverse} \rightarrow SP$, is easily formed when a shopper compares the same kind of items in front of a shelf, and shows that shoppers hover in a SP navigation pattern. These three segments, *i.e.*, SP, $SP_{reverse}$ and the second SP, actually show the same shopping interest and share the same behavior pattern, so these successive segments can be simplified as *SP*. Thus, we have the following simplification method of palindrome-contained pattern 1:

Definition 13. A shopping path containing a palindrome-contained pattern, *i.e.*, $SP \rightarrow SP_{reverse} \rightarrow SP$, can be simplified as *SP*. A shopping transaction path containing a palindrome-contained pattern, *i.e.*, $STP_{prefix} \rightarrow STP_1 \rightarrow STP_2 \rightarrow STP_3 \rightarrow STP_{suffix}$, where STP_i = <(s_1, $t_{1,i}$, $T_{1,i}$), (s_2, $t_{2,i}$, $T_{2,i}$),…, (s_m, $t_{m,i}$, $T_{m,i}$)> (i = 1, 3), STP_2 = <($s_{m,reverse}$, $t_{m,2}$, $T_{m,2}$), ($s_{m-1,reverse}$, $t_{m-1,2}$, $T_{m-1,2}$), …, ($s_{1,reverse}$, $t_{1,2}$, $T_{1,2}$)>, can be simplified as $STP_{prefix} \rightarrow STP_{combine} \rightarrow STP_{suffix}$, where $STP_{combine}$ = <(s_1, t_1, T_1), (s_2, t_2, T_2),…, (s_m, t_m, T_m)>, t_{walking} is the smallest value of time spent per unit length in this shopping transaction path, t_j and T_j (j = 1,…,m) are defined as below:

$$t_j = t_{\text{walking}} + \sum_{i=1}^{3} t_{\text{purchasing}}(j,i) \quad (j = 1,\dots,m) \tag{7}$$

$$t_{\text{purchasing}}(j,i) = t_{j,i} - t_{\text{walking}} \quad (i=1,2,3,\ j = 1,\dots,m) \tag{8}$$

$$T_j = \bigcup_{i=1}^{3} T_{j,i} \quad (j = 1,\dots,m) \tag{9}$$

For example, for the shopping path <AB, BC, CD, DE, ED, DC, CD, DE, EF> shown in Figure 7, there is a palindrome-contained pattern <CD, DE>→<ED, DC>→<CD, DE>, so this shopping path can be simplified as a mainstream of shopping path <AB, BC, CD, DE, EF>.

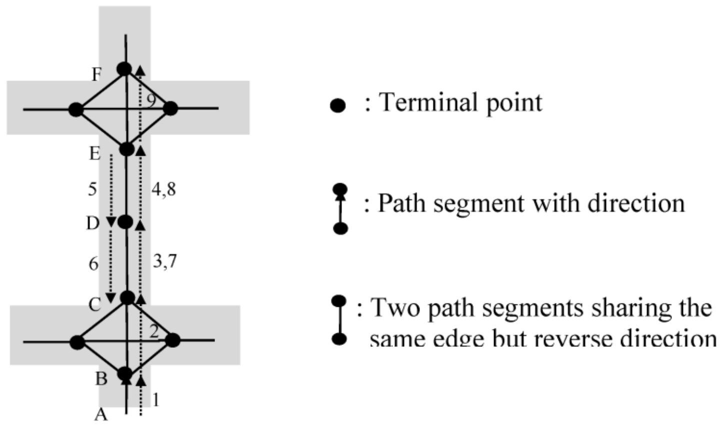

F
9
E
5
4,8
D
6
3,7
C
2
B
1
A
● : Terminal point
: Path segment with direction
: Two path segments sharing the same edge but reverse direction

Figure 7. An illustrative example for palindrome-contained pattern.

In a shopping transaction path <(AB, 1, Ø), (BC, 1, Ø), (CD, 3, Ø), (DE, 2, { i_1}), (ED, 4, {i_2, i_3}), (DC, 6, { i_4, i_5, i_6}), (CD, 5, {i_7}), (DE, 1, Ø), (EF, 1, Ø)>, a palindrome-contained pattern <CD, DE>→<ED, DC>→<CD, DE> appears. Triple elements sharing the same edge (without considering the direction of path segments) in this palindrome-contained pattern can be combined, *i.e.*, (CD, 3, Ø), (DC, 6, { i_4, i_5, i_6}) and (CD, 5, {i_7}) can be combined into a single triple, and (DE, 2, { i_1}), (ED, 4, {i_2, i_3}) and (DE, 1, Ø) also can be merged into one. Take the combination process of (CD, 3, Ø), (DC, 6, {i_4, i_5, i_6}) and (CD, 5, {i_7}) as an example. First, we know that t_{walking} is 1, which equals the smallest of time spent per unit length in this shopping transaction path. t_{CD} is computed as 1 + (3 − 1) + (6 − 1) + (5 − 1) = 12, and T_{CD} is Ø∪ {i_4, i_5, i_6}∪ {i_7} = {i_4, i_5, i_6, i_7}, so the result of the merging is (CD, 12, {i_4, i_5, i_6, i_7}). Thus, this shopping transaction path can be simplified as <(AB, 1, Ø), (BC, 1, Ø), (CD, 12, {i_4, i_5, i_6, i_7}), (DE, 5, {i_1, i_2, i_3}), (EF, 1, Ø) >.

5. Algorithm for Identifying Mainstreams of Shopping Transaction Paths

The algorithm is an iterative process of filtering loop repeat patterns (*i.e.*, Function LRP_Filtering(STP)) and palindrome-contained patterns (*i.e.*, Function PCP_Filtering(STP)) for identifying mainstreams of shopping transaction path from shopping transaction paths, as shown in Algorithm 1.

Algorithm 1 Identifying mainstreams of shopping transaction paths
Input: Shopping transaction paths D_{STP}
Output: Mainstreams of shopping transaction paths D_{MSTP}
Method:

```
1. For each shopping transaction path STP in D_STP do {
2.   while (loop repeat patterns and palindrome-contained patterns exist in STP) do {
3.     Call LRP_Filtering(STP) to filter loop repeat patterns
4.     Call PCP_Filtering(STP) to filter palindrome-contained patterns }
5.   Add STP to D_MSTP }
6. Return D_MSTP.
```

5.1. Function LRP_Filtering(STP)

Given a shopping transaction path STP = <(s_0, t_0, T_0), (s_1, t_1, T_1),…, (s_n, t_n, T_n)>, the process of filtering loop repeat patterns is an iterative procedure to find the start position of the loop (say μ) and the number of path segments in the loop (say λ), such that $s_{\mu+i} = s_{\mu+\lambda+i}$ ($i = 0,1,\ldots, \lambda - 1$; $\mu+\lambda+i \le n$). If μ and λ satisfying the above conditions are found, the fragments <(s_μ, t_μ, T_μ),…, ($s_{\mu+\lambda-1}$, $t_{\mu+\lambda-1}$, $T_{\mu+\lambda-1}$)> and <($s_{\mu+\lambda}$, $t_{\mu+\lambda}$, $T_{\mu+\lambda}$),…, ($s_{\mu+2\lambda-1}$, $t_{\mu+2\lambda-1}$, $T_{\mu+2\lambda-1}$)> in STP form a loop repeat pattern. This procedure is given in sub-function Find_RepeatLoops(*STP*), and three data structures (*i.e.*, path vector, hash table, and list of loop candidates) are used:

Definition 14. A *path vector* (say *PV*) is a vector of pair (*s*, *pos*), where *s* is a path segment and *pos* is the previous position of *s* in *PV*. A *hash table* (say *HT*) stores the current position (*i.e.*, *cur_pos_seg*) for each path segment (*i.e.*, *s*), and a hash function *f* is defined in *HT*, such that $HT[f(s)] = cur\_pos\_seg$. A *list of loop candidates* (say *List*) is a list of triple (*b_pos*, *e_pos*, *cur_pos*) representing a loop candidate (*i.e.*, the fragment <*PV*[*b_pos*].*s*, *PV*[*b_pos*+1].*s*, …, *PV*[*e_pos*].*s*>), where *cur_pos* is the current matching position between *b_pos* and *e_pos*. Based on these definitions, we have the following Function LRP_Filtering(*STP*), where the key part is the sub-function Find_RepeatLoops(*SP*).

Function. LRP_Filtering(*STP*)
Method:

```
1. While repeat loop pattern is found, do{
2.   (μ, λ, n_loops) ← Find_RepeatLoops(trans(STP))
3.   If repeat loop pattern is found, do {
4.     For STP, combine n_loops fragments, i.e., <(s_μ, t_μ, T_μ),…, (s_{μ+λ-1}, t_{μ+λ-1}, T_{μ+λ-1})>, <(s_{μ+λ}, t_{μ+λ}, T_{μ+λ}),…,
       (s_{μ+2λ-1}, t_{μ+2λ-1}, T_{μ+2λ-1})>, …, <(s_{μ+(n_loops-1)×λ}, t_{μ+(n_loops-1)×λ}, T_{μ+(n_loops-1)×λ}),…, (s_{μ+n_loops×λ-1}, t_{μ+n_loops×λ-1},
       T_{μ+n_loops×λ-1})>, to form a new STP according to Definition 12.} }
5. Return STP
```

Sub-Function. Find_RepeatLoops(*SP*)

```
1.  Initialize PV, HT and List as empty
2.  Suppose $SP = <s_0, s_1, \ldots, s_n>$. For each path segment $s_i$ in SP, do {
3.      If $s_i$ is a key in HT, let cur_pos_seg = HT[f($s_i$)]; otherwise, insert a key-value pair ($s_i$, "null") to HT and
        let cur_pos_seg="null". Push the pair ($s_i$, cur_pos_seg) onto PV, and get the position of this pair in PV
        (say new_cur_pos_seg). Set HT[f($s_i$)] to be new_cur_pos_seg in HT.
4.      For each triple (b_pos, e_pos, cur_pos) in List, do {
5.          If PV[cur_pos+1].s equals to $s_i$, do {
6.              If (cur_pos+1) equals to e_pos, repeat loops are found. Let $\mu$ be b_pos, $\lambda$ be e_pos-b_pos+1. Call
                n_loops $\leftarrow$ Test_RepeatLoops(SP, $\mu$, $\lambda$, 2). Return the triple ($\mu$, $\lambda$, n_loops) and exit this sub-
                function. Otherwise, let cur_pos++ and update this triple in List. }
7.          Else delete this triple from List. }
8.      If cur_pos_seg equals to new_cur_pos_seg-1, repeat loops are found. Let $\mu$ be cur_pos_seg and $\lambda$ be 1.
        Call n_loops $\leftarrow$ Test_RepeatLoops(SP, $\mu$, $\lambda$, 2). Return the triple ($\mu$, $\lambda$, n_loops) and exit this sub-
        function.
9.      While cur_pos_seg isn't "null", do {
10.         Generate a candidate triple (cur_pos_seg, new_cur_pos_seg-1, cur_pos_seg) and add it to List. Let
            cur_pos_seg = PV[cur_pos_seg].pos.}}
11. No repeat loop pattern is found. Return the triple ("null", "null", "null").
Sub-Function. Test_RepeatLoops(SP, $\mu$, $\lambda$, n_loops)
1.  If $\mu + (n\_loops + 1) \times \lambda - 1 \leq n$ and $<s_{\mu}, s_{\mu+1}, \ldots, s_{\mu+\lambda-1}> = <s_{\mu+n\_loops\times\lambda}, s_{\mu+n\_loops\times\lambda+1}, \ldots, s_{\mu+(n\_loops+1)\times\lambda-1}>$,
    n_loops++ and then call n_loops $\leftarrow$ Test_RepeatLoops(SP, $\mu$, $\lambda$, n_loops).
2.  Return n_loops.
```

We use a running example to explain the running process of sub-function Find_RepeatLoops(SP). Given a shopping path *SP* = <EA, AB, BC, CD, DE, EA, AB, BE, EA, AB, BE, EG, GD, DK>, the function reads path segments in SP one by one, and the process of finding loop repeat pattern is shown below.

For the first path segment EA, EA cannot be found as a key in empty *HT*, so the key-value pair (EA, "null") is inserted to *HT*, and the pair (EA, "null") is pushed onto an empty *PV*. The position of this pair in *PV* is 0. Therefore, the value associated with EA is changed to 0 in *HT*. *cur_pos_seg* (which is "null") doesn't equal to *new_cur_pos_seg*-1 (which is -1). *List* still remains empty.

For the second path segment AB, similar operations are done. After operations, the key-value pair (AB, 1) is added in *HT*, and the pair (AB, "null") is pushed onto *PV*.

Similarly, after reading the third path segment BC, the fourth CD, and the fifth DE, pairs (BC, 2), (CD, 3), (DE, 4) are inserted into *HT*, and pairs (BC, "null"), (CD, "null"), (DE, "null") are pushed onto *PV* sequentially. Because *cur_pos_seg* keeps "null", no candidate is generated.

For the sixth path segment EA, it is found as a key in *HT* and *cur_pos_seg* is 0, which is the position of previous EA in *PV*. Push the pair (EA, 0) onto *PV* and set the value associated with EA (*i.e.*, *HT*[*f*(EA)]) to be 5 in *HT*. Because *cur_pos_seg* is not "null", a candidate (0, 4, 0) is generated and added to *List*. And then, *cur_pos_seg* = *PV*[0].*pos* = "null", so no candidate is generated here.

For the seventh path segment AB, the pair (AB, 1) is pushed onto *PV* and *HT*[*f*(AB)] is set as 6. Since there is a loop candidate (*i.e.*, triple (0, 4, 0)) in *List*, we need to compare AB with the next path segment of this candidate (*i.e.*, *PV*[0+1].*s*). Both of them are AB and they are matching, so we set this triple to be (0, 4, 1). Because *cur_pos_seg* is 1, a new candidate (1, 5, 1) is produced and there are two candidates in *List* now.

When reading the eighth one BE, the pair (BE, 7) is pushed onto *PV* and *HT*[*f*(BE)] is 7. For the candidate (0, 4, 1), the next path segment is *PV*[1+1].*s*=BC, which does not match BE. So this candidate should be deleted from *List*. For the candidate (1, 5, 1), the next path segment, *i.e.*, *PV*[1+1].*s*=BC, does not match BE, so this candidate also needs to be pruned. No new candidate is generated, since *cur_pos_seg* equals to "null".

For the ninth one EA, the pair (EA, 5) is added at the end of *PV* and the value associated with EA is set as 8 in *HT*. Similarly, two new candidates (5, 7, 5) and (0, 7, 0) are obtained and added to *List*.

Similarly, when reading the tenth path segment AB, the pair (AB, 6) is pushed onto *PV* and *HT*[*f*(AB)] is set as 9. For candidates, triple (5, 7, 5) becomes (5, 7, 6), and triple (0, 7, 0) is converted to (0, 7, 1). And two new candidates, *i.e.*, triple (6, 8, 6) and triple (1, 8, 1), are generated.

When reading the eleventh one BE, the pair (BE, 7) is added at the end of *PV* and *HT*[*f*(BE)] is set as 10. For the candidate (5, 7, 6), the next path segment *PV*[6+1].*s* equals to BE, and is the last one in this candidate. Thus repeat loops are found, and μ, λ are 5, 3 respectively. And then, we call Test_RepeatLoops(*SP*, 5, 3, 2). Since $<s_5, s_6, s_7>$ (*i.e.*, <EA, AB, BE>) are not equal to $<s_{11}, s_{12}, s_{13}>$ (*i.e.*, <EG, GD, DK>), *n_loops* which equals 2 is returned. Thus, we have *n_loops* equals 2.

5.2. Function PCP_Filtering (STP)

Given a shopping transaction path $STP = <(s_0, t_0, T_0), (s_1, t_1, T_1), \ldots, (s_n, t_n, T_n)>$, the procedure of filtering palindrome-contained patterns (*i.e.*, $STP_1 \rightarrow STP_2 \rightarrow STP_3$, where Trans($STP_1$), the reverse-order path of Trans(STP_2), and Trans(STP_3) are equal (say *SP*)) is an iterative process of finding the start position of palindrome-contained pattern (say μ), and the number of path segments of *SP* (say λ), such that $s_{\mu+i} = s_{\mu+2\lambda-i-1,reverse} = s_{\mu+2\lambda+i}$ ($i = 0,1,\ldots, \lambda - 1$; $\mu + 2\lambda + i \leq n$), where $s_{\mu+2\lambda-i-1,reverse}$ is the reverse-order path segment of $s_{\mu+2\lambda-i-1}$. If μ and λ satisfying the above conditions are found, the connections of three fragments $<(s_\mu, t_\mu, T_\mu), \ldots, (s_{\mu+\lambda-1}, t_{\mu+\lambda-1}, T_{\mu+\lambda-1})>$, $<(s_{\mu+\lambda}, t_{\mu+\lambda}, T_{\mu+\lambda}), \ldots, (s_{\mu+2\lambda-1}, t_{\mu+2\lambda-1}, T_{\mu+2\lambda-1})>$ and $<(s_{\mu+2\lambda}, t_{\mu+2\lambda}, T_{\mu+2\lambda}), \ldots, (s_{\mu+3\lambda-1}, t_{\mu+3\lambda-1}, T_{\mu+3\lambda-1})>$ in *STP* form a palindrome-contained pattern. This procedure is presented in the sub-function Find_PCP(*STP*), and three data structures (*i.e.*, vector of path segment, list of candidate, and list of candidate suffix) are adopted.

Definition 15. A *vector of path segment* (say *V*) stores path segments. Suppose a potential palindrome-contained pattern is $SP \rightarrow SP_{reverse} \rightarrow SP$. A *list of candidate* (say *LC*) is a list of triple (*b_pos*, *e_pos*, *cur_pos*), and each triple represents a candidate *SP* (*i.e.*, the fragment <*V*[*b_pos*], *V*[*b_pos*+1], …, *V*[*e_pos*]>), where *cur_pos* is the current matching position between *b_pos* and *e_pos*. A *list of candidate suffix* (say *LCS*) is a list of pair (*inter_posi*, *e_posi*), and each pair represents a candidate suffix of *SP* (*i.e.*, the fragment <*V*[*inter_posi*], *V*[*inter_posi*+1], …, *V*[*e_posi*]>).

Based on the above definition, we have the following Function PCP_Filtering (*STP*):

```
Function. PCP_Filtering(STP)
Method:
1.  While palindrome-contained pattern is found, do{
2.    (μ, λ) ← Find_PCP(trans(STP))
3.    If palindrome-contained pattern is found, do {
4.      For STP, combine fragments <($s_{μ}$, $t_{μ}$, $T_{μ}$),…, ($s_{μ+λ-1}$, $t_{μ+λ-1}$, $T_{μ+λ-1}$)>, <($s_{μ+λ}$, $t_{μ+λ}$, $T_{μ+λ}$),…, ($s_{μ+2λ-1}$,
        $t_{μ+2λ-1}$, $T_{μ+2λ-1}$)> and <($s_{μ+2λ}$, $t_{μ+2λ}$, $T_{μ+2λ}$),…, ($s_{μ+3λ-1}$, $t_{μ+3λ-1}$, $T_{μ+3λ-1}$)> to form a new STP according to
        Definition 13.} }
5.  Return STP
Sub-Function. Find_PCP(SP)
1.  Initialize V, LC and LCS as empty
2.  Suppose SP = <$s_0$, $s_1$, …, $s_n$>. For each path segment $s_i$ in SP, do {
3.    If V is not empty, do {
4.      Get the position of the last element of V (say cur_pos_seg), and compare the last element of V
        with $s_i$. If they are reverse-order, let variable reverse-order be true. Otherwise, let variable reverse-
        order be false. }
5.    Else let reverse-order be false.
6.    Push $s_i$ onto V.
7.    For each candidate triple (b_pos, e_pos, cur_pos) in LC, do {
8.      If cur_pos is “null”, let cur_pos be b_pos; otherwise, let cur_pos be added by 1.
9.      Compare V[cur_pos] with $s_i$. If they are same, do {
10.       Let this triple be (b_pos, e_pos, cur_pos) and update it in LC. If cur_pos equals to e_pos, then a
          palindrome-contained pattern is found. Let μ be b_pos, λ be e_pos-b_pos+1, return the pair of μ
          and λ, and exit this sub-function. }
11.     Else delete this candidate triple from LC. }
12.   For each pair (inter_posi, e_posi) in LCS, do {
13.     If inter_posi-1 ≥ 0, and V[inter_posi-1] and $s_i$ are reverse-order, set this pair as (inter_posi-1,
        e_posi) in LCS, and generate a new candidate (inter_posi-1, e_posi, “null”) in LC. Otherwise,
        delete this pair from LCS. }
14.   If reverse-order is true, do {
15.     Produce a candidate (cur_pos_seg, cur_pos_seg, “null”) and insert it to LC.
16.     Generate a candidate suffix (cur_pos_seg, cur_pos_seg) and add it to LCS. } }
17. No palindrome-contained pattern is found. Return the pair of “null” and “null”.
```

In the following, we illustrate a running example to show the running procedure of sub-function Find_PCP(SP). For instance, given a shopping path SP = <AB, BC, CD, DE, EF, FE, ED, DC, CD, DE, EF, FG>, its process of finding palindrome-contained pattern is as described below:

For the first path segment AB, simply let *reverse-order* be false and push AB onto *V*. When reading the second path segment BC, we compare the last element of *V*, which is AB, with BC. Because they are not reverse-order, *reverse-order* is false. Then, we push BC onto *V*.

Similarly, we push CD, DE, EF onto *V*. And no candidate or candidate suffix is generated.

For the sixth path segment FE, since the last element of *V* (EF) and FE are reverse-order, *reverse-order* is true. We push FE onto *V*. A candidate (4, 4, “null”) and a candidate suffix (4, 4) are generated.

When reading the seventh path segment ED, *reverse-order* is false, and ED is pushed onto *V*. For candidate (4, 4, “null”), we compare *V*[4] (EF) with ED and they do not match, so we delete this candidate from *LC*. For candidate suffix (4, 4), since *V*[3] (DE) is reverse-order path segment of ED, this candidate suffix becomes (3, 4) and a new candidate (3, 4, “null”) is generated.

When reading the eighth path segment DC, *reverse-order* is also false, and DC is pushed onto *V*. For candidate (3, 4, “null”), since *V*[3] (DE) and DC do not match we also prune this candidate from *LC*. For candidate suffix (3, 4), *V*[2] (CD) and DC are reverse-order, so this candidate suffix turns to (2, 4) and a new candidate (2, 4, “null”) is produced.

When reading the ninth path segment CD, *reverse-order* is true and CD is pushed onto *V*. For candidate (2, 4, “null”), since *V*[2] (CD) and CD match, this candidate turns to (2, 4, 2). For candidate suffix (2, 4), since *V*[1] (BC) and CD are not reverse-order, we delete this candidate suffix from *LCS*. Because *reverse-order* is true, a candidate (7, 7, “null”) and a candidate suffix (7, 7) are produced.

When reading the tenth one DE, *reverse-order* is false and DE is pushed onto *V*. For candidate (2, 4, 2), since *V*[3] (DE) and DE are matching, it turns to (2, 4, 3). For candidate (7, 7, “null”), it is deleted for mismatch. For candidate suffix (7, 7), since *V*[6] (ED) and DE are reverse-order, this candidate suffix becomes (6, 7) and a new candidate (6, 7, “null”) is added to *LC*.

Then for the eleventh one EF, *reverse-order* is also false and EF is pushed onto *V*. For candidate (2, 4, 3), since *V*[4] (EF) and EF are matching, this candidate becomes (2, 4, 4). Thus, a palindrome-contained pattern is found, and μ, λ are 2, 3 respectively.

6. Generation of Synthetic Shopping Transaction Paths

In order to generate a synthetic workload, we build an agent [41]-based simulator to simulate the scenario of an individual shopping trip. The complete flow diagram for this simulator is shown in Figure 8, which mainly includes four steps: construction of a path graph, initialization of customer agents, generating a shopping transaction path, and attaching extra loop repeat patterns and palindrome-contained patterns. Among them, Step 4 is optional for testing. Steps 2, 3 and 4 can be performed repeatedly for $|D|$ times and then a database of shopping transaction paths *D* will be produced. In the following, we discuss these four steps in detail. For the sake of easy reference, the meanings of various variables used in our simulator are summarized in Table 4.

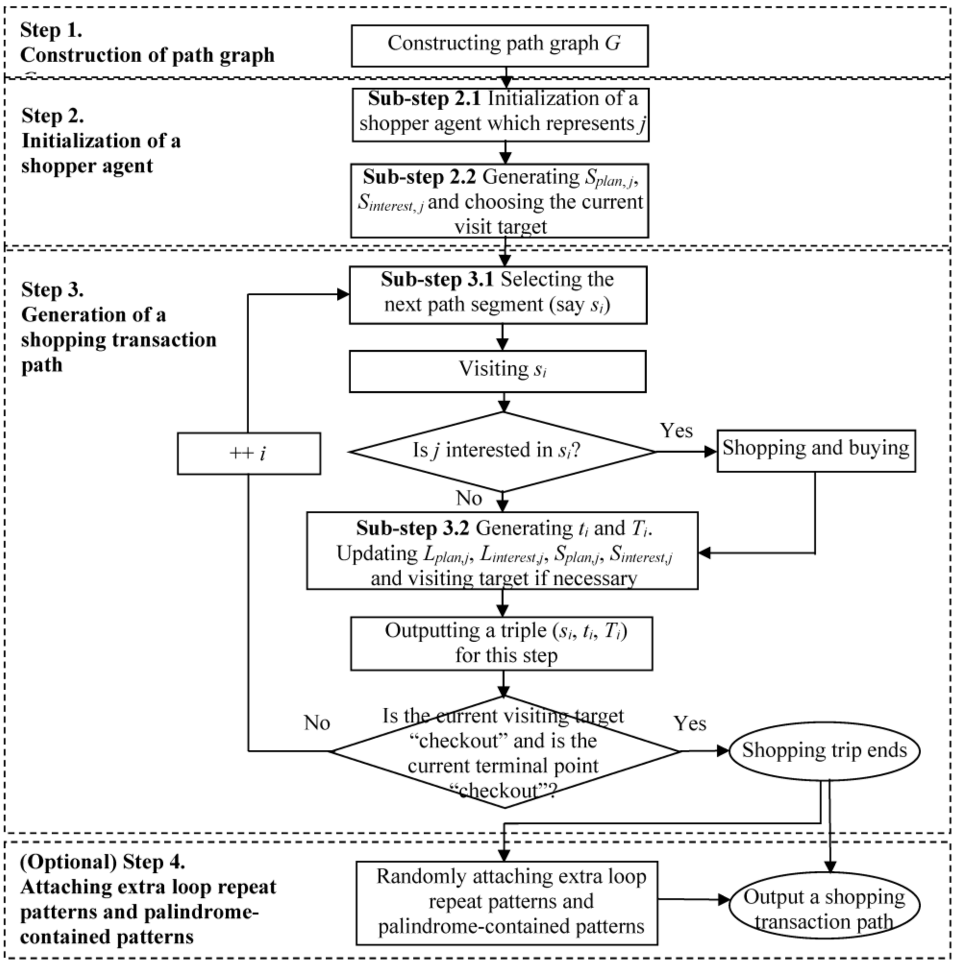

Step 1.
Construction of path graph
Constructing path graph G
Step 2.
Initialization of a shopper agent
Sub-step 2.1 Initialization of a shopper agent which represents j
Sub-step 2.2 Generating S_plan,j, S_interest,j and choosing the current visit target
Step 3.
Generation of a shopping transaction path
Sub-step 3.1 Selecting the next path segment (say s_i)
Visiting s_i
Is j interested in s_i?
Yes
Shopping and buying
No
++ i
Sub-step 3.2 Generating t_i and T_i. Updating L_plan,j, L_interest,j, S_plan,j, S_interest,j and visiting target if necessary
Outputting a triple (s_i, t_i, T_i) for this step
Is the current visiting target "checkout" and is the current terminal point "checkout"?
No
Yes
Shopping trip ends
(Optional) Step 4.
Attaching extra loop repeat patterns and palindrome-contained patterns
Randomly attaching extra loop repeat patterns and palindrome-contained patterns
Output a shopping transaction path

Figure 8. Flow diagram for generating a shopping transaction path.

Table 4. Meaning of various variables in our simulations.

Notation	**Description**
$n_{terminal\_points}$, $n_{path\_segments}$	The number of terminal points, path segments in path graph *G* respectively
n_{items}	The number of different items
$ShoppingTime(i_{item})$	The shopping time for i_{item}
j	A shopper
$speed_{normal,\,j}$, $speed_{\,j}$	The normal, actual moving speed for *j* respectively
$n_{plan,\,j}$	The number of different planned-purchasing items for *j*
$n_{interest,\,j}$	The number of different items that *j* feels interested in
$L_{plan,\,j}$	A set of planned-purchasing items for *j*
$L_{interest,\,j}$	A set of items that *j* feels interested in
$S_{plan,\,j}$	A set of path segments that *j* plans to visit
$S_{interest,\,j}$	A set of path segments that *j* feels interested in
μ_{speed_normal}, σ_{speed_normal}	The mean, the standard deviation of the Gaussian distribution of $speed_{normal,j}$
$lower\_bound_{n_{plan,j}}$, $upper\_bound_{n_{plan,j}}$	The lower bound, the upper bound of the uniform distribution of $n_{plan,\,j}$ on integers respectively
$n_{extra\_interest,\,j}$	The number of additional items (besides items in $L_{plan,\,j}$) that *j* feels interests in

Table 4. *Cont.*

Notation	Description
$lower\_bound_{n_{extra\_interest, j}}$, $upper\_bound_{n_{extra\_interest, j}}$	The lower bound, the upper bound of the uniform distribution of $n_{extra\_interest,j}$ on integers respectively
$\mu_{ShoppingTime}(i_{item,k})$, $\sigma_{ShoppingTime}(i_{item,k})$	The mean, the standard deviation of the Gaussian distribution of $ShoppingTime(i_{item,k})$ respectively
$PerceivedTimePressure_j$	The perceived time pressure for j
$\sigma_{PerceivedTimePressure}$	The standard deviation of the Gaussian distribution of $PerceivedTimePressure_j$
$time_{s, j}$	Time spent in path segment s for j
$time_{walking, s, j}$, $time_{shopping, s, j}$	Time spent for walking, shopping in path segment s for j respectively
$Distance(j, s)$	The distance between j and path segment s
$\lvert D\rvert$	The number of shopping transaction paths in D
$\bar{L}$	The average number of path segments in shopping transaction paths
n_{LRP}, n_{PCP}	The number of loop repeat patterns, palindrome-contained patterns respectively

6.1. Step 1. Construction of Path Graph G

A path graph, the container for customer agents moving in, is constructed in this step. We need to specify the components of a path graph: the set of terminal points of path segments, and the set of path segments. A length table *LT* and a segment-item table *SIT* are also needed to be produced. And then, an item-segment table *IST* can be derived.

6.2. Step 2. Initialization of a Shopper Agent

This step includes the following two sub-steps.

6.2.1. Shopper Agent Initialization

In this sub-step, a shopper agent representing an in-store shopper (say *j*) is initialized. Each shopper agent has the following parameters, which need to be specified:

(1) Normal moving speed $speed_{normal, j}$

$speed_{normal, j}$ means the normal moving speed of *j*, which is derived from a Gaussian distribution with mean μ_{speed_normal} and standard deviation σ_{speed_normal}.

(2) Number of different planned-purchasing items $n_{plan, j}$

$n_{plan, j}$ represents the number of different items that are planned to be purchased by *j*, and is derived from an uniform distribution on the integers $lower\_bound_{n_{plan, j}}$, $lower\_bound_{n_{plan, j}} + 1$, …, $upper\_bound_{n_{plan, j}}$.

(3) A set of planned-purchasing items $L_{plan, j}$

$L_{plan,j}$ means the set of different items that are planned to be purchased, and can be written as $\{i_{item,1}, i_{item,2}, \ldots, i_{item,n_{plan,j}}\}$.

(4) Number of different items that *j* feels interested in (say $n_{interest,j}$)

$n_{interest,j}$ is the sum of $n_{plan,j}$ and the number of additional items (besides items in $L_{plan,j}$) that *j* feels interests in (say $n_{extra\_interest,j}$). The latter is derived from an uniform distribution on the integers $lower\_bound_{n_{extra\_interest,j}}$, $lower\_bound_{n_{extra\_interest,j}} + 1, \ldots, upper\_bound_{n_{extra\_interest,j}}$.

(5) A set of items that *j* feels interested in (say $L_{interest,j}$)

$L_{interest,j}$ has the form $\{i_{item,1}, i_{item,2}, \ldots, i_{item,n_{interest,j}}\}$, and each item $i_{item,k}$ is associated with its shopping time $ShoppingTime(i_{item,k})$ ($k = 1, 2, \ldots, n_{interest,j}$). $ShoppingTime(i_{item,k})$ is derived from a Gaussian distribution with mean $\mu_{ShoppingTime}(i_{item,k})$ and standard deviation $\sigma_{ShoppingTime}(i_{item,k})$.

(6) Perceived time pressure [42] $PerceivedTimePressure_j$

$PerceivedTimePressure_j$ represents *j*'s perceived time pressure during a shopping trip, and significantly affects *j*'s moving speed. $PerceivedTimePressure_j$ is also derived from a Gaussian distribution with mean 1 and standard deviation $\sigma_{PerceivedTimePressure}$.

For *j*, his/her actual moving speed $speed_j$ can be simply computed as below:

$$speed_j = PerceivedTimePressure_j \times speed_{normal,j}. \tag{10}$$

6.2.2. Generating $S_{plan,j}$ and $S_{interest,j}$, and Choosing the Current Visit Target

In order to decide which direction a shopper agent would like to go, we need to know which path segments *j* plans to visit. These path segments are visit targets for *j*, and *j* will visit these path segments one by one. Here we use $S_{plan,j}$ to represent the set of path segments that *j* plans to visit, and use $S_{interest,j}$ to represent the set of path segments that *j* feels interested in. According to the item-segment table *IST*, we can derive $S_{plan,j}$ by mapping each item in $L_{plan,j}$ to path segments where this item is sold. Similarly, $S_{interest,j}$ also can be derived according to *IST* and $L_{interest,j}$.

Definition 16. Given a path graph *G*, suppose a shopper *j* is at terminal point *v*, and the start terminal point and the end terminal point of path segment *s* are *s.b* and *s.e* respectively. Then the distance between the shopper *j* and the segment *s* is defined as below:

$$Distance(j, s) = \min(shortest\_path\_length(j, s.b), shortest\_path\_length(j, s.e)) \tag{11}$$

where function $shortest\_path\_length(\cdot, \bullet)$ means length of the shortest path between two terminal points in *G*, and function $\min(\cdot, \bullet)$ represents the minimal one of two values. In the above definition, length of the shortest path between two terminal points in *G* can be obtained using well-known Dijkstra's algorithm [43,44]. Thus, based on the definition of distance between a shopper and a path segment, we simply use the following method to decide the current visit target.

Method 1 (deciding the visiting target). For a shopper *j*, among the elements of $S_{plan,j}$, the nearest path segment is regarded as the current visit target. If $S_{plan,j}$ is empty, "checkout" becomes the moving target. The current visit target remains unchanged until the current visit target has been visited and $S_{plan,j}$ is updated.

6.3. Step 3. Generation of a Shopping Transaction Path

The production of a shopping transaction path can be regarded as a repetitive process of deciding which path segment s_i ($i = 1,2,\ldots,n$) should be chosen as the next step, and generating unit time per unit length spent in s_i (say t_i) and the itemset purchased in s_i (say T_i).

6.3.1. Decision on the Next Path Segment

For simplicity, we suppose the walking process of a shopper *j* is as follows: first, *j* selects a visit target, and then he/she walks along the shortest path to the visit target. When he/she reaches the current visit target, he/she needs to decide the next visit target. The process is repeated, until he/she finishes his/her shopping and arrives at "checkout". Thus, we have the following method for deciding the next path segment.

Method 2 (deciding the next path segment). Given a path graph *G*, if a shopper *j* hasn't reached the current visit target, the next section along the shortest path to the current visit target is selected as the next path segment for *j*. If *j* arrives at the current visit target, he/she considers and decides the next visit target. Then, the next path segment along the shortest path to the next visit target is chosen as the next section.

In this method, the shortest path to a visit target can be obtained by popular Dijkstra's algorithm [43,44].

6.3.2. Sub-Step 3.2 Generating t_i and T_i, and Updating $L_{plan,j}$, $L_{interest,j}$, $S_{plan,j}$, $S_{interest,j}$ and Visiting Target

(1) Generating t_i and T_i

For a shopper *j*, t_i is the quotient of time spent in s_i (say $time_{s_i,j}$) divided by the length of s_i (say $s_i.l$). $time_{s_i,j}$ consists two parts: time spent for walking in s_i (say $time_{walking,s_i,j}$) and time spent for shopping in s_i (say $time_{shopping,s_i,j}$). $time_{walking,s_i,j}$ can be computed as below:

$$time_{walking,s_i,j} = s_i.l/speed_j = s_i.l/(PerceivedTimePressure_j \times speed_{normal,j}) \tag{12}$$

where $speed_j$ is *j*'s actual moving speed.

For simplicity, the value of $time_{shopping,s_i,j}$ depends on whether *j* feels interested in s_i (that is $s_i \in S_{interest,j}$) or not, and is computed as below:

$$time_{shopping,s_i,j} = \begin{cases} \sum\limits_{\forall i_{item},\, i_{item} \in L_{interest,j} \wedge i_{item} \in \Gamma_{s_i}} ShoppingTime(i_{item}) & \text{if } s_i \in s_{interest,j} \\ 0 & \text{if } s_i \notin s_{interest,j} \end{cases} \tag{13}$$

where Γ_{s_i} is the itemset sold in s_i and can be obtained from *SIT*.

For s_i, T_i simply equals to the set of items that belong to both $L_{plan,j}$ and Γ_{s_i}.

(2) Updating $L_{plan,j}$, $L_{interest,j}$, $S_{plan,j}$, $S_{interest,j}$ and visiting target

If $s_i \notin S_{interest,j}$, nothing needs to be updated. Otherwise, since s_i has been visited, it should be deleted from $S_{interest,j}$. For the reason that T_i has been purchased at s_i, items in T_i need to be removed from $L_{plan,j}$ and $L_{interest}$. If $s_i \in S_{plan,j}$, s_i also should be pruned from $S_{plan,j}$, and the visiting target should be updated further using Method 1, which is given in Section 6.2.2.

6.4. Step 4. Attaching Extra Loop Repeat Patterns and Palindrome-Contained Patterns

Producing extra loop repeat patterns and palindrome-contained patterns are exactly the reverse processes of simplifying these two patterns which are presented in Definitions 12 and 13. The methods for producing a loop repeat pattern and a palindrome-contained pattern are described below:

Method 3 (producing a loop repeat pattern). A shopping transaction path, *i.e.*, $STP_{prefix} \rightarrow STP_{combine} \rightarrow STP_{suffix}$, can be transformed to any $STP_{prefix} \rightarrow STP_1 \rightarrow STP_2 \rightarrow \ldots \rightarrow STP_n \rightarrow STP_{suffix}$, where $STP_{combine} = <(s_1, t_1, T_1), (s_2, t_2, T_2), \ldots, (s_\lambda, t_\lambda, T_\lambda)>$, $STP_i = <(s_1, t_{1,i}, T_{1,i}), (s_2, t_{2,i}, T_{2,i}), \ldots, (s_\lambda, t_{\lambda,i}, T_{\lambda,i})>$, $STP_{combine}$ and all STP_i ($i = 1,\ldots,n$) share the same navigation pattern (say Trans(TSP) = $<s_1, s_2, \ldots, s_\lambda>$), only if the following equations are satisfied:

$$t_j = t_{\text{walking}} + \sum_{i=1}^{n} t_{\text{purchasing}}(j,i) \quad (j = 1,\ldots,\lambda) \tag{14}$$

$$t_{\text{purchasing}}(j,i) = t_{j,i} - t_{\text{walking}} \quad (i = 1,\ldots,n, j = 1,\ldots,\lambda) \tag{15}$$

$$T_j = \bigcup_{i=1}^{m} T_{j,i} \quad (j = 1,\ldots,\lambda) \tag{16}$$

where t_{walking} is the smallest value of time spent per unit length in this shopping transaction path.

Method 4 (producing a palindrome-contained pattern). A shopping transaction path, *i.e.*, $STP_{prefix} \rightarrow STP_{combine} \rightarrow STP_{suffix}$, can be transformed to any $STP_{prefix} \rightarrow STP_1 \rightarrow STP_2 \rightarrow STP_3 \rightarrow STP_{suffix}$, where $STP_{combine} = <(s_1, t_1, T_1), (s_2, t_2, T_2), \ldots, (s_\lambda, t_\lambda, T_\lambda)>$, $STP_i = <(s_1, t_{1,i}, T_{1,i}), (s_2, t_{2,i}, T_{2,i}), \ldots, (s_\lambda, t_{\lambda,i}, T_{\lambda,i})>$ ($i = 1, 3$), $STP_2 = <(s_{\lambda,reverse}, t_{\lambda,2}, T_{\lambda,2}), (s_{\lambda-1,reverse}, t_{\lambda-1,2}, T_{\lambda-1,2}), \ldots, (s_{1,reverse}, t_{1,2}, T_{1,2})>$, only if the following equations are satisfied:

$$t_j = t_{\text{walking}} + \sum_{i=1}^{3} t_{\text{purchasing}}(j,i) \quad (j=1,\ldots,\lambda) \tag{17}$$

$$t_{\text{purchasing}}(j,i) = t_{j,i} - t_{\text{walking}} \quad (i=1,2,3, j=1,\ldots,\lambda) \tag{18}$$

$$T_j = \bigcup_{i=1}^{3} T_{j,i} \quad (j=1,\ldots,\lambda) \tag{19}$$

where t_{walking} is the smallest value of time spent per unit length in this shopping transaction path. Therefore, in order to produce a loop repeat pattern or palindrome-contained pattern, firstly, we randomly choose a fragment of shopping transaction path as $STP_{combine}$. And then, transform $STP_{combine}$ according to Method 3 or Method 4. Multiple loop repeat patterns and palindrome-contained patterns can be produced after executing the above process multiple times.

For a database of shopping transaction paths D, five parameters are introduced here: the number of loop repeat patterns (say n_{LRP}), the number of palindrome-contained patterns (say n_{PCP}), the average number of path segments in $STP_{combine}$ for loop repeat patterns (say $\bar{\lambda}_{LRP}$), the average number of path segments in $STP_{combine}$ for palindrome-contained patterns (say $\bar{\lambda}_{PCP}$), the average repeat times in loop repeat patterns (say $\bar{n}_{repeat}$).

7. Experimental Results

To assess the performance of the algorithm of identifying the mainstream shopping transaction paths and PFNP-forest algorithm, we conducted several experiments on a PC with a 3.00GHz Intel Core™ 2 Duo E8400 CPU (Santa Clara, CA, USA) and 4GB main memory, running Windows 7 Enterprise Edition. All algorithms are implemented using VC++ 2010. In these experiments, we establish a path graph, which has 159 terminal points and 554 path segments, as an example to generate shopping transaction paths. Without specific explanations, Default values of various parameters used in our simulations are summarized in Table 5. Since the kernel parts of identifying mainstreams are Function LRP_Filtering() (which is used for filtering loop repeat patterns) and Function PCP_Filtering() (which is for filtering palindrome-contained patterns), we test the performance of these two functions.

Table 5. Default values of various parameters used in our simulations.

Parameter	Value	Parameter	Value	Parameter	Value
$n_{terminal\_points}$	159	$lower\_bound_{n_{plan,j}}$	1	$\|D\|$	1000
$n_{path\_segments}$	554	$upper\_bound_{n_{plan,j}}$	20	$\bar{n}_{repeat}$	3
n_{items}	3000	$lower\_bound_{n_{extra\_interest,j}}$	0	n_{LRP}	1000
μ_{speed_normal}	0.6	$upper\_bound_{n_{extra\_interest,j}}$	8	n_{PCP}	1000
σ_{speed_normal}	0.1	$\mu_{ShoppingTime}(i_{item,k})$	1	$\bar{\lambda}_{LRP}$	3
$\sigma_{PerceivedTimePressure}$	0.1	$\sigma_{ShoppingTime}(i_{item,k})$	0.5	$\bar{\lambda}_{PCP}$	3

7.1. Variations of n_{LRP} (or n_{PCP})

Firstly, variations of different execution times with different values of n_{LRP} (or n_{PCP}) are evaluated and compared. We run the simulation (without Step 4) 1000 times and obtain an original database of shopping transaction paths with $|D|$ = 1000 and $\bar{L}$ = 57.4. And then, extra 1000, 2000, 3000, 4000, 5000 and 6000 loop repeat patterns (or palindrome-contained patterns) are attached to the original database with $\bar{\lambda}_{LRP}$ = 3, $\bar{\lambda}_{PCP}$ = 3, and $\bar{n}_{repeat}$ = 3, respectively, using Step 4 in simulation. The execution time of Function LRP_Filtering() and Function PCP_Filtering() in response to different n_{LRP} (or n_{PCP}) are shown in Figure 9.

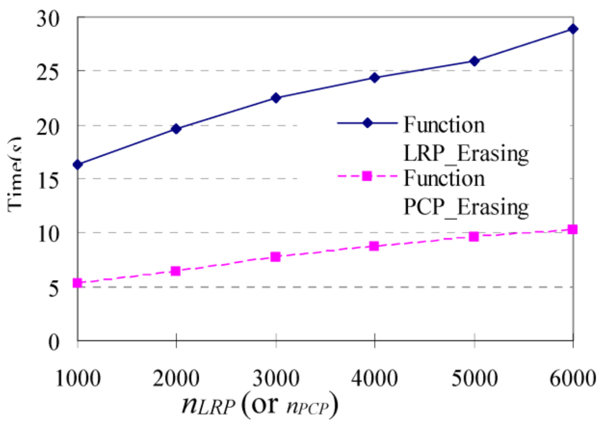


Figure 9. Execution time in response to changes in different n_{LRP} (or n_{PCP}).

We can find that the execution time of Function LRP_Filtering() is about three times that of Function PCP_Filtering() for the same n_{LRP} (or n_{PCP}). Both of them increase linearly with the increase of n_{LRP} (or n_{PCP}) and have a good scalability.

7.2. Variations of $\bar{\lambda}_{LRP}$ (or $\bar{\lambda}_{PCP}$)

Secondly, we test the scalability of these two functions with the increasing of parameter $\bar{\lambda}_{LRP}$ (or $\bar{\lambda}_{PCP}$). Similarly, we obtain the original database of shopping transaction paths with $|D|$ = 1000 and $\bar{L}$ = 57.4 by running the simulation (without Step 4) 1000 times. Then extra 1000 loop repeat patterns (or palindrome-contained patterns), whose $\bar{\lambda}_{LRP}$ (or $\bar{\lambda}_{PCP}$) is 2, 3, 4, 5, 6, 7 and 8, are attached to the original database respectively. The impact of $\bar{\lambda}_{LRP}$ and $\bar{\lambda}_{PCP}$ to the execute time is shown in Figure 10. We can find that both these two functions have a relatively stable execution time with the increase of $\bar{\lambda}_{LRP}$ (or $\bar{\lambda}_{PCP}$).

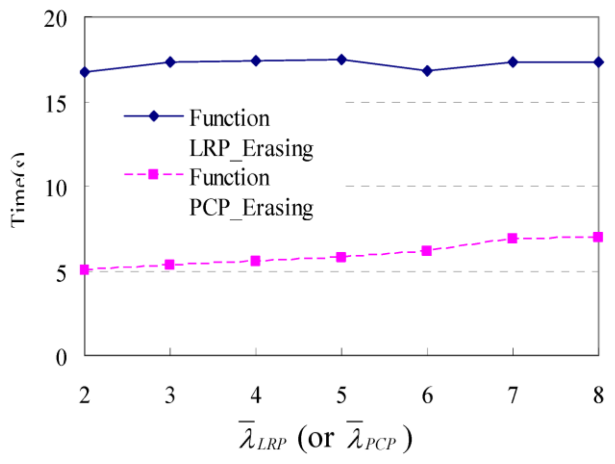


Figure 10. Execution time in response to changes in different $\bar{\lambda}_{LRP}$ (or $\bar{\lambda}_{PCP}$).

7.3. Variations of |D|

Thirdly, the impact of Variations of $|D|$ on the execution time is examined. We run the simulation (without Step 4) 500, 1000, 1500, 2000, 2500, 3000 times, respectively, and obtain the corresponding databases of shopping transaction paths. The values of $\bar{L}$ of these databases are 57.7, 57.4, 57.0, 56.9, 56.9 and 57.0, respectively, which are approximately 57. Then, an extra one loop repeat pattern (or palindrome-contained pattern) with $\bar{\lambda}_{LRP} = \bar{\lambda}_{PCP} = 3$ and $\bar{n}_{repeat} = 3$ is attached to each shopping transaction path of these databases by using Step 4 in the simulator. The experimental results are shown in Figure 11. It is obvious that the execution time of these two functions increases with the increase of $|D|$, and both of them have a good scalability.

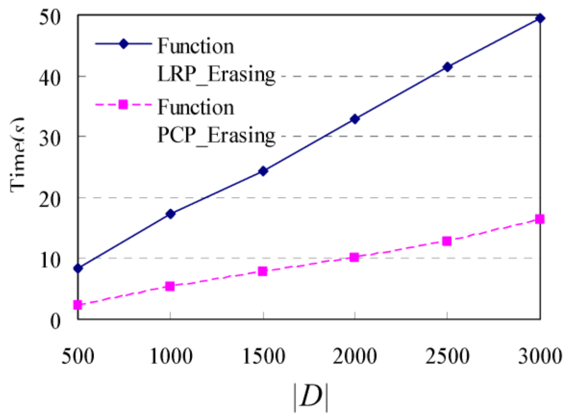


Figure 11. Execution time in response to changes in different |*D*|.

7.4. Variations of $\bar{L}$

The fourth test examines the execution performance of these two functions with varying $\bar{L}$. To obtain different databases of shopping transaction paths with different $\bar{L}$, we set the value of the pair ($lower\_bound_{n_{plan,\,j}}$, $upper\_bound_{n_{plan,\,j}}$) to (1, 1), (1, 8), (1, 18), (5, 25), (12, 30) and (20, 36), and running the simulation (without Step 4) 1000 times respectively. Thus we generate six databases of shopping transaction paths whose $\bar{L}$ is 17.4, 35.7, 53.1, 71.1, 89.5 and 105.0, respectively. The difference between successive values of these $\bar{L}$ is approximately 18. Then, we test Function LRP_Filtering() and Function PCP_Filtering() on these databases, and the experimental results are given in Figure 12. We can find that the execution time of these two functions increases in a linear manner, and both of these two functions show a good scalability with the increase of $\bar{L}$.

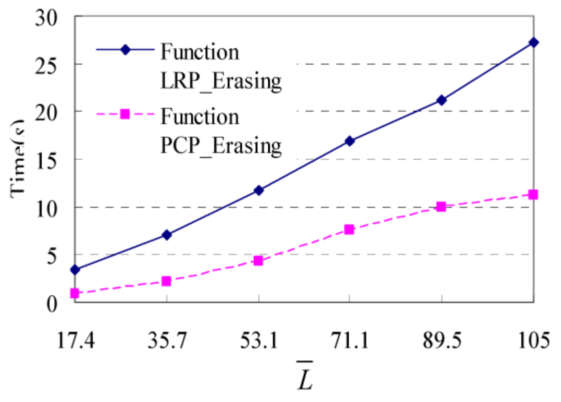


Figure 12. Execution time in response to changes in different $\bar{L}$.

7.5. Variations of $\bar{n}_{repeat}$

Since only Function LRP_Filtering() has the parameter $\bar{n}_{repeat}$, here we test the impact of variations of $\bar{n}_{repeat}$ on the execution time of Function LRP_Filtering(). After obtaining the original database of shopping transaction paths with |*D*| = 1000 and $\bar{L}$ = 57.4, 1000 loop repeat patterns with varying $\bar{n}_{repeat}$ (which are 2, 3, 4, 5, 6) are attached to the original database respectively, and then the corresponding databases with varying $\bar{n}_{repeat}$ are obtained. We test the varying execution time for these databases and show the experimental results in Figure 13. We find that the execution time is almost stable with different $\bar{n}_{repeat}$. This result also can be obtained by analysing Function LRP_Filtering(). Since varying $\bar{n}_{repeat}$ will not change the number of repeat loop patterns that are found, the execution times will not change too much for different $\bar{n}_{repeat}$.

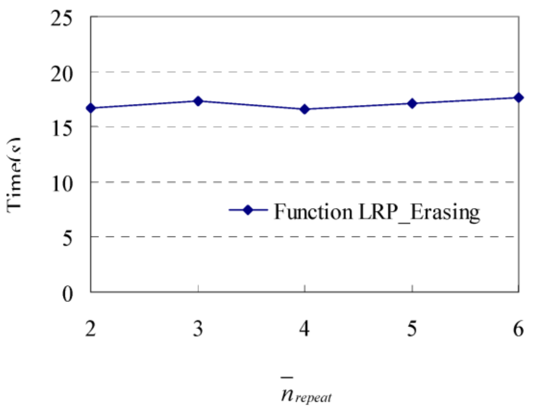


Figure 13. Execution time in response to changes in different $\bar{n}_{repeat}$.

8. Contributions toward a Real Supermarket Scenario

The contributions of the framework towards a real supermarket scenario include the following aspects: (1) It provides a feasible way for retail practitioners to record customers’ shopping trajectories associated with their purchasing behaviors using RFID technology; (2) It designs a path graph schema with the support of a segment-item table and item-segment table, which can be used for the mapping between the physical world and the semantic cyber space. After the mapping, the data semantics can be understood by retail practitioners; (3) It offers a practical approach for preprocessing raw in-store RFID data, which contains five steps: data ordering and compression, data merging and anomaly detection, segment extraction, segment reassembling and extracting mainstream shopping transaction paths. Based on this approach, the raw data will become reliable and clean for retail practitioners; (4) It aims at mining actionable navigation patterns from a combination of customers’ shopping paths and their purchasing behavior data. Actionable knowledge is quite useful for decision making [24,25]. For example, we firstly cluster trajectories according to the duration of customers’ stays in the store. Then, practitioners can intuitively explore long “stock-up” trajectories where a long time is spent and many different types of products are purchased by customers. Some interesting patterns may be discovered by data mining algorithms, e.g., shoppers of these “stock-up” trajectories tend to frequently walk through a certain popular spot. Thus, decision-makers may consider offering active services, such as product recommendations and advertising, in that spot.

9. Conclusions

In this paper, we use the retail industry as an example to explore the potential of RFID technology for indoor mapping and navigation. In a supermarket scenario, RFID provides the ability to interact with items (*i.e.*, transport carts, trolleys, kegs and valuable products) without physical contact. Thus, item-level RFID infrastructures not only provide item handling efficiency, but also offer a promising way to capture customers' in-store behavior data and then gain insight into these data using data mining technology.

In this context, we provide a framework for mining actionable navigation patterns by combining RFID in-door mapping and data mining techniques. In the framework, multi-source in-door RFID data (*i.e.*, shopping path data and RFID-supported customers' purchasing behavior data) is integrated together for in-depth customers' behavior analytics. The framework consists of four modules: (1) mapping from the physical space to the cyber space; (2) data preprocessing; (3) data mining mechanism; and (4) knowledge understanding and utilization. Among them, the kernel part, *i.e.*, the scheme of extracting mainstream shopping transaction paths, is discussed in detail. The scheme of identifying mainstreams aims at catching the mainstream path sequences while discarding unnecessary redundant and repeated details, and is quite different from the scheme of extracting maximal forward reference. Two types of redundant patterns, *i.e.*, loop repeat pattern and palindrome-contained pattern, are recognized, and the corresponding algorithms are proposed and evaluated. Experimental results show that the algorithm is efficient and scalable for filtering these redundant patterns. On the whole, this work builds a bridge between indoor positioning and advanced data mining technologies, and provides a feasible way to study customers' shopping behaviors via multi-source RFID data.

Acknowledgments

We would like to thank Longbing Cao (University of Technology, Sydney) for helpful suggestions. This work is supported by National Natural Science Foundation of China (No. 71271191), the National Science & Technology Pillar Program during the 12th Five-year Plan Period of China (2012BAF12B11), Zhejiang Provincial Natural Science Foundation of China (No. Y1110960) and Scientific Research Foundation for the Returned Overseas Chinese Scholars (Ministry of Human Resources and Social Security of China, 2013).

Author Contributions

Bin Shen has initiated the idea of the work, conducted the research design and implemented the research. Qiuhua Zheng, Xingsen Li and Libo Xu have participated in concept, algorithm design and implementation and commented on the manuscript. Bin Shen has written the manuscript. The final manuscript has been approved by all authors.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Khoshelham, K.; Elberink, S.O. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors* **2012**, *12*, 1437–1454.
2. Girard, G.; Cote, S.; Zlatanova, S.; Barette, Y.; St-Pierre, J.; van Oosterom, P. Indoor Pedestrian Navigation Using Foot-Mounted IMU and Portable Ultrasound Range Sensors. *Sensors* **2011**, *11*, 7606–7624.

3. Ogawa, K.; Verbree, E.; Zlatanova, S.; Kohtake, N.; Ohkami, Y. Toward Seamless Indoor-Outdoor Applications: Developing Stakeholder-Oriented Location-Based Services. *Geo-Spat. Inf. Sci.* **2011**, *14*, 109–118.
4. Khoshelham, K.; Altundag, D.; Ngan-Tillard, D.; Menenti, M. Influence of Range Measurement Noise on Roughness Characterization of Rock Surfaces Using Terrestrial Laser Scanning. *Int. J. Rock Mech. Min. Sci.* **2011**, *48*, 1215–1223.
5. Gu, T.; Wang, L.; Wu, Z.Q.; Tao, X.P.; Lu, J. A Pattern Mining Approach to Sensor-Based Human Activity Recognition. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1359–1372.
6. Gu, Y.; Lo, A.; Niemegeers, I. A survey of Indoor Positioning Systems for Wireless Personal Networks. *IEEE Commun. Surv. Tutor.* **2009**, *11*, 13–32.
7. Isikdag, U.; Zlatanova, S.; Underwood, J. A BIM-Oriented Model for Supporting Indoor Navigation Requirements. *Comput. Environ. Urban Syst.* **2013**, *41*, 112–123.
8. Angeles, R. RFID Technologies: Supply-Chain Applications and Implementation Issues. *Inform. Syst. Manag.* **2005**, *22*, 51–65.
9. RFID Arena. Available online: http://www.rfidarena.com/2013/4/11/grocery-industry-operations-are-facing-a-real-paradigm-shift.aspx (accessed on 7 December 2014).
10. Prater, E.; Frazier, G.V.; Reyes, P.M. Future Impacts of RFID on E-Supply Chains in Grocery Retailing. *Supply Chain Manag. Int. J.* **2005**, *10*, 134–142.
11. Roussos, G. Enabling RFID in Retail. *Computer* **2006**, *39*, 25–30.
12. Loebbecke, C. Emerging Information System Applications in Brick-and-Mortar Supermarkets: A Case Study of Content Provision Devices and RFID-Based Implementations. In Proceedings of the 9th Pacific Asia Conference on Information Systems, Bangkok, Thailand, 7–10 July 2005.
13. Larson, J.S.; Bradlow, E.T.; Fader, P.S. An Exploratory Look at Supermarket Shopping Paths. *Int. J. Res. Mark.* **2005**, *22*, 395–414.
14. Sorensen, H. The Science of Shopping. *Mark. Res.* **2003**, *15*, 30–35.
15. Sorensen, H. Inside the Mind of the Shopper: The Science of Retailing. Pearson Prentice Hall: Upper Saddle River, NJ, USA, 2009.
16. Díaz-Vilariño, L.; Martínez-Sánchez, J.; Lagüela, S.; Armesto, J.; Khoshelham, K. Door Recognition in Cluttered Building Interiors Using Imagery and Lidar Data. In Proceedings of ISPRS Technical Commission V Symposium, Riva del Garda, Italy, 23–25 June 2014; Volume XL-5, pp. 203–209.
17. Liu, H.; Darabi, H.; Banerjee, P.; Liu, J. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Trans. Syst. Man Cybern.* **2007**, *37*, 1067–1080.
18. Zou, H.; Lu, X.; Jiang, H.; Xie, L. A Fast and Precise Indoor Localization Algorithm Based on an Online Sequential Extreme Learning Machine. *Sensors* **2015**, *15*, 1804–1824.
19. RFIDinsider. Available online: http://blog.atlasrfidstore.com/active-rfid-vs-passive-rfid (accessed on 21 February 2015).
20. Wikipedia. Available online: http://en.wikipedia.org/wiki/Radio-frequency_identification (accessed on 21 February 2015).
21. Hightower, J.; Borriello, G.; Want, R. *SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength*; UW CSE Technical Report #2000-02-02: Univ. Washington, Seattle, WA, USA, 18 February 2000.

22. Ni, L.M.; Liu, Y.; Lau, Y.C.; Patil, A.P. LANDMARC: Indoor Location Sensing Using Active RFID. *Wirel. Netw.* **2004**, *10*, 701–710.
23. Song J.; Haas, C.T.; Caldas, C.H. Tracking the Location of Materials on Construction Job Sites. *J. Construct. Eng. Manag.* **2006**, *132*, 911–918.
24. Cao, L. Actionable Knowledge Discovery and Delivery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2012**, *2*, 149–163.
25. Cao, L.; Zhang, C. Domain-Driven Actionable Knowledge Discovery in the Real World. In *Lecture Notes in Artificial Intelligence*; Ng, W.K., Kitsuregawa, M., Li, J., Eds.; Springer-Verlag: Berlin, Germany, 2006; Volume 3918, pp. 821–830.
26. Mabroukeh, N.R.; Ezeife, C.I. A Taxonomy of Sequential Pattern Mining Algorithms. *ACM Comput. Surv. (CSUR)* **2010**, *43*, 3.
27. Chen, M.-S.; Park, J.S.; Yu, P.S. Efficient Data Mining for Path Traversal Patterns. *IEEE Trans. Knowl. Data Eng.* **1998**, *10*, 209–221.
28. Wang, Y.-T.; Lee, A.J. Mining Web Navigation Patterns With a Path Traversal Graph. *Expert Syst. Appl.* **2011**, *38*, 7112–7122.
29. Borges, J.; Levene, M. Data Mining of User Navigation Patterns. In *Web Usage Analysis and User Profiling*; Springer: Berlin, Germany, 2000; pp. 92–112.
30. Pei, J.; Han, J.; Mortazavi-Asl, B.; Zhu, H. Mining Access Patterns Efficiently from Web Logs. In *Knowledge Discovery and Data Mining Current Issues and New Applications,* Springer: Berlin, Germany, 2000; pp. 396–407.
31. Shahabi, C.; Zarkesh, A.M.; Adibi, J.; Shah, V. Knowledge discovery from users web-page navigation. In Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE), Birmingham, UK, 7–8 April 1997; pp. 20–29.
32. Xing, D.; Shen, J. Efficient Data Mining for Web Navigation Patterns. *Inf. Softw. Technol.* **2004**, *46*, 55–63.
33. Zhu, W.; Cao, J.; Xu, Y.; Yang, L.; Kong, J. Fault-Tolerant RFID Reader Localization Based on Passive RFID Tags. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 2065–2076.
34. Trotter, M.S.; Durgin, G.D. Survey of Range Improvement of Commercial RFID Tags with Power Optimized Waveforms. In Proceedings of IEEE International Conference on RFID, Orlando, FL, USA, 14–16 April 2010; pp. 195–202.
35. Jeffery, S.R.; Alonso, G.; Franklin, M.J.; Hong W.; Widom, J. Declarative Support for Sensor Data Cleaning. In Proceedings of the International Conference on Pervasive, Dublin, Ireland, 7–10 May 2006; pp. 83–100.
36. Gonzalez, H.; Han, J.; Li, X. Mining Compressed Commodity Workflows From Massive RFID Data Sets. In Proceedings of the 15th ACM International Conference on Information and Knowledge Management, Arlington, VA, USA, 5–11 November 2006; pp 162–171.
37. Gonzalez, H.; Han, J.; Cheng, H.; Li, X.; Klabjan, D.; Wu, T. Modeling Massive RFID Data Sets: A Gateway-Based Movement Graph Approach. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 90–104.
38. Lee, C.-H.; Chung, C.-W. RFID Data Processing in Supply Chain Management Using a Path Encoding Scheme. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 742–758.

39. Johnson, R.; Tsouri, G.R.; Walsh, E. Continuous and Automated Measuring of Compliance of Hand-Hygiene Procedures Using Finite State Machines and RFID. *IEEE Instrum. Meas. Mag.* **2012**, *15*, 8–12.
40. Wang, F.; Liu, S.; Liu, P. Complex RFID Event Processing. *VLDB J. Int. J. Very Large Data Bases* **2009**, *18*, 913–931.
41. Terano, T.; Kishimoto, A.; Takahashi, T.; Yamada, T.; Takahashi, M. Agent-Based In-Store Simulator for Analyzing Customer Behaviors in a Super-Market. In *Knowledge-Based and Intelligent Information and Engineering Systems*; Springer: Berlin, Germany, 2009; pp. 244–251.
42. Dhar, R.; Nowlis, S.M. The Effect of Time Pressure on Consumer Choice Deferral. *J. Consum. Res.* **1999**, *25*, 369–384.
43. Dijkstra, E.W. A Note on Two Problems in Connexion With Graphs. *Numer. Math.* **1959**, *1*, 269–271.
44. Skiena, S. Dijkstra's Algorithm. In *Implementing Discrete Mathematics: Combinatorics and Graph Theory With Mathematica*; Addison-Wesley: Reading, MA, USA, 1990; pp. 225–227.