INFORMS Journal on Computing
Vol. 25, No. 1, Winter 2013, pp. 161–176
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1110.0484

Discovery of Online Shopping Patterns Across Websites

Yinghui (Catherine) Yang
Graduate School of Management, University of California, Davis, Davis, California 95616, yiyang@ucdavis.edu

Hongyan Liu, Yuanjue Cai
School of Economics and Management, Tsinghua University, Beijing 100084, China
{liuhy@sem.tsinghua.edu.cn, caiyj.04@sem.tsinghua.edu.cn}

In the online world, customers can easily navigate to different online stores to make purchases. The products purchased on one site are often associated with product purchases on other sites (e.g., a hotel reservation on one site and a car rental on another site). Whereas market basket analysis is often used to discover associations among products for brick-and-mortar stores, it is rarely applied in the online setting where consumers navigate among different online stores to buy products. We define online shopping patterns and develop two novel methods to perform market basket analysis across websites. While this research is motivated by online shopping applications, our contribution is mainly methodological. The two methods we develop in this paper can not only be used to identify various online shopping patterns across sites and products but can also be applied to settings where patterns exist across different dimensions. Experiments on both synthetic data and real online shopping data demonstrate the effectiveness of our methods.

Key words: data mining; e-commerce; analysis of algorithms
History: Accepted by Alexander Tuzhilin, Area Editor for Knowledge and Data Management; received January 2010; revised December 2010, July 2011; accepted August 2011. Published online in *Articles in Advance* December 7, 2011.

1. Introduction

Ever since the association rule technique (Agrawal and Srikant 1994) was developed for market basket analysis, it has been widely applied to various applications, especially for grocery stores. Market basket analysis is a useful method for discovering customer purchasing patterns by extracting associations among products from purchase transactions. It discovers rules showing that customers are more or less likely to buy certain products given the purchase of other products. This information can help store managers make better decisions about shelf allocation, product promotion, etc. For example, it has been discovered that purchasers of Barbie dolls are more likely to buy candy (Palmeri 1997); stores can therefore place high-margin candy near the Barbie doll display to increase sales.

Even though market basket analysis has gained some success in individual stores, it has rarely been used for Internet purchase transactions, which often span multiple sites. On the Internet, consumers can easily navigate to different online stores to make purchases. Associations exist not only among the products copurchased on the same site but also among the products purchased from different sites. A Web store can sell various products, and the same product can be purchased from more than one Web store. Our analysis of online shopping data discovered a number of patterns of associated purchases from multiple sites. For example, airline ticket booking on one site is often accompanied by a hotel reservation and car rental on some other sites. Similarly, apparel purchases on one site often co-occur with shoe purchases on other sites, and a purchase of event tickets on one site is often accompanied by a hotel reservation on some other site. In this paper, we focus on this type of pattern (i.e., the purchase of products on one site is associated with the purchase of products on other sites; note the sites do not need to be specific).

Discovering such patterns can, first of all, help us understand consumer online shopping behavior. In addition, there are a number of potential applications for this type of online shopping patterns. First, for companies looking to expand their product lines, they can use these patterns to include products, which are associated with the existing products on their sites. For example, sites selling event tickets may look into expanding their business to include local hotel reservations. Second, for sites that already carry all the products in the pattern, discovering such patterns can help them make product recommendations and plan promotions to attract consumers to buy products that

could otherwise be purchased from other stores. For example, if a site already carries both apparel and shoes, the site can run a discount on shoes (or recommend shoes) to the consumers who have just purchased apparel on the site to avoid losing sales to other sites that also sell shoes. Third, many sites now host advertisements for other sites for revenue purposes. These patterns can help them choose which ads to post on their sites. For example, hotel reservation sites may host ads for sites offering car rentals or plane or train tickets. There are also many other potential applications for these types of patterns. For affiliated online stores, which share sales data, identifying associations between products purchased from different stores can help them coordinate product selection, promotion, and cross-selling activities. As we discuss below, the types of patterns we can find in the online shopping domain can also be widely observed in many other applications, which makes discovering these patterns very beneficial to a wide variety of businesses online or off-line. The type of patterns these examples exhibit has a specific format (see §§2 and 3.4 for more details). This type of pattern is the focus of our study, and to our knowledge, it cannot be easily discovered by existing methods.

The data used for discovering such patterns can come from several different types of sources. First, there are many data vendors such as comScore Inc., which provide third-party, user-centric clickstream data. Such data vendors often sample Web users and pay them to track all their online activities. It has become a common business practice for companies to acquire such data sets. We surveyed major e-commerce companies, many of which indicate that they have purchased user-centric Web data from data vendors to analyze consumer behavior across sites and the performance of their competitors. Second, online advertisers (businesses that advertise their products online) often share data with online advertising companies such as a search engine, which has the capability to track Web users' behavior across its clients. Third, online marketplaces such as Amazon, eBay, and Yahoo! can easily observe the transactions generated by the many stores or sellers they service. Furthermore, a regular website that only has access to its own data can also benefit from analyzing data generated from different departments or product categories.

The type of pattern studied in this paper not only exists in the online world but can also be found in other contexts. For example, in many retail associations, the members share data among themselves. Such associations are particularly common among companies selling consumer products. For example, the WorldWide Retail Exchange enables information sharing among retailers operating in the food, apparel, and drugstore sectors. Its members include CVS, Walgreens, Safeway, Best Buy, Target, and JCPenney, among many others. Even for a single retailer, there can also be many channels or locations where its customers can make purchases. In this situation, it is possible for a company to analyze its consumers' behavior across these different channels or locations. With the development of modern data technologies such as data warehouses, it is becoming increasingly feasible for businesses to pool information resources from various channels or places to get a better picture of consumer behavior. In many cases, products can be interpreted as other types of items such as attribute–value pairs, and sites can be stores, locations, sellers, etc. (e.g., a site can be calling customer service, where the product is a certain type of complaint). Therefore the methods we developed in this paper can be applied to many applications to identify patterns across multiple dimensions, where sites and products can be interpreted according to the application.

In this paper, we define online shopping patterns that can potentially be discovered in the applications discussed above. We are not able to modify existing methods to discover such patterns (see §5 for more details on the existing literature, and see §3.4 for more discussion on our contribution). In this paper, we develop two methods to discover online shopping patterns among products sold on different websites as well as similar patterns in other applications. While this research is motivated by online shopping applications, our contributions in this paper are mainly methodological. The two methods we develop in this paper can not only be used to find various online shopping patterns with sites and products but can also be applied to settings where patterns exist across different dimensions, as we discuss more in §3.4.

The remainder of the paper is organized as follows. Section 2 defines the problem and introduces the definition of the online shopping pattern (OS-pattern). Section 3 develops two frequent OS-pattern mining methods—the OSP-tree and OSP-level. Section 4 reports the experimental results from the analysis of both real Web shopping data and synthetic data. Related work is presented in §5. Finally, §6 summarizes our study and outlines an agenda for future research.

2. Online Shopping Pattern

We consider a market basket transaction database that contains purchase records generated by a group of consumers on multiple websites. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of products, and let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of websites. Consumers can purchase any product p_i on any website s_j (for $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$).

Definition 1 (Online Shopping Transaction (OS-Transaction)). An online shopping transaction is defined as a list of products purchased on different sites (by the same consumer) within a given period of time: $\{s_{k1}(p^1_{k1}, p^2_{k1}, \ldots, p^{a1}_{k1}), s_{k2}(p^1_{k2}, p^2_{k2}, \ldots, p^{a2}_{k2}), \ldots, s_{kl}(p^1_{kl}, p^2_{kl}, \ldots, p^{al}_{kl}),\}$, where $p^{ai}_{ki} \in P$, $s_{ki} \in S$, and $s_{ki} \neq s_{kj}$ (for $i \neq j$ and $i, j \in \{1, 2, \ldots, l\}$).

We refer to each component (e.g., $s_{k1}(p^1_{k1}, p^2_{k1}, \ldots, p^{a1}_{k1})$) within an OS-transaction as a *site-level transaction*. A product can occur in more than one site-level transaction within an OS-transaction, but a site can only appear once in an OS-transaction (i.e., purchases made on the same site on multiple occasions within the scope of the OS-transaction will be grouped together to form one site-level transaction). The time period can be a day, a week, or a month, depending on how broad we want to look at the associations among purchases. Rather than time period, the scope can instead be defined along other dimensions. For example, an OS-transaction can be used to capture all the purchases from one single consumer (within a certain time frame). Table 1 shows some sample OS-transactions.

Different types of association rules can be discovered from a set of OS-transactions (e.g., an association rule *book* → *CD* means a customer who buys the book will also buy the CD). Association rules that do not mix products with sites (e.g., $\{p_1, p_2\} \rightarrow p_3$ and $s_1 \rightarrow \{s_2, s_3\}$) can be easily discovered using existing association rule methods (Agrawal and Srikant 1994). Here, we define online shopping patterns considering the relationships between sites and products. The patterns we define in this paper are compatible to the concept of itemsets (e.g., itemset {*book, CD*} means the book and CD are purchased together), and the corresponding association rules can be easily derived from itemsets. (See §5 for the background knowledge on association rules and itemsets.) Let an OS-transaction database be denoted $\mathrm{DB} = \{T_1, T_2, \ldots, T_u\}$, where T_k ($k \in \{1, 2, \ldots, u\}$) is an OS-transaction.

Definition 2 (Online Shopping Pattern (OS-Pattern)). An online shopping pattern is $\{(p^1_{d1}, p^2_{d1}, \ldots, p^{f1}_{d1}), (p^1_{d2}, p^2_{d2}, \ldots, p^{f2}_{d2}), \ldots, (p^1_{de}, p^2_{de}, \ldots, p^{fe}_{de})\}$, which satisfies the following condition: for each T_k ($k \in \{1, 2, \ldots, u\}$) in DB counting toward the frequency of this pattern, the site that generated $(p^1_{di}, p^2_{di}, \ldots, p^{fi}_{di})$ is different from the site that generated $(p^1_{dj}, p^2_{dj}, \ldots, p^{fi}_{dj})$ (for $i \neq j$ and $i, j \in \{1, 2, \ldots, e\}$). Each element of the *OS-pattern* $(p^1_{di}, p^2_{di}, \ldots, p^{fi}_{di})$ is called a site-level itemset, or itemset for short. The number of itemsets an OS-pattern contains is called the size or length of the pattern.

Table 1 An Online Shopping Database, DB

TID	OS-transactions
T_1	$\{s_1(p_1, p_2), s_2(p_3, p_4), s_3(p_5)\}$
T_2	$\{s_2(p_3, p_4), s_3(p_5, p_6), s_4(p_1)\}$
T_3	$\{s_1(p_1, p_4), s_2(p_1, p_3)\}$

Note. TID is the unique identifier of an OS-transaction.

The *support* of an OS-pattern is the number of OS-transactions that contain this pattern. An OS-pattern is considered frequent if its support exceeds the given minimum support threshold. Take Table 1 as an example; $\{(p_1), (p_3, p_4), (p_5)\}$ will be a frequent OS-pattern if the support threshold is 2. Since (p_1), (p_3, p_4), and (p_5) appear in different site-level transactions in T_1 and T_2, both T_1 and T_2 will contribute to the support count of $\{(p_1), (p_3, p_4), (p_5)\}$. Note, for ease of representation, we use (p_3, p_4) and (p_3p_4) (i.e., with and without a comma) interchangeably throughout the paper.

A subset of the OS-patterns is the site-specific OS-patterns, which can be written as $\{s_{d1}(p^1_{d1}, p^2_{d1}, \ldots, p^{f1}_{d1}), s_{d2}(p^1_{d2}, p^2_{d2}, \ldots, p^{f2}_{d2}), \ldots, s_{de}(p^1_{de}, p^2_{de}, \ldots, p^{fe}_{de})\}$, where $s_{di} \neq s_{dj}$ (for $i \neq j$ and $i, j \in \{1, 2, \ldots, e\}$). In the site-specific OS-patterns, we know exactly the site where certain products are purchased. Compared with the site-specific OS-patterns, the OS-patterns are more general. It only requires the groups of products within the pattern to occur on different sites but does not care about the specific sites. For example, an OS-pattern {(*book*, *CD*), (*games*, *movie*)} says that the book and CD are bought on one site, and concurrently, the games and movie are bought on another (different) site. A site-specific OS-pattern {Amazon(*book*, *CD*), eBay(*games*, *movie*)} says that the book and CD are bought on Amazon, and concurrently, the games and movie are bought on eBay.

Both the site-specific OS-patterns and the more general OS-patterns can be useful to managers. For a certain Web store, site-specific OS-patterns can pinpoint the exact sites whose products are associated with the products on its own site. This can facilitate cooperation between affiliated sites. However, such site-specific OS-patterns may be rare because of the large number of websites on the Internet (this is validated in our experiments). Even though a site-specific OS-pattern may not be frequent, its corresponding OS-pattern can be frequent. This means there will be more OS-patterns than site-specific OS-patterns. Finding the more general OS-patterns will give managers more patterns to work with, and in many situations, we may not care so much about the exact sites. This can help the site design mechanisms to attract sales, which could otherwise be lost to other stores. Under the situation where the number of websites involved

is limited (e.g., for several affiliated websites), the site-specific OS-patterns might be more useful to managers. If the number of websites involved is large, finding the more general OS-patterns will be more helpful. If needed, the methods we develop for the more general OS-patterns can be tailored to discover the site-specific OS-patterns.

We have investigated existing methods and were not able to modify them to discover frequent OS-patterns because of the difference in data structure and pattern format. In the next section, we present two methods to discover frequent OS-patterns. As we explain in §3.4, the methods we propose can also be applied to discover other types of online shopping patterns.

3. Discovering Frequent OS-Patterns

In this study, we develop two methods, the OSP-tree (§3.1) and OSP-level (§3.2), to mine the complete set of frequent OS-patterns. In §3.3, we provide theoretical analysis comparing these two methods on time and space complexity. Section 3.4 lays out the different types of patterns these two methods can discover.

3.1. Tree-Based Method: OSP-Tree

3.1.1. Main Steps of OSP-Tree. Figure 1 outlines the main procedures of the OSP-tree approach (online shopping pattern tree).

Below, we explain the major steps of this algorithm in detail.

Step 1. *Find frequent itemsets from site-level transactions*: For an OS-pattern to be frequent, each itemset inside that OS-pattern has to appear in at least one site-level transaction for a sufficient number of OS-transactions. For example, the OS-pattern $\{(p_1), (p_3, p_4), (p_5)\}$ contains three itemsets (p_1), (p_3, p_4), and (p_5). Each one of them needs to appear in at least one site-level transaction of an OS-transaction for that OS-transaction to be counted toward its support value. If an itemset appears in more than one site-level transaction in an OS-transaction, it will only be counted once. For example, in Table 1, (p_1) appears in two site-level transactions in T_3, but T_3 will only contribute 1 to (p_1)'s support count. For an itemset to be counted, all the products in the itemset have to come from the same site-level transaction within an OS-transaction. For example, T_1 will contribute 1 to the support count of (p_3, p_4), but not to (p_3, p_5). The reason is because p_3 and p_4 are both from s_2, and p_3 and p_5 are from two different sites—s_2 and s_3. For each such itemset found, we map it to a unique identifier. At the end of this step, we get a set of frequent itemsets, as shown in Table 2. Throughout the paper, *itemset* is strictly used to refer to the ones found from the site-level transactions in this step (e.g., $i_1, \ldots, i_5$).

Input: An OS-transaction database DB, a minimum support α
Output: The complete set of frequent OS-patterns
Major steps:

1. Find all the frequent itemsets from the site-level transactions in DB, given the minimum support α.
2. Transform DB to DT by replacing the original products with the frequent itemsets discovered.
3. Call *BuildTree*(DT) to construct a tree T for database DT.
4. Call *findOSP*(T, α) to find the complete set, O, of OS-patterns.
5. Replace the itemsets of each OS-pattern in O with the original products, then Output O.

Figure 1 Major Steps of OSP-Tree

Table 2 Frequent Itemsets (Assuming $\alpha = 2$)

Itemsets	(p_1)	(p_3, p_4)	(p_3)	(p_4)	(p_5)
Support	3	2	3	3	2
Mapped to	i_1	i_2	i_3	i_4	i_5

Step 2. *Database transformation*: Each OS-transaction in DB is transformed according to the new identifiers generated in Step 1. If a site-level transaction contains all the products in i_k, we write i_k inside this site-level transaction. Now, the OS-transactions in Table 1 are converted into the ones in Table 3.

Step 3. *Tree construction*: We first sort the frequent itemsets found in Step 1 (e.g., $\{i_1, i_2, i_3, i_4, i_5\}$ in Table 2) according to a certain order (e.g., support-based order or alphabetic order). Here, suppose the order is $i_1 \to i_2 \to i_3 \to i_4 \to i_5$ for Table 2. For each OS-transaction in DT, we pick one itemset from each site-level transaction, sort the chosen itemsets according to the predefined order, and create a path in the tree following the order. For example, for T_1 in Table 3, we pick i_1 from the first site-level transaction $s_1(i_1)$, i_2 from the second $s_2(i_2, i_3, i_4)$, and i_5 from the last $s_3(i_5)$, to form an OS-pattern $\{i_1, i_2, i_5\}$, which corresponds to the second leftmost path of the tree in Figure 2. In the leaf node of the path, we record the TID of the corresponding OS-transaction in a set, which is called *TIDset*. T_1 has three possible OS-patterns ($\{i_1, i_2, i_5\}$, $\{i_1, i_3, i_5\}$, $\{i_1, i_4, i_5\}$), which correspond to three paths in the tree. When inserting a new path to the tree, the children of each node need to be sorted according to the predefined order. For the database shown in Table 3, all the OS-patterns that we use to build the tree are listed in Table 4, and the tree is illustrated in Figure 2. (Note that these OS-patterns

Table 3 Transformed Database DT

TID	OS-transactions
T_1	$\{s_1(i_1), s_2(i_2, i_3, i_4), s_3(i_5)\}$
T_2	$\{s_2(i_2, i_3, i_4), s_3(i_5), s_4(i_1)\}$
T_3	$\{s_1(i_1, i_4), s_2(i_1, i_3)\}$

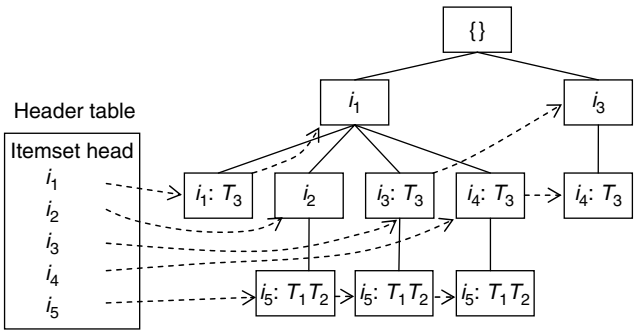

Figure 2 The Tree for DT in Table 3

are not the entire set of possible OS-patterns. They are just the ones used to build the tree.)

The header table is created to facilitate the tree traversal. The itemsets in the header table are sorted by the predefined order, and the link from an itemset in the table points to the first occurrence of that itemset in the tree. (Note that i_k is an itemset instead of an item.) Nodes with the same itemset are linked together using such node links based on the sequence of their appearances in the tree (going through the paths from left to right and for each path going from the leaf to the root). We call this tree an OSP-tree.

Step 4. *Read the complete set of OS-patterns from the OSP-tree*: Although the way the tree is built is very different from the frequent pattern (FP)-tree (Han et al. 2000), this step is similar to FP-growth (Han et al. 2000) once the tree is built. The major difference is how the support of an OS-pattern is computed. One OS-transaction can generate more than one OS-pattern containing a particular itemset (e.g., i_5) but will only contribute 1 to the support of that itemset. Therefore we cannot record the count in the nodes as was done in FP-growth. Instead, we record the TIDs of the OS-transactions in the nodes. The support of an OS-pattern is the number of TIDs in the *TIDset* in the corresponding node.

In this paper, we do not provide formal algorithms for this step. Please refer to Han et al. (2000) for more details. Here, we use an example to illustrate how to read all the OS-patterns from the OSP-tree built in Step 3. Taking the OSP-tree shown in Figure 2 as an example, we first consider mining OS-patterns containing itemset i_5, which is at the end of the header table list. Then, we consider the following sets of OS-patterns consecutively: the set with itemset i_4 but without itemset i_5; the set with itemset i_3 but without itemsets i_4 and i_5; the set with itemset i_2 but without itemsets i_3, i_4, and i_5; and the set with i_1 itself. The combination of these sets covers the entire search space. Since i_5 is frequent, we first output it as an OS-pattern. Then, following the chain of its node links, three branches can be found ($i_1i_2i_5$, $i_1i_3i_5$, and $i_1i_4i_5$), which form the conditional database (i.e., all the tuples in this database contain certain itemsets such as i_5). After filtering out the itemsets that are not frequent based on this subdatabase, an OSP-tree is built based on this conditional database, which is called the conditional tree (see Figure 3 for the tree, and refer to Han et al. 2000 for more details on the conditional tree).

In Figure 3, the *TIDset* of each leaf node is replaced by i_5's *TIDset*, because these itemsets co-occur with i_5. Then mining is done recursively on this tree in the same way. Every OS-pattern found from this tree will be concatenated with i_5. For example, when we start from i_4 for this tree, we first output OS-pattern i_5i_4. Then, we get a one-node tree: (i_1: T_1T_2). At this point, we stop further recursive calls and output OS-pattern $i_5i_4i_1$, and we then return to consider itemset i_3 in Figure 3. When we perform mining for i_3, we will output OS-patterns i_5i_3 and $i_5i_3i_1$. Similarly, we process for i_2 in the header table and get OS-patterns i_5i_2 and $i_5i_2i_1$. After we processed all the itemsets in the header table in Figure 3, we return to the OSP-tree in Figure 2 and consider i_4.

For itemset i_4, two branches are extracted from Figure 2 as shown in Figures 4(a) and 4(d). For the first branch, it is easy to see that OS-pattern i_1i_4 not only occurs in transaction T_3 but also co-occurs with i_5 in transactions T_1 and T_2. Therefore i_4's *TIDset* should be the union of its original *TIDset* and the *TIDset* of i_4's child node. Because we have derived all the itemsets containing i_5 before, this branch is converted to

Table 4 Equivalent Database

No.	OS-pattern	TID
1	$\{i_1, i_2, i_5\}$	T_1
2	$\{i_1, i_3, i_5\}$	T_1
3	$\{i_1, i_4, i_5\}$	T_1
4	$\{i_1, i_2, i_5\}$	T_2
5	$\{i_1, i_3, i_5\}$	T_2
6	$\{i_1, i_4, i_5\}$	T_2
7	$\{i_1, i_1\}$	T_3
8	$\{i_1, i_4\}$	T_3
9	$\{i_1, i_3\}$	T_3
10	$\{i_3, i_4\}$	T_3

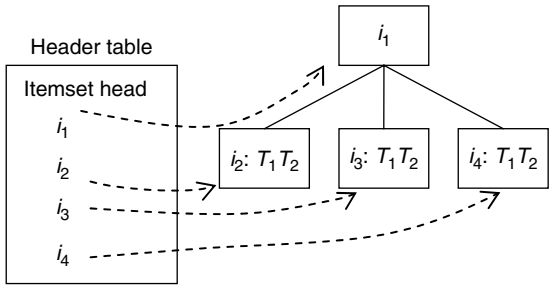

Figure 3 Conditional Tree for i_5

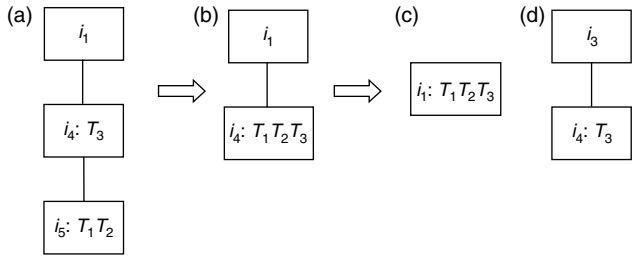

Figure 4 Building a Conditional Tree for i_4

the one shown in Figure 4(b). For the second branch shown in Figure 4(d), as i_3 co-occurs with i_4 only in one transaction, T_3, it does not satisfy the minimum support threshold, and therefore it will not be included in the conditional tree. Finally, i_4's conditional tree is shown in Figure 4(c).

Following the same procedure, we process i_3, i_2, and i_1 in the header table and discover all the frequent OS-patterns from the OSP-tree.

In Step 5, we replace the itemset identifiers with the products (e.g., replace i_2 with (p_3, p_4)).

Below, we reason that by using OSP-tree, we can discover the complete set of OS-patterns.

Theorem 1. *Given an OS-transaction database DB and a minimum support α, OSP-tree can discover the complete set of OS-patterns.*

Rational: If an OS-pattern is frequent, each itemset it contains must be frequent. Therefore, in the first step of this algorithm, all the frequent itemsets are found. As an OS-pattern consists of itemsets coming from different site-level transactions of an OS-transaction, in the second step, each site-level transaction of an OS-transaction is replaced by all the frequent itemsets it contains. Then, in the third step, for each OS-transaction, each possible largest combination of itemsets as listed in Table 4 is obtained and inserted to an OSP-tree. We say each combination is the largest because it includes itemsets from every site of the OS-transaction. All the possible OS-patterns we discover in the end will come from these largest itemsets. All the necessary information that will be needed to discover the final set of OS-patterns is encoded into the OSP-tree. Therefore the problem of mining frequent OS-patterns in database DB is transformed to that of mining the OS-patterns from the OSP-tree.

While traversing the OSP-tree, all the possible combinations of itemsets in the same branch are checked in a recursive and divide-and-conquer way. For example, for the five frequent itemsets in Figure 2, i_1, i_2, i_3, i_4, and i_5, all of their combinations can be divided into five sets. One includes combinations with itemset i_5, one with itemset i_4 but without itemset i_5, one with itemset i_3 but without itemsets i_4 and i_5, one with itemset i_2 but without itemsets i_3, i_4, and i_5, and the last one is i_1 itself. In this way, all the possible combinations will be checked against the minimum support threshold, and we will output the frequent ones. Therefore, all the frequent OS-patterns will be discovered.

Input: An OS-transaction database DB, a minimum support α
Output: The complete set of frequent OS-patterns
Major steps:
1. Find all the frequent itemsets from the site-level transactions in DB, given the minimum support α.
2. Transform DB to DT by replacing the original products with the frequent itemsets discovered.
3. Transform DT to DI by inverting sites and itemsets.
4. Call *findOSPLevel*(DI, α) to find the complete set, O, of OS-patterns.
5. Replace the itemsets of each OS-pattern in O with the original products, then Output O.

Figure 5 Major Steps of OSP-Level

3.2. Levelwise Method: OSP-Level

Tree-based methods often need large memory to store the tree, but they are normally more efficient at performing the mining task (Han et al. 2000, 2004). Here, we present a levelwise method as an alternative for situations where memory becomes a constraint. We later provide a theoretical complexity analysis to compare the tree-based method and the levelwise method on time and space consumption. Experiments on both synthetic data and real data are conducted to compare the two methods as well. Figure 5 outlines the main procedures of the OSP-level approach (online shopping pattern—levelwise algorithm).

Steps 1 and 2 are the same as those in Figure 1. Below, we explain how the database DI is built and how the levelwise method *findOSPLevel*(DI, α) works.

In Step 3, database DT is transformed to database DI. For each itemset in a transaction in DT, we will record all the sites on which it occurred. For example, for transaction T_3 in Table 3, itemset i_1 occurs on sites s_1 and s_2, i_3 occurs on site s_2, and i_4 occurs on site s_1. Therefore, this transaction is transformed to $i_1(s_1, s_2)$, $i_3(s_2)$, $i_4(s_1)$, which is shown in Table 5.

Figure 6 illustrates the details of *findOSPLevel*(DI, α) in Step 4.

The framework is similar to the frequent itemset mining algorithm Apriori (Agrawal and Srikant 1994). However, the most important steps (Steps 4 and 5

Table 5 Database DI

TID	OS-transactions
T_1	$i_1(s_1)$, $i_2(s_2)$, $i_3(s_2)$, $i_4(s_2)$, $i_5(s_3)$
T_2	$i_1(s_4)$, $i_2(s_2)$, $i_3(s_2)$, $i_4(s_2)$, $i_5(s_3)$
T_3	$i_1(s_1, s_2)$, $i_3(s_2)$, $i_4(s_1)$

Input: The transformed database DI, a minimum support α
The frequent itemsets discovered from Step 1:
$I = \{i_1, i_2, \ldots, i_f\}$
Output: The complete set O of frequent OS-patterns
Major steps:
1. $L_1 = \{i_1, i_2, \ldots, i_f\}$
2. $k = 1$
3. while L_k is not empty, do
4. Generate a set C_{k+1} of candidate OS-patterns
5. For each transaction T_i of DI
For each OS-pattern $O_j \in C_{k+1}$
Call *countFrequency*$(T_i, O_j, \varnothing, 1)$ to count the frequency of O_j
6. L_{k+1} = {all of OS-patterns in C_{k+1} that has frequency not less than α}
7. $k = k + 1$
8. return $O = \bigcup_k Lk$

Figure 6 ***findOSPLevel*(DI, α)**

Input: An OS-transaction T in DI, an OS-pattern O, a set of sites D, the sequence number of the itemset currently examined i (i.e., the ith itemset in O)
Output: If T contributes 1 to O's frequency, O.frequency increases by 1; otherwise not increased
Major steps:
1. $k = O.itemset[i]$, $j = 1$;
2. for each site of k in T, k.site[j] do begin
3. if k.site[j] is not in D, do begin
4. if i = sizeof(O)
5. O.frequency = O.frequency + 1;
6. Return 1;
7. else
8. Add k.site[j] to D;
9. if *countFrequency*$(T, O, D, i+1) = 1$ return 1;
10. else remove k.site[j] from D;
11. end
12. $j = j + 1$
13. end
14. return 0;

Figure 7 ***countFrequency*(T, O, D, i)**

in Figure 6) are different. Next, we will describe the details of these two steps.

In Step 4 of Figure 6, candidate OS-patterns with $(k + 1)$ itemsets are generated from frequent OS-patterns in L_k. What is different from algorithm Apriori is that when considering itemset combinations, a candidate OS-pattern can contain multiple itemsets with the same name (e.g., $\{i_1, i_1\}$). The reason why we also consider OS-patterns with the same itemsets is because the same itemsets can be purchased from different sites. We again take Table 3 as an example. We treat $I = \{i_1, i_2, i_3, i_4, i_5\}$ as a set of itemsets, and they are already frequent because they have been screened according to Step 1 in Figure 5. Now, we generate all the possible OS-patterns with two itemsets together with their frequency (see Table 6). In Table 6, the OS-patterns in boldface are frequent, and they are used to generate OS-patterns with three itemsets.

From Table 6, we pick any two frequent OS-patterns that only have one different itemset to form an OS-pattern with three itemsets. For example, $\{i_1i_2\}$ and $\{i_1i_3\}$ will form $\{i_1i_2i_3\}$. Then, we need to prune the ones whose subsets are not frequent (e.g., $\{i_1i_3i_4\}$). Note $\{i_1i_1\}$ and $\{i_1i_2\}$ will be combined into $\{i_1i_1i_2\}$.

In Step 5 of Figure 6, for each transaction T_i in DI and each candidate OS-pattern O_j, a recursive function *countFrequency* is called to check whether T_i contains O_j. If yes, the frequency of O_j is incremented by 1. The major steps of *countFrequency* are given in Figure 7.

Function *countFrequency*( ) mainly checks whether each itemset within a candidate OS-pattern comes from a different site within an OS-transaction in *DT*. If yes, increase the support count by 1. To facilitate this process, DT is transformed into DI first in Step 3 in Figure 5.

Taking candidate OS-pattern i_1i_4 as an example, *countFrequency*$(T_3, i_1i_4, \varnothing, 1)$ is called to check if transaction T_3 contributes 1 to i_1i_4's frequency. Note, $i = 1$ here, and the size of (i_1i_4), which is the number of itemsets contained in i_1i_4, is 2. k.site[j] represents the jth site in which itemset k occurs. In the first step, $k = i_1$, which is the first itemset of the candidate OS-pattern i_1i_4, and $j = 1$. In the second step, i_1's first site, s_1, is checked if it is in D, i.e., if this site is already claimed by another itemset in this OS-pattern. D is empty in the beginning, so s_1 is not in D (Step 3). Because $i = 1$ and the size of $(i_1i_4) = 2$ (Step 4 is not satisfied), Steps 8 and 9 are executed. After Step 8, $D = \{s_1\}$, and in Step 9, *countFrequency*$(T_3, i_1i_4, \{s_1\}, 2)$ is called. This time, i_1i_4's second itemset is checked. Now, $k = i_4$, and k.site[1] $= s_1$, which is in D. As i_4 only occurs in one site, this call is returned with a value 0. Then, in Step 10, s_1 is removed from D. The second site s_2, where i_1 occurred, will be checked, and this time, $D = \{s_2\}$. Checking then proceeds to itemset i_4; its site, s_1, is not in D. Therefore Steps 4 and 5 are executed. As a result, the frequency of i_1i_4 is increased by 1. This process is illustrated in Figure 8.

For the database shown in Table 3, Table 7 shows the frequent OS-patterns with three itemsets.

Because we cannot generate any larger candidate OS-pattern from these three OS-patterns, we go to

Table 6 Candidate OS-Patterns and Frequent OS-Patterns Containing Two Itemsets

Itemsets	$\{i_1i_1\}$	$\{\boldsymbol{i_1i_2}\}$	$\{\boldsymbol{i_1i_3}\}$	$\{\boldsymbol{i_1i_4}\}$	$\{\boldsymbol{i_1i_5}\}$	$\{\boldsymbol{i_2i_5}\}$	$\{\boldsymbol{i_3i_5}\}$	$\{\boldsymbol{i_4i_5}\}$	$\{i_3i_4\}$
Frequency counts	1	2	3	3	2	2	2	2	1

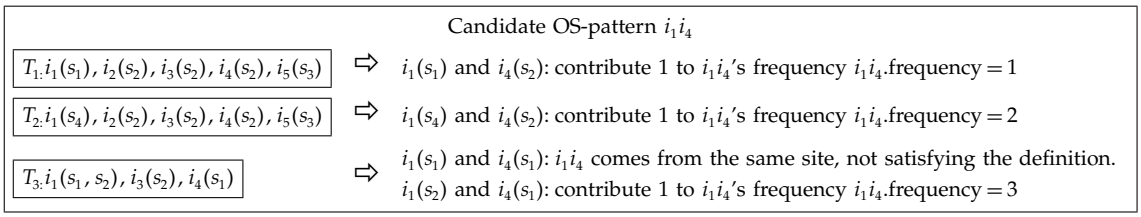

Figure 8 Illustration of the Frequency Counting Process

Step 5 of the algorithm OSP-level (see Figure 5). Every itemset of each OS-pattern is replaced with the original products. Table 8 lists the final results.

We can go one step further by removing the itemsets, which are not closed itemsets. The boldface cells in Table 8 are the closed itemsets. Note that $\{p_1p_4p_5\}$ is also not closed because $\{p_1(p_3p_4)p_5\}$ is its superset with the same support.

3.3. Complexity Analysis for the OSP-Tree and OSP-Level

Here, we conduct time and space complexity analysis to better understand the performance of the two methods we proposed. Suppose there are M OS-transactions in total. On average, each OS-transaction has S websites, and each site within an OS-transaction has P items. For the OSP-tree method, each OS-transaction corresponds to at most P^S paths, and each path averages S nodes (each node corresponds to an itemset). Then, the time complexity for building the OSP-tree is $O(MSP^S)$, and the space for storing the tree in memory is $O(MSP^S)$ in terms of the number of nodes if there is no sharing of nodes among paths. Suppose the longest OS-pattern contains N items and the number of frequent OS-patterns with length j is K for any $j \in [1 \ldots N]$. Then, there are about NK frequent OS-patterns in total. For each frequent OS-pattern, a conditional tree is built. The smallest tree only contains one node, and the biggest tree is the initial tree with at most MSP^S nodes. Suppose the average size of the tree is $MSP^S/2$. Therefore the time complexity of the OSP-tree method is about $O(NMKSP^S)$. For frequent OS-patterns with length N, there are at most N trees in the memory concurrently. Therefore the space complexity is $O(NMSP^S)$.

For the OSP-level method, suppose that the average number of candidate OS-patterns of each size is $(K+L)$ (note that the number of candidate OS-patterns is greater than the number of frequent OS-patterns). For an OS-pattern with average length $N/2$, suppose that the average number of sites that contain the pattern is $S/2$. The time complexity to count the frequency of such pattern is $(S/2)^{(N/2)}$. Therefore the time complexity of OSP-level is $O[NM(K+L)(S/2)^{(N/2)}]$. For each OS-transaction in database DI, there are, on average, SP itemsets. Therefore M OS-transactions corresponds to MSP itemsets in total. As the OSP-level needs to save this database and all of the frequent OS-patterns in memory, the space complexity is $O(MSP + NK)$.

Table 7 Frequent OS-Patterns with Three Itemsets

Itemsets	$\{i_1i_3i_5\}$	$\{i_1i_4i_5\}$	$\{i_1i_2i_5\}$
Counts	2	2	2

According to the above complexity analysis, OSP-tree has a time complexity of $O(NMKSP^S)$ and a space complexity of $O(NMSP^S)$, and the OSP-level has a time complexity of $O[NM(K+L)(S/2)^{(N/2)}]$ and a space complexity of $O(MSP + NK)$. It appears that the OSP-level tends to use less memory than the OSP-tree (especially as P and S increase). As for time consumption, there is no clear dominance. The OSP-tree might have an advantage when P and S are small, and the OSP-level will likely perform better as P and S increase (i.e., when a database contains longer OS-transactions, especially when users purchase a lot of items on each website). The OSP-level will likely have slower performance than the OSP-tree as L and N increase (L can be potentially big as the number of candidate patterns can be high; N is unlikely to be very big because the size of the OS-pattern will not be very big).

3.4. Extension to Other Types of Patterns and Discussion on Contribution

The OSP-tree and OSP-level methods we proposed can also be used to discover more types of online shopping patterns. Below, we list three types of online shopping patterns when both site and product information are available.

1. *Type* 1: $\{(p^1_{d1}, p^2_{d1}, \ldots, p^{f1}_{d1}), (p^1_{d2}, p^2_{d2}, \ldots, p^{f2}_{d2}), \ldots, (p^1_{de}, p^2_{de}, \ldots, p^{fe}_{de})\}$. This is the one we defined in Definition 2.

2. *Type* 2: $\{s_{d1}(p^1_{d1}, p^2_{d1}, \ldots, p^{f1}_{d1}), s_{d2}(p^1_{d2}, p^2_{d2}, \ldots, p^{f2}_{d2}), \ldots, s_{de}(p^1_{de}, p^2_{de}, \ldots, p^{fe}_{de})\}$, where $s_{di} \neq s_{dj}$ (for $i \neq j$ and $i, j \in \{1, 2, \ldots, e\}$). This kind of patterns is the site-specific OS-pattern we discussed in §2.

3. *Type* 3: A combination of three different types of components: sites, products purchased on the same site (without knowing which site), and products purchased on a specific site. This type also

Table 8 Final Results

OS-patterns	$\{(p_1)\}$	$\{(p_3)\}$	$\{(p_4)\}$	$\{(p_5)\}$	$\{(p_3p_4)\}$		
Counts	3	3	3	2	2		
OS-patterns	$\{(p_1)(p_3p_4)\}$	$\{(\boldsymbol{p_1})(\boldsymbol{p_3})\}$	$\{(\boldsymbol{p_1})(\boldsymbol{p_4})\}$	$\{(p_1)(p_5)\}$	$\{(p_3p_4)(p_5)\}$	$\{(p_3)(p_5)\}$	$\{(p_4)(p_5)\}$
Counts	2	3	3	2	2	2	2
OS-patterns	$\{(p_1)(p_3)(p_5)\}$	$\{(p_1)(p_4)(p_5)\}$	$\{(\boldsymbol{p_1})(\boldsymbol{p_3p_4})(\boldsymbol{p_5})\}$				
Counts	2	2	2				

includes those with two out of these three types of components:

$$\{s_{d1}, s_{d2}, \ldots, s_{dk}, (p^1_{d(k+1)}, \ldots, p^{f(k+1)}_{d(k+1)}), \ldots, (p^1_{dl}, \ldots, p^{fl}_{dl}),$$
$$s_{d(l+1)}(p^1_{d(l+1)}, \ldots, p^{f(l+1)}_{d(l+1)}), \ldots, s_{de}(p^1_{de}, p^2_{de}, \ldots, p^{fe}_{de})\}.$$

The patterns with only sites $\{s_{d1}, s_{d2}, \ldots, s_{de}\}$ or only products $\{p_{d1}, p_{d2}, \ldots, p_{de}\}$ can be easily discovered using standard association rule mining algorithms (Agrawal et al. 1993, Han et al. 2000). Note that the type 1 pattern is different from the product-only pattern $\{p_{d1}, p_{d2}, \ldots, p_{de}\}$. In type 1 patterns, we consider not only products but also websites (i.e., the products in each pair of parentheses need to come from the same site, and products in different parentheses need to come from different sites). For product-only patterns $\{p_{d1}, p_{d2}, \ldots, p_{de}\}$, the site information is ignored, and all the products purchased in the OS-transaction are considered to be items in a standard transaction regardless of the websites on which they are purchased.

Although it could be possible to modify existing algorithms, including those for multiple-level association rules (Han and Fu 1999) and intertransaction association rules (e.g., Tung et al. 1999, Feng et al. 2002) to discover type 2 patterns by using site–product pairs (e.g., s1p1) as items, we are not able to modify them to discover type 1 and type 3 patterns. This is the most important contribution of our paper. In addition, our data structure is different from that in multilevel association rule mining, which is most similar to our approach. Our data structure is a bipartite type of structure (as illustrated in Figure 9), where one product can be purchased on different sites, whereas multilevel association rule mining deals with a tree structure, where a child (product) can only belong to one parent (site). Moreover, our methods can find patterns with the same product appearing on both sides of an association rule (e.g., a rule $(book, CD) \rightarrow (book, movie)$, where a book appears on both sides of the rule (note that this rule can be easily derived from the OS-pattern $\{(book, CD), (book, movie)\}$).

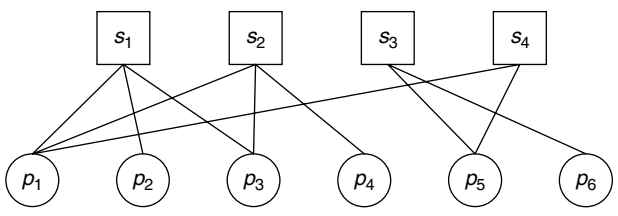

Figure 9 Our Data Structure

The OSP-tree and OSP-level methods we proposed can be directly applied to discover type 1 patterns. With the following small modification, our methods can be used to discover all three types of online shopping patterns. We regard each site s_i associated with each itemset (a set of products purchased from s_i) as an item and put it into the set of products. Then, the database shown in Table 1 is transformed into that in Table 9.

After that, both the OSP-tree and OSP-level can be applied to this data set to find OS-patterns. A component of an OS-pattern discovered could be $(s_i, p_{i1}, p_{i2}, \ldots, p_{ik})$. We can change it to $s_i(p_{i1}, p_{i2}, \ldots, p_{ik})$, which means products $p_{i1}, p_{i2}, \ldots, p_{ik}$ are purchased from site s_i.

As we extend our methods to discover more types of patterns, we can adapt our methods to more applications. For example, if there are a limited number of websites in the database, type 2 patterns might be of more importance as they are more specific. This might apply to the setting where a website or company is only sharing its data with a few other websites or companies (e.g., a retail association with a limited number of members). The type 1 pattern is more general and is a superset containing type 2 patterns. Finding type 1 patterns can apply to the setting where the number of websites is large (e.g., Amazon has a large number of stores), as well as cases where the managers are simply more interested in the more general patterns. It will also give managers more patterns to work with. Type 3 patterns can be even more general, including different types of components in an OS-pattern. One example for such pattern is {called customer service, website (product 1, product 2), (product 3, product 4)}. This pattern could be derived from

Table 9 A Transformed Online Shopping Database, DB

TID	OS-transactions
T_1	$\{(s_1, p_1, p_2), (s_2, p_3, p_4), (s_3, p_5)\}$
T_2	$\{(s_2, p_3, p_4), (s_3, p_5, p_6), (s_4, p_1)\}$
T_3	$\{(s_1, p_1, p_4), (s_2, p_1, p_3)\}$

a company's customer data from various communication channels. One explanation for this pattern is that customers often contact customer service via the phone, purchase products 1 and 2 from the company's website, and purchase products 3 and 4 from some other channel (such as a store at a certain location). Here, the different touch points (call, online website, different off-line stores) can be considered to be different websites and what people have done through these touch points can be considered to be products.

4. Experiments

We conducted experiments using both real Web shopping data and synthetic data to compare the time and space cost of the OSP-tree and OSP-level, and to evaluate the effectiveness of both methods.

4.1. Synthetic Data

Synthetic OS-transactions are generated to evaluate our methods. We use the generator developed in Agrawal and Srikant (1995), a standard synthetic data generator for associations and sequential patterns. It is widely used to evaluate the performance of various association and sequential pattern mining algorithms. On top of this data generator, we also added the site information (i.e., for each site-level transaction, we keep track of which site it is from). Table 10 maps the main parameters used in our experiments with those in Agrawal and Srikant (1995).

We vary the value of several parameters across multiple data sets. The parameters we studied are the number of OS-transactions (T), the average number of sites within an OS-transaction (S), the average number of products purchased on each site (P) (i.e., the average size of a site-level transaction), the total number of products available (I), and minimum support (minisup). The experiments demonstrate that our method can find the complete set of OS-patterns embedded in the synthetic data. The following figures illustrate the comparisons between the OSP-tree and OSP-level on time and space costs.

As Figure 10 demonstrates, OSP-tree takes less time than the OSP-level when the minimum support is not too high. The maximum memory consumed is always higher for the OSP-tree. Figure 11 conveys a similar message, that the OSP-tree takes less time but consumes more memory space.

Table 10 Parameter Mapping

Parameters in Agrawal and Srikant (1995)	Corresponding parameters in our data generator	Our symbols
Number of customers	Number of OS-transactions	T
Average number of transactions per customer	Average number of sites within an OS-transaction	S
Average number of items per transaction	Average number of products purchased on each site	P
Number of items	Number of products	I

As Figures 12 and 13 demonstrate, the execution time increases with the average number of sites within an OS-transaction (S) and the average number of products purchased on each site (P) for both the OSP-tree and OSP-level. For the OSP-tree, the most time-consuming step is the tree construction step. As P and S increase, each transaction becomes more complex, so the tree is likely to become more complex, which, in turn, increases the time spent for tree building and the space taken by the tree. The increase for the OSP-level is not as dramatic as that for the OSP-tree. Thus, when S and P are relatively low, the OSP-tree is a better choice in terms of execution time, but the OSP-level becomes more advantageous as S and P become larger. The OSP-level is better at space consumption as the tree takes up more memory space.

As the number of items (I) increases (see Figure 14), the support of each item will decrease given the fixed number of transactions. Thus the number of frequent patterns will decrease, which results in the decrease in execution time.

In each of the above five sets of experiments, the two algorithms output exactly the same set of OS-patterns. Figure 15 shows that the number of OS-patterns discovered under different parameters as minimum support increases. Note that the y axis is in logarithmic scale. We can see that these two algorithms output the same number of OS-patterns as we vary different parameters. This helps us validate the effectiveness of both algorithms (i.e., they can discover all the qualifying OS-patterns).

4.2. Online Shopping Data

We use the user-centric, online shopping data provided by comScore Inc. The data set we use captures all online purchases of 61 different product categories by 50,000 panelists during 10 months of 2004 (January–October). Among these 50,000 panelists, 24,824 have made purchases. More than 62% (15,416 panelists) of them have purchased at two or more sites. Each panelist in the sample represents a single computer that is tracked; hence we restrict our selection of Web users to those corresponding to a household size of one in the household demographic file provided. However, we note that this does not guarantee that only a single *person* uses each computer. We consolidate purchases from the same customer in the same month into one OS-transaction. Thus the maximum number of OS-transactions a customer can have is 10. After data processing, the total number of OS-transactions we get is 75,053. We deem an OS-pattern frequent if its count exceeds 10. We use product categories instead of individual products

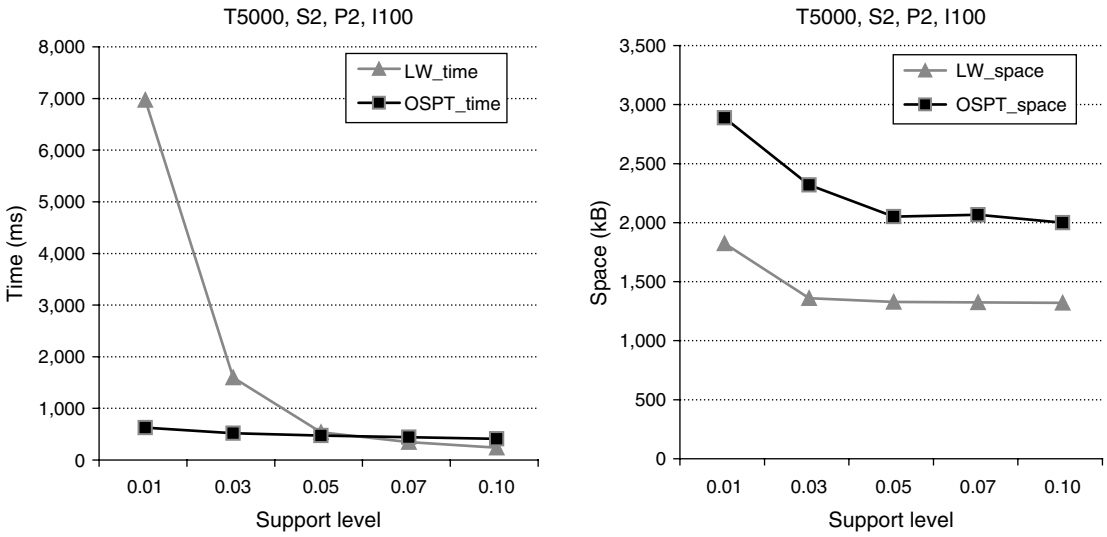

Figure 10 Effect of minisup on Performances

Notes. Here, T5000, S2, P2, I100 means we set T = 5,000, S = 2, P = 2, and I = 100 while changing the minimum support. LW represents the OSP-level, and OSPT represents the OSP-tree.

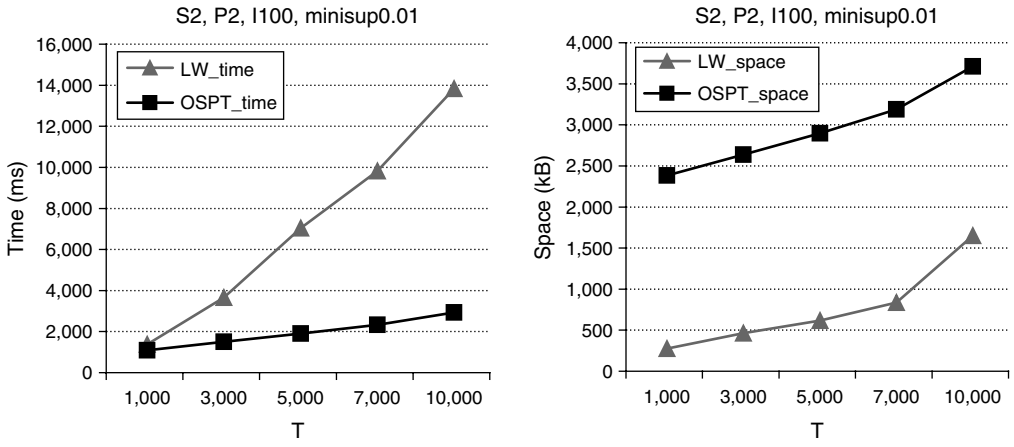

Figure 11 Effect of T on Performances

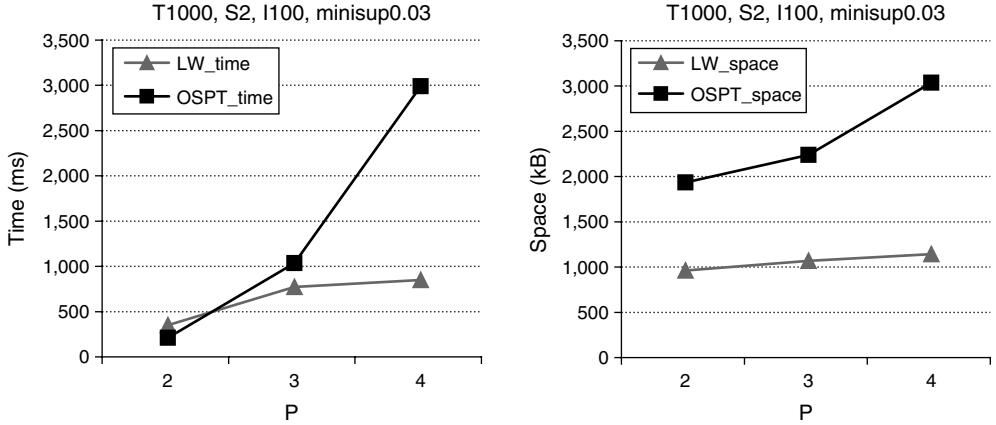

Figure 12 Effect of P on Performances

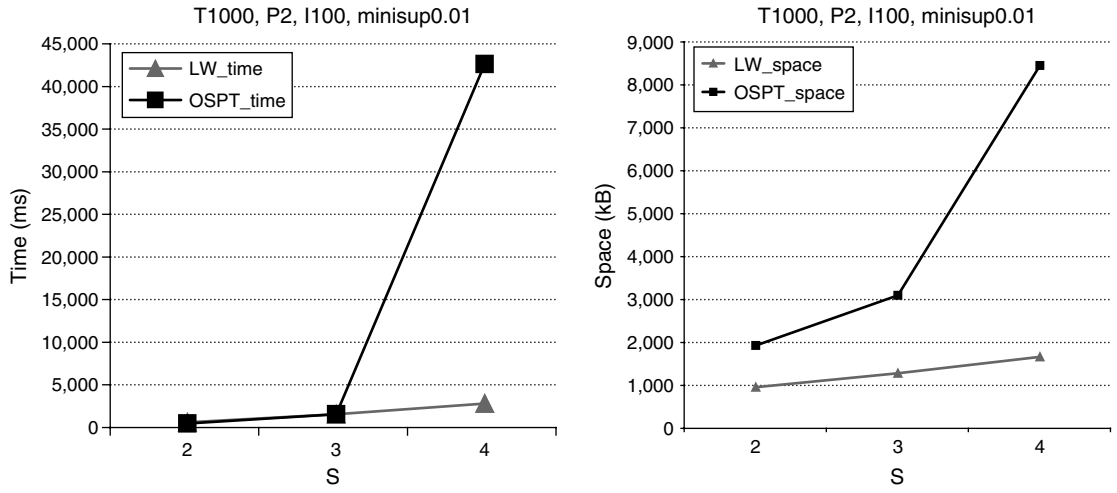

Figure 13 Effect of S on Performances

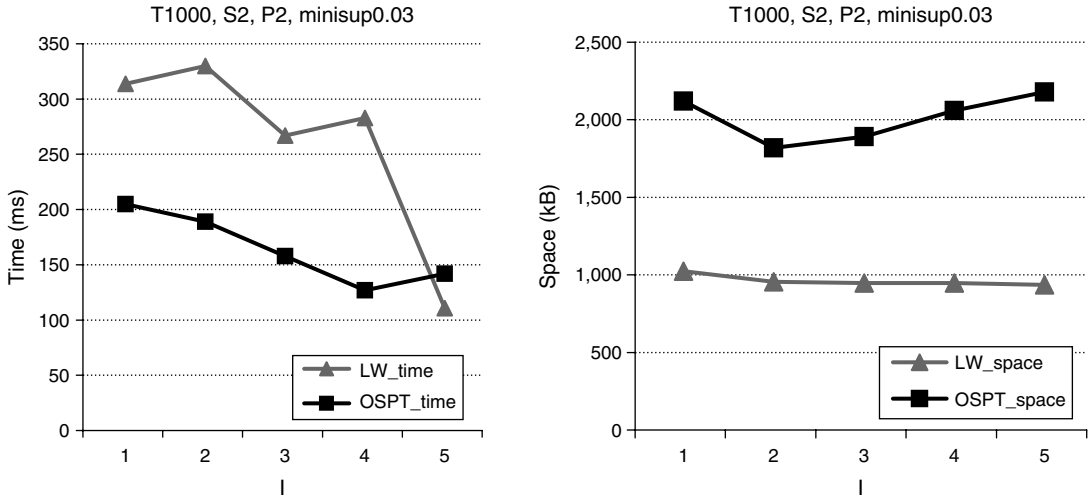

Figure 14 Effect of I on Performances

Note. The unit of the x axis is 100 (i.e., I = 2 means I = 200).

because it is very difficult to uniquely identify a product through its long name provided in the data. During the 10 months we study, the maximum number of distinct sites where a customer purchased products is 40, the minimum is 1, and the average is 3. The number of customers who purchased products from at least two sites within one month is 4,771. Figure 16 shows the distribution of the number of customers who have made purchases on a different number of sites. Figure 17 illustrates the distribution of the number of customers who purchased in a different number of product categories. On average, a customer made purchases in 3.35 distinct product categories during the 10 months we study. Within one month, the largest number of product categories a customer purchased in is 14.

The frequent OS-patterns discovered by the OSP-level and OSP-tree are exactly the same. We find 116 frequent OS-patterns with two itemsets (each occurs on a different site) and three frequent OS-patterns with three itemsets. We found a number of very interesting frequent OS-patterns. Table 11 lists some of the OS-patterns discovered from the data together with the number of OS-transactions supporting the pattern. As mentioned in §1, airplane ticket booking on one site is often accompanied by a hotel reservation and car rental on some other sites, and apparel purchases on one site often co-occur with shoe purchases on other sites. Consumers also tend to purchase apparel on one site and jewelry or watches on another site. These types of patterns can help a store decide which types of products to place on its site if it plans to extend its product line and also on what kinds of advertising to place on the site. For example, sites selling event tickets can help to advertise the local hotels. There may not be hundreds

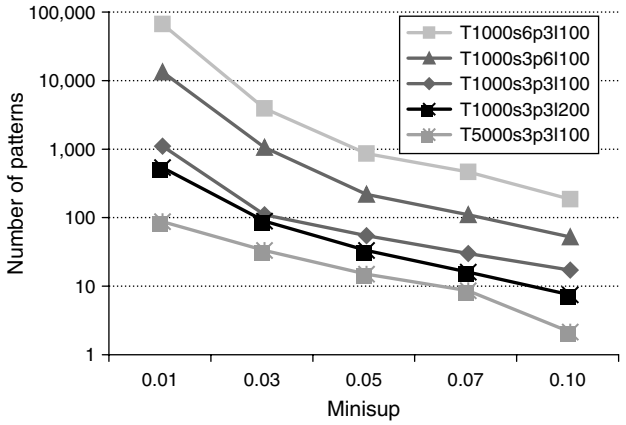

Figure 15 **The Number of OS-Patterns Discovered by Both Algorithms**

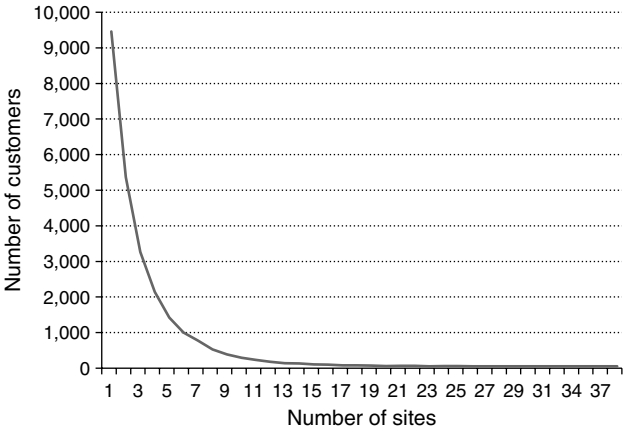

Figure 16 **Number of Customers Who Purchased on a Different Number of Sites**

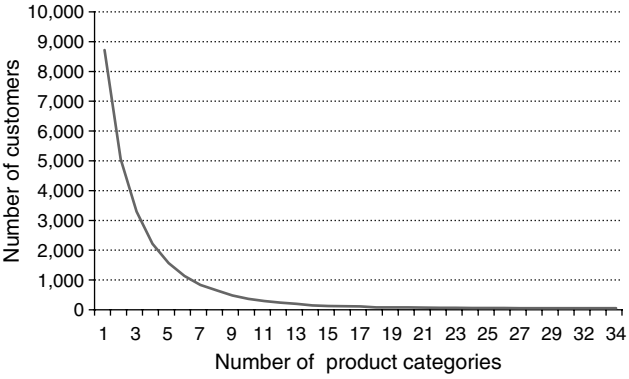

Figure 17 **Number of Customers Who Purchased in a Different Number of Product Categories**

of surprising OS-patterns in a particular real-world data set as expected in many data mining applications because most patterns reflect expected behavior. The more important thing is that we provide methods to find all such patterns if they do exist in data. On the other hand, finding those not-so-surprising patterns can also help confirm the existence of some suspected consumer behavior (e.g., the co-occurrence of an air ticket purchase, hotel reservation, and car rental on different sites).

Table 11 **OS-Pattern Examples**

OS-pattern	Count
{Air travel, Hotel reservations, Car rental}	21
{Hotel reservations, (Air travel, Car rental)}	19
{Air travel, Hotel reservations}	167
{Air travel, Car rental}	201
{Hotel reservations, Car rental}	134
{Event tickets, Hotel reservations}	38
{Music, Movies & videos}	70
{Books & magazines, Movies & videos}	74
{Health & beauty, Books & magazines}	45
{Apparel, Shoes}	85
{Apparel, Jewelry & watches}	41
{Apparel, Kitchen & dining}	26
{Apparel, Bed & bath}	44
{Apparel, Health & beauty}	76
{Apparel, Books & magazines}	95

We also investigated the OS-transactions that generated these interesting OS-patterns. The findings support our conjecture that if we restrict our patterns to specific websites (site-specific OS-patterns), we will have very few patterns to work with. Many of the OS-patterns we discover have products purchased from many different sites. For example, among the 134 OS-transactions supporting {Hotel reservations, Car rental}, hotels are reserved on 26 different sites, and car rental reservations are placed on 16 different sites. This further supports the importance of discovering OS-patterns, which are not site specific.

5. Related Works

In this section, we review the related works, which can be categorized into the following categories. (Yang et al. 2008 is an earlier version of this paper with just the OSP-tree algorithm and no experiments on synthetic data.)

Association Rule Mining: Association rule mining techniques are the main techniques used for market basket analysis. The concept of association rule mining is first introduced in Agrawal et al. (1993). Since then, there has been a great deal of work on association rule mining. An association rule is an expression $X \rightarrow Y$, where X and Y are sets of items. Each transaction in the transaction database used to mine association rules contains a set of items. Support and confidence are often used to measure the significance of an association rule. Support is the percentage or number of transactions that contains both X and Y. Confidence is the percentage of the transactions containing both X and Y among the transactions containing X. The association rule discovery process is often divided into two stages. The first stage is to

find all the frequent itemsets (i.e., sets of itemsets whose support is sufficiently high), and the second stage is to derive association rules from the frequent itemsets (confidence is often used as the measurement to select significant association rules). Some of the most popular association rule mining techniques include Apriori (Agrawal and Srikant 1994), Eclat (Zaki 2000), FP-growth (Han et al. 2000, 2004), tree projection (Agarwal et al. 2001), and PRICES (Wang and Tjortjis 2004) (see Kotsiantis and Kanellopoulos 2006 for more details on these algorithms). Our definition of OS-pattern is very different from that of an association rule, and the traditional association rule mining techniques cannot discover the type of online shopping patterns we are interested in.

Multilevel Association Rule Mining: Research in association rule mining has been extended in many directions. One of these directions is multilevel association rule mining. Research on multilevel association rules is most related to our work. Multiple-level association rules (Han and Fu 1999, Srikant and Agrawal 1995) are used to mine associations at multiple levels of taxonomy (e.g., the association between milk and bread and the association between 2% milk and wheat bread). Our approach differs from existing methods in multiple-level association rule mining on two aspects. First, the data structure is different. Our data structure is a bipartite type of structure (see Figure 9) where one product can be purchased on different sites, whereas multilevel association rule mining deals with a tree structure where a child (product) can only belong to one parent (site). Second, and most importantly, the format of the patterns we are looking for are different from those in the multiple-level association rule setting (as shown in §3.4). In addition, the rules generated from our OS-patterns can have the same product on both sides of the rule (e.g., $(book, CD) \rightarrow (book, movie)$).

Intertransaction Association Rule Mining: Another extension of association rule mining is intertransaction association rule mining. Traditional association rule mining finds associations among items within the same transaction, whereas intertransaction association rule mining finds associations among items across different transactions. The problem of intertransaction association rule mining was first proposed in Lu et al. (1998). Following that, algorithms such as E-Apriori and EH-Apriori (Lu et al. 2000), FITI (Tung et al. 1999), and ITP-miner (Lee and Wang 2007) were proposed for efficient mining. In Feng et al. (1999, 2002), a template model is introduced to guide the mining process so that the number of rules can be reduced. In Chen et al. (2005), an algorithm is presented to mine closed intertransaction association rules. Intertransaction association rules have been used in stock price movement prediction (Lu et al. 1998, 2000; Tung et al. 1999) and weather prediction (Feng et al. 2001). In intertransaction association rule mining, each item (product) is associated with a domain attribute value (site) (i.e., each site–product pair becomes an item). Our work, in contrast, does not attach the specific value (site) to items (products), which allows us to discover more general patterns. More specifically, the intertransaction association rule mining methods cannot be used to discover type 1 or type 3 patterns, but it is possible to apply these methods to discover type 2 patterns after appropriate modifications of the sliding windows.

Sequential Pattern Mining: Our work also relates to the literature on sequential pattern mining. Many algorithms have been proposed to detect frequent subsequences from sequence databases (Agrawal and Srikant 1995, Srikant and Agrawal 1996, Pei et al. 2004). A typical sequence pattern is that the purchase of a set of products is followed by another set of products. Time sequence plays a very important role in sequential pattern mining, and our work on mining OS-patterns can potentially be extended to incorporate the time factor. Currently, there is no sequence within an OS-pattern. All the site-level transactions within an OS-transaction are independent from each other.

Tag-Based Association Rule Mining: Another related literature is tag-based association rule mining. Schmitz et al. (2006) focus on discovering associations among users, tags, and resources. Because traditional association rule mining can only handle two-dimensional data, the paper projected the three-dimensional data (users, tags, and resources) onto a two-dimensional data structure (items and transactions) by either combining users and resources or combining users and tags. This method is not able to find our type 1 and type 3 patterns. In addition, we find associations between itemsets, whereas tag-based rule mining finds associations among items.

6. Conclusions

In this paper, we define online shopping patterns and develop novel methods to discover such patterns. More specifically, we define online shopping patterns to capture users' purchase behavior across different websites. Methods for discovering such patterns are not readily available to find online shopping patterns among products sold on different websites. Two methods (OSP-tree and OSP-level) are developed to effectively discover the complete set of such patterns from data. We provide theoretical analysis on the time and space complexity of these methods. Experiments on both real online shopping data and synthetic data demonstrate that our patterns are interesting and that our methods are effective at finding such patterns. Our contributions here include

new definitions of online shopping patterns, discovery methods, and experimental (simulated and real) results.

Our research is motivated by the online shopping application, but the methods we developed are general and can be applied to many other applications outside of the online shopping setting used to introduce the definition of OS-patterns. Websites can be interpreted as sellers in an online marketplace such as Amazon, clients of a search engine advertising company, and departments within a retailer. It can also be used to discover similar shopping patterns for off-line stores where user-centric shopping data across multiple stores are available. In addition, it can be applied in a setting where discovering associations between different categories of products is desirable. Product categories are loosely defined and can therefore incorporate customer service activities such as credit card purchases, telephone calling patterns, etc. Theoretically, the methods we developed can be applied to settings where patterns exist across different dimensions, which can be interpreted differently in different applications.

The methods developed in this paper can be easily applied to discover several types of online shopping patterns, involving both sites and products as we discussed in §3.4. For example, we can use the methods to discover the pattern site A $\rightarrow$ product B, which means that any purchase on site A will lead to the purchase of product B (on any other site), and the pattern site A product B $\rightarrow$ product C, which means that the purchase of product B on site A will lead to the purchase of product C (on any other site). Given that sites and products can be interpreted as other types of entities in different applications, the methods we developed can be widely applied to discover a broad range of patterns across multiple dimensions.

Although we conducted experiments on both synthetic and real data to prove the effectiveness of our methods, we can potentially provide even greater value if real data from more applications become available. One major drawback of the comScore Inc. data set we used in the experiments is that it is very difficult to identify the products across different sites. The data collection company (comScore Inc.) is a third-party company, and it did not coordinate with online retailers to work out a systematic way to identify products across different sites on the Internet. It is also very difficult to do so given that the entire Internet is huge and the total number of online retailers is very large. Using product categories instead of individual products serves the purpose of demonstrating the effectiveness of our methods, but we will look for opportunities in the future to acquire data from companies where products can be uniquely identified. In many of the applications discussed in §1, data obtained from a specific business (such as a search engine (e.g., Google), a marketplace (e.g., Amazon), and a retailer) can be of great value to study in the future. Such businesses have more control over their data; as a result, products are more likely to be properly identified, and the data, in general, will be cleaner.

From a methodological perspective, our work also offers several interesting extensions in the future. For instance, data streams offer a unique challenge and opportunity for methodology development as data need to be acted on in real time and existing patterns need to be updated as more data are observed. Developing an incremental learning mechanism will greatly improve the efficiency of the pattern discovery process. Another methodological extension would be to add more layers of dimensions and define more complex patterns across more than two dimensions. In addition, there is no sequence among the components within an OS-pattern. In the future, we can investigate ways to extend our methods to incorporate sequence. We hope to build on the framework presented in this paper and consider more viable extensions in the future.

Acknowledgments

This research is partly supported by the National Natural Science Foundation of China [Grants 70871068, 70890083, 90924302, and 70890084], and the Ministry of Science and Technology of China [Grant 2006AA010106].

References

Agrawal, R., R. Srikant. 1994. Fast algorithms for mining association rules. *Proc. 20th VLDB Conf., Santiago, Chile*, 487–499.

Agrawal, R., R. Srikant. 1995. Mining sequential patterns. *Proc. 1995 Internat. Conf. Data Engrg.*, IEEE Computer Society, Washington, DC, 3–14.

Agarwal, R., C. Aggarwal, V. Prasad. 2001. A tree projection algorithm for generation of frequent itemsets. *J. Parallel Distributed Comput.* **61**(3) 350–371.

Agrawal, R., T. Imielinski, A. N. Swami. 1993. Mining association rules between sets of items in large databases. *Proc. 1993 ACM SIGMOD Internat. Conf. Management of Data, Washington, DC*, 207–216.

Chen, J., L. Ou, J. Yin, J. Huang. 2005. Efficient mining of cross-transaction Web usage patterns in large database. *Proc. 3rd Internat. Conf. Networking Mobile Comput., Zhangjiajie, China*, 519–528.

Feng, L., T. Dillon, J. Liu. 2001. Inter-transactional association rules for multi-dimensional contexts for prediction and their application to studying meteorological data. *Data Knowledge Engrg.* **37**(1) 85–115.

Feng, L., H. Lu, J. Yu, J. Han. 1999. Mining inter-transaction associations with templates. *Proc. 1999 ACM CIKM*, ACM, New York, 225–233.

Feng, L., J. X. Yu, H. Lu, J. Han. 2002. A template model for multidimensional inter-transactional association rules. *VLDB J.* **11**(2) 153–175.

Han, J., Y. Fu. 1999. Mining multiple-level association rules from large databases. *IEEE Trans. Knowledge Data Engrg.* **11**(5) 798–805.

Han, J., J. Pei, Y. Yin. 2000. Mining frequent patterns without candidate generation. *Proc. 2000 ACM-SIGMOD Internat. Conf. Management Data, Dallas*, 1–12.

Han, J., J. Pei, Y. Yin, R. Mao. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining Knowledge Discovery* **8**(1) 53–87.

Kotsiantis, S., D. Kanellopoulos. 2006. Association rules mining: A recent overview. *GESTS Internat. Trans. Comput. Sci. Engrg.* **32**(1) 71–82.

Lee, A. J. T., C. S. Wang. 2007. An efficient algorithm for mining frequent inter-transaction patterns. *Inform. Sci.* **177**(17) 3453–3476.

Lu, H., L. Feng, J. Han. 2000. Beyond intratransaction association analysis: Mining multidimensional intertransaction association rules. *ACM Trans. Inform. Systems* **18**(4) 423–454.

Lu, H., J. Han, L. Feng. 1998. Stock movement prediction and *n*-dimensional inter-transaction association rules. *Proc. 3rd ACM-SIGMOD Workshop Res. Issues Data Mining Knowledge Discovery, Seattle*, 12:1–12:7.

Palmeri, C. 1997. Believe in yourself, believe in the merchandise. *Forbes* **160**(5) 118–124.

Pei, J., J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M. C. Hsu. 2004. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Trans. Knowledge Data Engrg.* **16**(11) 1424–1440.

Schmitz, C., A. Hotho, R. Jäschke, G. Stumme. 2006. Mining association rules in folksonomies. *Data Sci. Classification: Proc. 10th IFCS Conf., Stud. Classification, Data Anal., Knowledge Organ., Ljubljana, Slovenia*, 261–270.

Srikant, R., R. Agrawal. 1995. Mining generalized association rules. *Proc. 21st VLDB Conf., Zurich, Switzerland*.

Srikant, R., R. Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. *Proc. Internat. Conf. Extending Database Tech., Taipei, Taiwan*, 3–17.

Tung, A. K. H., H. Lu, J. Han, L. Feng. 1999. Breaking the barrier of transactions: Mining inter-transaction association rules. *Proc. SIGKDD-1999, San Diego*, 297–301.

Wang, C., C. Tjortjis. 2004. PRICES: An efficient algorithm for mining association rules. Z. R. Yang, H. Yin, R. M. Everson, eds. *Intelligent Data Engineering and Automated Learning*. Lecture Notes in Computer Science, Vol. 3177. Springer, Berlin, 352–358.

Yang, Y., H. Liu, Y. Cai. 2008. Online market basket analysis across web sites. *Proc. 18th Workshop Inform. Tech. Systems, Paris*.

Zaki, M. J. 2000. Scalable algorithms for association mining. *IEEE Trans. Knowledge Data Engrg.* **12**(3) 372–390.