Electronic Commerce Research and Applications 8 (2009) 241–251

Contents lists available at ScienceDirect

Electronic Commerce Research and Applications

journal homepage: www.elsevier.com/locate/ecra

Discovering recency, frequency, and monetary (RFM) sequential patterns from customers' purchasing data

Yen-Liang Chen [a,*], Mi-Hao Kuo [a], Shin-Yi Wu [b], Kwei Tang [c]

[a] Department of Information Management, National Central University, No. 300, Jhongda Road, Chung-Li 320, Taiwan, ROC
[b] The Industrial Technology Research Institute, Hsinchu 320, Taiwan, ROC
[c] Krannert School of Management, Purdue University, West Lafayette, IN, USA

ARTICLE INFO

Article history:
Received 11 August 2008
Received in revised form 13 March 2009
Accepted 13 March 2009
Available online 25 March 2009

Keywords:
Sequential pattern
Constraint-based mining
RFM
Segmentation

ABSTRACT

In response to the thriving development in electronic commerce (EC), many on-line retailers have developed Web-based information systems to handle enormous amounts of transactions on the Internet. These systems can automatically capture data on the browsing histories and purchasing records of individual customers. This capability has motivated the development of data-mining applications. Sequential pattern mining (SPM) is a useful data-mining method to discover customers' purchasing patterns over time. We incorporate the recency, frequency, and monetary (RFM) concept presented in the marketing literature to define the RFM sequential pattern and develop a novel algorithm for generating all RFM sequential patterns from customers' purchasing data. Using the algorithm, we propose a pattern segmentation framework to generate valuable information on customer purchasing behavior for managerial decision-making. Extensive experiments are carried out, using synthetic datasets and a transactional dataset collected by a retail chain in Taiwan, to evaluate the proposed algorithm and empirically demonstrate the benefits of using RFM sequential patterns in analyzing customers' purchasing data.

1. Introduction

Electronic commerce (EC) is generally defined as the process of buying, selling, or exchanging products, services, and information via computer networks including the Internet (Turban et al. 2006). In response to the thriving development in EC, many on-line retailers have developed Web-based information systems to handle enormous amounts of transactions on the Internet. These systems can automatically capture data on the browsing histories and purchasing records of individual customers. As a result, many data-mining applications have been developed to discover useful customer and market information from the data, such as Web merchandising (Lee et al. 2001), Web site design improvement (Spiliopoulou and Pohle 2001), click stream analysis (Lee et al. 2001), product recommendation (Lawrence et al. 2001, Schafer et al. 2001), e-retailing (Chen et al. 2005, Lin et al. 2003, Tang et al. 2008), cross-selling (Lin et al. 2003), customer profiling (Hu and Chen 2008, Mahdavi et al. 2008), and e-catalog design (Lin and Hong 2008).

Sequential pattern mining (SPM) is particularly useful in EC applications, because it can be used to discover customers' behavioral and purchasing patterns over time. The methodology was first introduced by Agrawal and Srikant (1995), which is described as follows. Consider a dataset consisting of "data-sequences", which are lists of items purchased by individual customers over time. The goal of SPM is to find all the frequent subsequences in the dataset. For example, the method can be used to find the frequent browsing patterns on a Web site, which are useful in evaluating the design of the Web site.

In this study, we focus on mining sequential purchasing patterns for the e-retailing industry. Existing studies in this area often focus on finding sequential patterns based on the frequency with which these patterns occur in the data. For example, a study may find that a significant number of customers purchase a memory card, an Internet cable, and a computer game in sequence. In practice, however, customers' purchasing behaviors may change over time for different reasons; for example, because of a change in the economic environment. As a result, it is possible that a past purchasing pattern may not be present again. Furthermore, a pattern may not be important to the retailer when the product items included in the pattern have low values (e.g., prices or profits). Without considering the monetary value in selecting sequential patterns, retailers may be overwhelmed by a large number of low-value patterns.

To address these issues, we incorporate the concept of recency, frequency, and monetary (RFM) introduced by Bult and Wansbeek (1995) in SPM. We define *recency* as the period since a customer's last purchase; *frequency*, as the number of purchases made within a

* Corresponding author. Tel.: +886 3 4267266; fax: +886 3 4254604.
E-mail addresses: ylchen@mgt.ncu.edu.tw (Y.-L. Chen), 944203006@cc.ncu.edu.tw (M.-H. Kuo), sywu@itri.org.tw (S.-Y. Wu), ktang@purdue.edu (K. Tang).

certain period; and *monetary* (value), as the amount of money that a customer spent during a certain period. Using these three selection criteria properly helps retailers focus on important purchasing patterns.

We use two thresholds, upper and lower, to form a constraint for each selection criterion. The thresholds for recency, respectively denoted by *Rtime_min* to *Rtime_max*, are the earliest and latest times from a given starting time for the last transaction of a selected pattern. For example, if we set *Rtime_min* = 200 and *Rtime_max* = 270, the last transaction of a selected pattern must occur between 200 and 270 days from the starting date. Similarly, *M_min* and*M_max* are the upper and lower thresholds for the monetary constraint, ensuring that the value of the selected pattern is between *M_min* and *M_max*. The frequency of a pattern is the percentage of sequences that satisfy the recency and monetary constraints. A pattern is an RFM pattern if its frequency falls within *minsup_min* and *minsup_max*. By setting these three constraints, we may discover RFM-patterns such as, "30% of customers who recently bought a computer will return to buy a scanner and a microphone, and the total value will exceed NT$ 50,000". Retailers can adjust these three constraints according to their needs. Note that, if a retailer is interested only in the traditional sequential patterns, it can relax the recency and monetary constraints. Similarly, a retailer can relax the monetary constraint to focus on the other two constraints, if monetary is not a concern.

We develop an efficient algorithm for finding RFM sequential patterns from customers' data-sequences. The algorithm also allows us to partition the RFM-patterns into segments relevant to the RFM criteria. By comparing, contrasting, and aggregating the patterns in the segments, retailers can identify patterns that require special attention and action. For example, a retailer may find patterns with a large monetary value, which appear just recently. In this case, the retailer could devote more resources to promotion, inventory, etc., associated with the items in those patterns. Furthermore, a retailer may find some patterns with decreasing frequencies or monetary values in recent time, which may cause the retailer to adjust marketing and/or product strategies.

The rest of this paper is organized as follows. We give a literature review of SPM in Section 2 and formally define the problem and RFM-patterns in Section 3. In Section 4, we develop the RFM-Apriori algorithm for finding all RFM-patterns. Experimental results for evaluation are given in Section 5, and our conclusions are presented in Section 6.

2. Related work

SPM is a useful method of extracting sequential patterns with a support level exceeding a predefined minimal threshold. The method has been widely used in areas such as mining user access patterns on Web sites, using the history of symptoms to predict potential disease, recommend products to users, and adjust the structure of Web sites. Business organizations can also use SPM to study customer behaviors and provide better management of customer relationships.

Existing research on SPM can be divided into two main categories: (1) improving the efficiency of the mining process (Pei et al. 2000, 2001; Srikant and Agrawal 1996, Zaki 2001); and (2) proposing additional time-related patterns, such as frequent episodes, traversal patterns, cyclic patterns, periodic patterns, multiple-dimensional sequential patterns, hybrid sequential patterns, time-interval sequential patterns, and fuzzy sequential patterns (Chen et al. 1998, 2001, 2003; Chen and Huang 2005, Han et al. 1999, Mannila et al. 1997, Toroslu 2003, Yu and Chen 2005). These studies focus on the frequency in selecting sequential patterns.

Several researchers have considered RFM variables in developing prediction and classification models. For example, a Bayesian Networks approach has been proposed, using RFM variables to predict a customer's response to direct marketing (Cui et al. 2006). Additional examples include data-mining models for predicting customer loyalty (Cheng and Chen 2009) and customer lifetime value (Etzion et al. 2005), and classifying customers in their profitability. Applications based on RFM concepts have also been proposed for the mobile communications industry (Mozer et al. 2000, Weiss 2005), including churn prediction, customer retention, and cross-selling.

To the best of our knowledge, however, this paper is the first in applying the RFM criterion in SPM. As discussed in the Introduction, all the three criteria could be very important to an online retailer, which motivates this research.

3. Problem definition

We represent a customer's data sequence by $A = \langle(a_1,t_1,m_1), (a_2,t_2,m_2),\ldots,(a_n,t_n,m_n)\rangle$, where (a_j,t_j,m_j) indicates that item a_j was purchased at time t_j with a total value of $m_j, 1 \leqslant j \leqslant n$, and $t_{j-1} \leqslant t_j$ for $2 \leqslant j \leqslant n$. If items occur at the same time in the data sequence, they are ordered alphabetically.

Based on this data format, we have the following definitions.

Definition 1 (*Containment of itemset*). Let I denote the set of all items in the database. Let $A = \langle(a_1,t_1,m_1), (a_2,t_2,m_2),\ldots,(a_n,t_n,m_n)\rangle$ be a data sequence where $t_1 \leqslant t_2 \leqslant,\ldots,\leqslant t_n$, and $I_q = (i_1 i_2,\ldots,i_m)$ be an *itemset* where $i_p \in I$ $(1 \leqslant p \leqslant m)$. Itemset I_q is *contained* in A if there are m integers $1 \leqslant k_1 < k_2 < \ldots < k_m \leqslant n$ such that $i_1 = a_{k_1}$, $i_2 = a_{k_2},\ldots,i_m = a_{k_m}$ and $t_{k_1} = t_{k_2} =,\ldots,= t_{k_m}$. We refer to k_1 and t_{k_1} as the position and the time at which I_q occurs in A, respectively.

Definition 2 (*Subsequence*). Following Definition 1, let $B = \langle I_1 I_2,\ldots, I_s\rangle$ be a sequence of itemsets, where each $I_q \subseteq I$, $(1 \leqslant q \leqslant s)$, is an itemset. Then, sequence B is *contained* in A (meaning B is a *subsequence* of A), if the following conditions are satisfied: (1) each I_q in B is *contained* in A, and (2) $t_{I_1} < t_{I_2} < \ldots < t_{I_s}$, where t_{Iq} $(1 \leqslant q \leqslant s)$ is the time at which I_q occurs in A. The length of a subsequence B is the number of itemsets in B (*length* $(B) = s$).

Example 1. (Subsequence) Itemset (ab) is contained in data sequence $A = \langle(a, 1, 10), (c, 3, 40), (a, 4, 30), (b, 4, 70), (a, 6, 50), (e, 6, 90), (c, 10, 70)\rangle$, because both items a and b occur in A at time 4. The sequence $\langle(ab)(ae)\rangle$ is a subsequence of A because itemset (ab) occurs in A at time 4 and (ae) occurs at time 6.

Definition 3 (*Recent subsequence*). Following Definition 2, assume that t_{Iq} is the time at which I_q $(1 \leqslant q \leqslant s)$ occurs in A, and *Rtime_min*, *Rtime_max* are the user-specified recency thresholds. B is a **recent** *subsequence* of A if and only if we can find a subsequence B from A that satisfies the recency constraint ($t_{Is} \geqslant Rtime\_min$ and $t_{Is} < Rtime\_max$).

Example 2. (Recent subsequence) Let $A = \langle(a, 1, 10), (c, 3, 40), (a, 4, 30), (b, 4, 70), (a, 6, 50), (e, 6, 90), (c, 10, 70)\rangle$ be a data sequence, *Rtime_min* = 5, and *Rtime_max* = 8. Then, sequence $\langle(c)(b)(ae)\rangle$ is a **recent subsequence** of A because $\langle(c)(b)(ae)\rangle$ is a subsequence of A and the occurrence time of itemset $(ae) = 6 \geqslant Rtime\_min$ and $6 < Rtime\_max$.

Definition 4 (*Recent monetary subsequence*). Following Definition 3, assume that m_{I_q} is the total value of I_q $(1 \leqslant q \leqslant s)$ in A, and *M_min*, *M_max* are the user-specified monetary thresholds. B is called a **recent monetary subsequence** of A if and only if we can find a subsequence B from A such that (1) B is a recent subsequence of A, and (2) the monetary constraint is satisfied ($M\_min \leqslant m_{I_1} + m_{I_2} + \ldots. + m_{I_s} < M\_max$).

Example 3. (Recent monetary subsequence) Let $A = \langle(c, 5, 30), (b, 6, 40), (a, 10, 30), (b, 17, 60), (b, 19, 90), (c, 19, 70)\rangle$ be a data sequence, $Rtime\_min = 8$, $Rtime\_max = 20$, $M\_min = 170$, and $M\_max = 220$. According to Definition 4, $\langle(c)(b)(c)\rangle$ is a recent monetary subsequence of A because we can find a subsequence ($\langle(c, 5, 30), (b, 19, 90), (c, 19, 70)\rangle$) from A that is a recent subsequence of A and whose total value is 190, satisfying $M\_min \leqslant 190 < M\_max$.

Definition 5. (F pattern, RF pattern, and RFM pattern) Let $B = \langle I_1 I_2 \ldots I_s\rangle$ be a sequence of itemsets. If the percentage of data-sequences in the database containing B as a subsequence, called *f*-support and denoted $B.sup^f$, is no less than *minsup_min*, we call B an F pattern. B is an RF pattern if the percentage of data-sequences in the database containing B as a recent subsequence, called *rf*-support and denoted $B.sup^{rf}$, is no less than *minsup_min*. Finally, B is an RFM pattern if the percentage of data-sequences in the database containing B as a recent monetary subsequence, called *rfm*-support and denoted $B.sup^{rfm}$, is between *minsup_min* and *minsup_max*.

4. Algorithms

In this section, we introduce the RFM-Apriori algorithm, which is designed to find sequential patterns from customers' data-sequences. The algorithm is developed by modifying the well-known Apriori (GSP) algorithm (Srikant and Agrawal 1996). Before giving the details of the algorithm, we provide a general description of the algorithm and compare it with GSP.

The GSP algorithm consists of iterative phases. First, it puts all items into C_1, the set of candidate F patterns with length 1, and then scans the database to find the frequent 1-patterns (L_1). Second, suppose we already have the set of frequent $(k-1)$-patterns L_{k-1}. It generates the set of candidate F patterns C_k by joining L_{k-1} with L_{k-1}. Afterward, it scans the database to determine the supports of the patterns in C_k, and then finds L_k by removing those patterns from C_k with supports lower than the minimum support. We repeat this phase, increasing k by one, until no more patterns can be generated.

RFM-Apriori differs from the Apriori (GSP) algorithm in the following ways:

(1) *Candidate generation*: In RFM-Apriori, we also generate candidate patterns from the set of frequent patterns with shorter length. Let CI_k denote the set of candidate RF patterns with length k in RFM-Apriori. Then, we have the following four differences between the two algorithms: (a) In GSP, each phase expands the patterns by one item, but, in RFM-Apriori, each phase expands the patterns by one itemset; (b) CI_1 is generated from the set of all frequent itemsets found by the traditional Apriori algorithm; (c)CI_2 is generated by joining LI_1^f and LI_1^{rf}, where LI_1^f is the set of F patterns with length 1 and LI_1^{rf} is the set of RF patterns with length 1; and (d) CI_k, where $k > 2$, is generated by joining LI_{k-1}^{rf} and LI_{k-1}^{rf}, where LI_{k-1}^{rf} is the set of RF patterns with length $k-1$.

(2) *Support counting*: The traditional GSP algorithm computes the support $B.sup^f$ for a candidate pattern B in C_k. RFM-Apriori, however, computes $B.sup^{rf}$ and $B.sup^{rfm}$ for each pattern B in CI_k. An inverse candidate tree, a structure that stores all patterns in CI_k, is used to speed up the support-counting process.

An overview of the algorithm is given in Fig. 1. First, we include all frequent itemsets found by the traditional Apriori in LI_1^f. Then, we scan the database to determine their *rf*-supports and *rfm*-supports. Since this procedure is similar to the support-counting procedure discussed in Section 4.2, we omit the explanation here. After this, we obtain LI_1^{rf} and LI_1^{rfm}. Next, we perform an iterative process of candidate generation and support counting, just like the traditional GSP. These two components will be introduced in Sections 4.1 and 4.2, respectively. Section 4.3 provides a simple yet complete example to illustrate the algorithm. Finally, we analyze the time complexity of the algorithm in Section 4.4.

```
Method: Call RFM-Apriori
Input: $D$: Transferred data sequence database;
        $minsup\_min, minsup\_max$: The threshold of support given by the user
        $Rtime\_min, Rtime\_max$: The constraints of recency given by the user
        $M\_min, M\_max$: The constraints of monetary given by the user
Output: $LRFM$: The set of all large RFM patterns
Procedure RFM-Apriori ($D, minsup\_min, minsup\_max, Rtime\_min,$
                        $Rtime\_max, M\_min, M\_max$);
{
    $CI_1$= Apriori ( );
    Scan the database to determine $LI_1^f$, $LI_1^{rf}$ and $LI_1^{rfm}$
    $LRFM = LI_1^{rfm}$
    For ($k = 2$; $LI_{k-1}^{rf} \neq \varnothing$; $k$++) {
        If $k = 2$ then
            $CI_2 = candidate\_gen\ (LI_1^f, LI_1^{rf})$;
        Else
            $CI_k = candidate\_gen\ (LI_{k-1}^{rf}, LI_{k-1}^{rf})$;

        //scan $D$ to compute support counts of $CI_k$
        Build the inverse candidate tree from $CI_k$.
        For each sequence $s \in D$ do
        {
            Traverse the tree to accumulate the supports.
        }

        $LI_k^{rf} = \{c \in CI_k \mid c.sup^{rf} \geq minsup\_min\}$
        $LI_k^{rfm} = \{c \in CI_k \mid c.sup^{rfm} \geq minsup\_min \text{ and } c.sup^{rfm} < minsup\_max\}$
        $LRFM = LRFM \cup LI_k^{rfm}$
    }

    Output $LRFM$;
}
```

Fig. 1. The RFM-Apriori algorithm overview.

4.1. Candidate generation

There are two major differences between the proposed and traditional methods with regard to candidate generation. First, we use an itemset as a unit to expand the patterns, rather than just an item. The advantage of using an itemset is that it can reduce the number of phases needed to complete the algorithm, thus improving efficiency. Second, we use a different method to generate CI_k, where $k \geqslant 2$. We join two frequent patterns of length $k-1$ to generate candidate RF patterns of length k, where $k \geqslant 2$, if they have the same $(k-2)$-postfix. Next, we explain how we generate CI_2 and CI_k in detail.

We generate CI_2 by joining LI_1^f and LI_1^{rf}. In other words, if $B_1 = \langle I_a\rangle$, $B_2 = \langle I_b\rangle$, where $B_1 \in LI_1^f$, $B_2 \in LI_1^{rf}$, then we will have $\langle I_a, I_b\rangle$ in CI_2. The following lemma shows that CI_2 contains all candidate RF patterns of length 2. The proofs are given in Appendix A.

Lemma 1. *All patterns in LI_2^{rf} must exist in CI_2.*

Lemma 2. *If pattern B is an RF pattern (B is in LI_k^{rf}), then a subsequence of B with the same last itemset as B must also be an RF pattern.*

For example, if $\langle(a)(cd)(e)\rangle$ satisfies the recency constraint, then $\langle(a)(e)\rangle$, $\langle(cd)(e)\rangle$, and $\langle(e)\rangle$ also satisfy the recency threshold. $\langle(a)(cd)\rangle$ may not, however, satisfy the recency constraint.

Lemma 3. *All patterns in LI_k^{rf} ($k > 2$) must exist in CI_k.*

Example 4. Suppose we have $LI_1^f = \{\langle a\rangle, \langle b\rangle, \langle c\rangle, \langle(ab)\rangle, \langle(bc)\rangle\}$ and $LI_1^{rf} = \{\langle b\rangle, \langle c\rangle\}$. Then, $CI_2 = \{\langle(a)(b)\rangle, \langle(a)(c)\rangle, \langle(b)(b)\rangle, \langle(b)(c)\rangle, \langle(c)(b)\rangle, \langle(c)(c)\rangle, (ab)(b)\rangle, \langle(ab)(c)\rangle, (bc)(b)\rangle, \langle(bc)(c)\rangle\}$. Further assume that we have $LI_3^{rf} = \{\langle(b)(a)(c)\rangle, \langle(c)(a)(c)\rangle, \langle(b)(b)(c)\rangle, \langle(c)(b)(c)\rangle, \langle(b)(ab)(c)\rangle, \langle(c)(ab)(c)\rangle\}$. Then, $CI_4 = \{\langle(b)(b)(a)(c)\rangle, \langle(b)(c)(a)(c)\rangle, \langle(c)(b)(a)(c)\rangle, \langle(c)(c)(a)(c)\rangle, \langle(b)(b)(b)(c)\rangle, \langle(b)(c)(b)(c)\rangle, \langle(c)(b)(b)(c)\rangle, \langle(c)(c)(b)(c)\rangle, \langle(b)(b)(ab)(c)\rangle, \langle(b)(c)(ab)(c)\rangle, \langle(c)(b)(ab)(c)\rangle, \langle(c)(c)(ab)(c)\rangle\}$.

4.2. Counting supports by traversing an inverse candidate tree

To count supports, we use an inverse candidate tree to store all candidate patterns in CI_k, where a leaf node corresponds to a candidate pattern. Using every data sequence to traverse the tree, we can accumulate support values in each leaf node. This is an efficient method of determining whether a candidate pattern satisfies the recency constraint. Fig. 2 shows how each data sequence traverses the tree and accumulates supports in leaf nodes. Basically, this traversal procedure is a recursive program by which we can match all subsequences in T with all candidate patterns in CI_k. If we can find a matched subsequence that satisfies both the recency and monetary constraints for a pattern (leaf node), we increase the *rfm*-support and *rf*-support of this pattern by one. If it satisfies only the recency constraint, however, we increase only the *rf*-support by one. In the following subsections, we discuss the details of the inverse candidate tree, but omit the procedure of support counting for simplicity.

```
Procedure: Traverse (u, T, i, et, totalmoney)
Parameters:
u: the node in the candidate tree where we are currently located
T: the data-sequence that we are dealing with
i: the position in the sequence that the preceding traversal matches
et: the time of item q in T, where q is the first internal node in the path from the
    root to node u
item(u, v): the item of arc (u, v)
item (T(j)): the item of T(j), the j-th item in T
time (T(j)), money(T(j)): the time stamp and the monetary value of T(j)
totalmoney: the total account of candidate itemsets in a data-sequence
Method:
if u is a leaf node then
  if et >=Rtime_min and et < Rtime_max
    and totalmoney>=M_min and totalmoney <M_max then
      add 1 into the sup^rfm and sup^rf counters of the leaf node
  else
    if et >=Rtime_min and et < Rtime_max then
        add 1 into the sup^rf counter of the leaf node
  return
else   /* we traverse the tree to find the patterns occurred in T,
        starting from the last item of T towards the beginning*/
  for ( j = i; j < T.length; j++ )
      for each child v of u
         if u is the root node then
            if time(T(j))<Rtime_min then
                return;
            if item(u, v ) = item(T( j ))
               and time(T(j)) >=Rtime_min and time(T(j))<Rtime_max
            then
                Traverse (v, T, j+1, time(T(j)), totalmoney+money(T(j)))
         else
            if item(u, v) = item(T(j)) then
                Traverse (v, T, j+1, et, totalmoney+money(T(j)))
Return
```

Fig. 2. Procedure *Traverse* $(u, T, i, et, totalmoney)$.

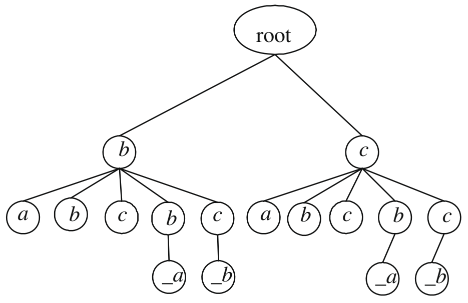

Fig. 3. The inverse candidate tree.

4.2.1. Inverse candidate tree

This tree is constructed by inserting the itemsets in all candidate patterns of CI_k into an empty tree in reverse order. For example, for $CI_2 = \{\langle(a)(b)\rangle, \langle(a)(c)\rangle, \langle(b)(b)\rangle, \langle(b)(c)\rangle, \langle(c)(b)\rangle, \langle(c)(c)\rangle, (ab)(b)\rangle, \langle(ab)(c)\rangle, (bc)(b)\rangle, \langle(bc)(c)\rangle\}$, the constructed inverse candidate tree is shown in Fig. 3, where a hyphen preceding an item indicates that it is in the same itemset as its parent item.

Now, we use an example to illustrate why the inverse tree can improve efficiency in support counting. Suppose we have a candidate pattern $\langle(a)\,(c)\rangle$, a data sequence $T = \langle(a, 1, 10), (c, 3, 40), (a, 4, 30), (b, 4, 70), (a, 6, 50), (c, 10, 70)\rangle$, *Rtime_min* = 15, and *Rtime_max* = 20. In a normal recursive support-counting procedure, the subsequences in T would be examined in this order: $\langle(a, 1, 10), (c, 3, 40)\rangle$, $\langle(a, 1, 10), (c, 10, 70)\rangle$, $\langle(a, 4, 30), (c, 10, 70)\rangle$, $\langle(a, 6, 50), (c, 10, 70)\rangle$. Since we are using an inverse candidate tree, however, we have to invert the data sequence's examination order. As a result, the subsequences in T would be examined in this order: $\langle(c, 10, 70), (a, 6, 50)\rangle$, $\langle(c, 10, 70), (a, 4, 30)\rangle$, $\langle(c, 10, 70), (a, 1, 10)\rangle$, $\langle(c, 3, 40), (a, 1, 10)\rangle$. We now investigate the differences between these two approaches.

- *Traverse in normal order.* In this case, all four subsequences in T must be checked in order to ensure that $\langle(a)(c)\rangle$ does not satisfy the recency constraint.
- *Use the inverse candidate tree.* In this case, the data sequence is examined from back to front. When we check $(c, 10, 70)$, we find that the recency constraint is violated because 10 < *Rtime_min*. This means we can skip all succeeding subsequences because their times cannot be greater than 10. Therefore, we have discovered a convenient property of the inverse candidate tree; if one itemset in the sequence occurs earlier than *Rtime_min*, then this subsequence and all of its succeeding subsequences will not satisfy the recency constraint.

Table 1
RFM-Apriori example data sequence.

Sid	Sequence
10	$\langle(a, 1, 10), (c, 3, 40), (a, 4, 30), (b, 4, 70), (a, 6, 50), (c, 10, 70)\rangle$
20	$\langle(b, 3, 30), (c, 5, 50), (a, 7, 20), (b, 7, 70), (c, 14, 20)\rangle$
30	$\langle(a, 8, 40), (b, 8, 50), (b, 16, 20), (c, 20, 100)\rangle$
40	$\langle(b, 15, 30), (b, 22, 20), (c, 22, 120)\rangle$
50	$\langle(c, 5, 30), (b, 6, 40), (a, 10, 30), (b, 10, 60), (b, 19, 90), (c, 19, 70)\rangle$

4.3. A complete example

Consider a data sequence *DB* given in Table 1 and six thresholds, *Rtime_min* = 10, *Rtime_max* = 21, *M_min* = 150, *M_max* = 250, *minsup_min* = 2, and *minsup_max* = 4. The goal is to find all RFM-patterns.

The proposed algorithm produces the following results:

- $LI_1^f = \{\langle a\rangle, \langle b\rangle, \langle c\rangle, \langle (ab)\rangle, \langle (bc)\rangle\}$, $LI_1^{rf} = \{\langle b\rangle, \langle c\rangle\}$, $LI_1^{rfm} = \{\}$
- $LI_2^{rf} = \{\langle (a)(b)\rangle, \langle (a)(c)\rangle, \langle (b)(b)\rangle, \langle (b)(c)\rangle, \langle (c)(c)\rangle, \langle (ab)(b)\rangle, \langle (ab)(c)\rangle\}$, $LI_2^{rfm} = \{\langle (ab)(c)\rangle\}$
- $LI_3^{rf} = \{\langle (b)(a)(c)\rangle, \langle (a)(b)(c)\rangle, \langle (c)(a)(c)\rangle, \langle (b)(b)(c)\rangle, \langle (c)(b)(c)\rangle, \langle (b)(ab)(c)\rangle, \langle (c)(ab)(c)\rangle\}$,
  $LI_3^{rfm} = \{\langle (a)(b)(c)\rangle, \langle (b)(b)(c)\rangle, \langle (c)(b)(c)\rangle, \langle (c)(ab)(c)\rangle\}$
- $LI_4^{rf} = \{\langle (c)(b)(a)(c)\rangle\}$, $LI_4^{rfm} = \{\langle (c)(b)(a)(c)\rangle\}$

4.4. Time complexity

The following theorem gives the time complexity of the algorithm. Its proof is given in Appendix A.

Theorem 1. *The time complexity of the RFM-Apriori is* $O(((m \times C_R^n \times R) + (\rho \times R)) \times K)$, *where* ρ *is* $\max_k |CI_k|$, *m the number of data-sequences in the database, n the maximum number of items in a data sequence, R the maximum number of items in a candidate pattern, and K the total number of iterations in the algorithm.*

5. Performance evaluation

The three algorithms, GSP, RFM-Apriori and R^{A}FM-Apriori, have been implemented in Java language on a Pentium M 1.73 GHz Windows XP system with 1 gigabyte of main memory. Note that R^{A}FM-Apriori and RFM-Apriori are the same except that we measure the recency of a pattern by taking the average of recency values for all itemsets in its sequence. For example, assume we have pattern $\langle (c)(b)(ae)\rangle$. Then, the average recency value of this pattern is the average time values of itemsets (c), (b), and (ae) in data sequences. We ignore the time stamp and the money attribute in transactions in applying the GSP algorithm, because it is designed for mining sequential patterns without recency and monetary considerations.

The content of this section is divided into four parts. Section 5.1 describes the synthetic datasets and the real dataset used in the experiments. In Section 5.2, we compare the running time, monetary value, recency, and the number of patterns between GSP and RFM-Apriori. We further conduct a scalability analysis for the two algorithms. In Section 5.3, we first compare the running time and number of patterns generated by RFM-Apriori and R^{A}FM-Apriori. Then, we compare the number of patterns generated by these two algorithms with different time windows. Finally, in Section 5.4, we propose a pattern segmentation method for generating valuable information for managerial decision-making.

5.1. Synthetic data generation and real-life dataset

Synthetic datasets are generated using Agrawal and Srikant (1995) synthetic data- generation algorithm. Each transaction contains a sequence of itemsets; however, the transaction data are extended so that items in different itemsets are assigned different time values and items in the same itemsets are assigned the same time values. All items are divided into three price levels: high, medium, and low. An item's price is determined by drawing a value from a Poisson distribution with mean *H_price*, *M_price*, and *L_price*. We also assume that the quantity of an item purchased in a transaction is inversely related to its price. According to this assumption, the purchased quantities of high-, medium-, and low-priced items are drawn from a Poisson distribution with mean *H_quantity*, *M_quantity*, and *L_quantity*, respectively. A detailed description of the datasets is given in Appendix B.

Table 2 contains the list of parameters used in the simulation; the first eight parameters have often been used in previous research, but the other parameters are new ones created for our study. In the simulation, these parameters are fixed: $|S| = 4$, $|I| = 1.25$, $N_S = 5000$, $N_I = 25{,}000$, $N = 10000$, $T_I = 10$, *H_price* = 1000, *M_price* = 500, *L_price* = 100, *H_quantity* = 1, *M_quantity* = 3, and *L_quantity* = 5. The parameter settings for the nine synthetic datasets are shown in Table 3.

We include a real dataset, called SC-POS, which is the sales records of a supermarket chain in Taiwan. The SC-POS dataset recorded all transactions from twenty branches between 12/27/2001 and 12/31/2002. Each data sequence in SC-POS is the shopping list of a customer's transaction, which recorded the purchase times, prices, and quantities. After performing a series of data preprocessing and cleaning tasks, the final dataset contains 17,685 product items and 33,500 customers' data-sequences. For additional information, please see Appendix B.

5.2. The comparison between algorithms GSP and RFM-Apriori

The first comparison is based on the nine synthetic datasets shown in Table 3 and a real dataset, SC-POS. We compare the runtimes of the two algorithms by varying *minsup_min* from 0.5% to 1.25% in the synthetic datasets and from 3.5% to 2.5% in the real-life dataset. Because of the space limitation, we report only the results from dataset C10-T2.5-D25k; the results from the other eight synthetic datasets are similar. As shown in Tables 4 and 5, the proposed method takes slightly less time and generates much smaller numbers of patterns. These results are expected, because our method is designed to find only the patterns that satisfy the recency and monetary criteria. Nevertheless, the comparisons

Table 2
The parameters of synthetic datasets.

$|D|$: Number of customers
$|C|$: Average number of transactions per customer
$|T|$: Average number of items per transaction
$|S|$: Average length of maximal potentially large sequences
$|I|$: Average size of itemsets in maximal potentially large sequences
N_S: Number of maximal potentially large sequences
N_I: Number of maximal potentially large itemsets
N: Number of items
T_I: Average length of time intervals
H_price: Average price of high-priced items
M_price: Average price of medium-priced items
L_price: Average price of low-priced items
H_quantity: Average purchased quantity of high-priced items
M_quantity: Average purchased quantity of medium-priced items
L_quantity: Average purchased quantity of low-priced items

Table 3
The test synthetic datasets.

Name	$|C|$	$|T|$	$|D|$
SYN-DS1	10	2.5	250k
SYN-DS2	10	2.5	500k
SYN-DS3	10	2.5	750k
SYN-DS4	10	2.5	250k
SYN-DS5	15	2.5	250k
SYN-DS6	20	2.5	250k
SYN-DS7	10	2.5	250k
SYN-DS8	10	3.5	250k
SYN-DS9	10	4.5	250k

Table 4
Running time and number of patterns vs. *minsup_min* for dataset SYN-DS1.

	Running time (s)		Number of patterns	
Minsup	RFM	GSP	RFM	GSP
0.0050	150	172	291	2717
0.0075	100	81	48	835
0.0100	78	55	30	336
0.0125	43	48	4	156

Table 5
Running time and number of patterns vs. *minsup_min* for dataset SC-POS.

	Running time (s)		Number of patterns	
Minsup	RFM	GSP	RFM	GSP
Minsup	RFM	GSP	RFM	GSP
0.025	70	16692	4	592
0.030	50	2472	3	426
0.035	41	2454	2	303

suggest a very large number of patterns in the datasets, which are potentially not useful to the retailer.

We use Table 6 to compare the average monetary values and number of patterns (in parentheses) produced by RFM-Apriori and GSP under selected values of *M_Min* and *Minsup_min*. The results show that RFM-Apriori can effectively identify small numbers of high-values patterns.

Similarly, we use Table 7 to compare the recency value and number of patterns (in parentheses) produced by RFM-Apriori and GSP under selected values of recency ranges, while the *M_Min* value is kept at 50. As expected, the average recency value increases as the lower limit of recency increases. The results also show that RFM-Apriori can effectively identify more recent patterns as the limit is specified as a retailer.

Next, the scalabilities of the two algorithms are compared. Three tests are designed, and the dataset *C*10-*T*2.5-*S*4-*I*1.25-*D*250K is treated as the baseline. During the tests, we vary the value of a selected parameter and keep all the other parameters constant. In each test, however, a parameter is increased to determine how the algorithms scale up as the parameter increases. The first test varies the number of customers, $|D|$, from 250,000 to 750,000; the second varies the average number of transactions per customer, $|C|$, from 10 to 20; and the final test varies the average number of items bought per transaction, $|T|$, from 2.5 to 4.5. We set *minsup_min* at 0.5%, 1%, and 0.8% for $|D|$, $|C|$, and $|T|$, respectively. The results (Figs. 4–6) indicate that the runtimes and numbers of patterns of these two algorithms scale up linearly with $|D|$, but exponentially with $|C|$ and $|T|$. It is interesting to note, however, that the difference in the numbers of patterns is much larger than that of runtimes. This is because RFM-Apriori trims many uninteresting patterns at the cost of more complicated processing. As a result, although the number of patterns in RFM-Apriori is only one-fifth of GSP's, the difference in runtime is not as large. We also notice that, as the parameter value increases, the difference in runtimes between the two algorithms also increases. This phenomenon also occurs for $|C|$ and $|T|$ with regard to the number of patterns. Since *C* and *T* determine the length of a data sequence, this would indicate that longer sequences would result in more patterns. This does not hold for $|D|$, however, because the number of generated patterns remains the same for all values of $|D|$. Since $|D|$ stands for the number of sequences in a database, this result implies that the patterns found from a subset of data sequences in *D* are a good approximation of the results mined from all sequences in *D*.

Table 6
A comparison between RFM and GSP in the average monetary value and the number of patterns.

Minsup_min	RFM			GSP
	M_Min			
	50	75	100	
0.027	169 (71)*	201 (52)	232 (39)	75 (509)
0.030	176 (46)	218 (33)	251 (25)	73 (426)
0.033	166 (34)	211 23	232 (19)	70 (338)

* The numbers of patterns are reported in parentheses.

Table 7
A comparison between RFM and GSP in recency.

Minsup_min	RFM			GSP
	Recency			
	80–360	160–360	240–360	
0.027	221 (129)*	256 (83)	294 (23)	188 (509)
0.030	221 (105)	252 (58)	295 (18)	187 (426)
0.033	219 (80)	252 (58)	294 (14)	184 (338)

* The numbers of patterns are reported in parentheses.

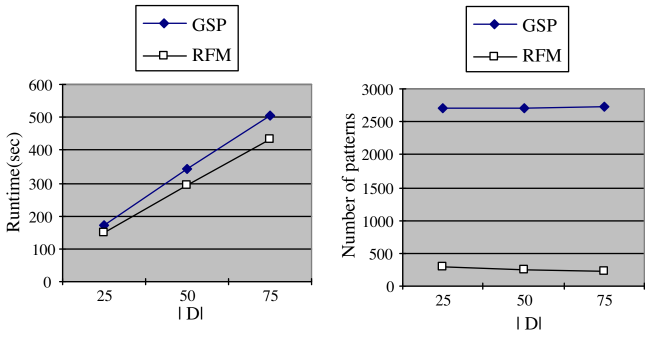

Fig. 4. Scalability test with respect to the number of customers.

The following two tests are designed to evaluate how the recency and monetary constraints influence the proposed algorithm's runtime and number of patterns. All of the tests are based on the *C*20-*T*2.5-*S*4-*I*1.25-*D*250K dataset, and we vary only the value of a single parameter, *Rtime_min* or *M_min*, keeping all other parameters constant. The results in Fig. 7 show that, as *Rtime_min* increases, both the runtime and the number of patterns of the RFM-Apriori algorithm decrease. This is a reasonable result because, if *Rtime_min* increases, it becomes more difficult for a pattern to satisfy the recency constraint. Fig. 8 shows how the runtime and number of patterns change as we vary the value of *M_min*. Similarly, as *M_min* increases, the number of patterns decreases, because it is more difficult for a pattern to satisfy the monetary constraint. The runtime, however, remained constant at different values of *M_min*. This may be because the most time-consuming part of our algorithm is generating CI_k from LI^{rf}_{k-1}. After obtaining CI_k, the support-counting procedure is executed to determine LI^{rf}_k and LI^{rfm}_k. In other words, no matter how tight the monetary constraint is, we cannot reduce the resources needed to generate CI_k and obtain LI^{rf}_k, which are the most time-consuming parts of the algorithm. Therefore, we cannot save runtime by imposing tight monetary constraints.

Next, we examine various types of patterns generated from the nine synthetic datasets and the real-life dataset. In this investigation, we consider only three types of patterns, RFM-patterns, RF patterns, and F patterns. F patterns are traditional sequential pat-

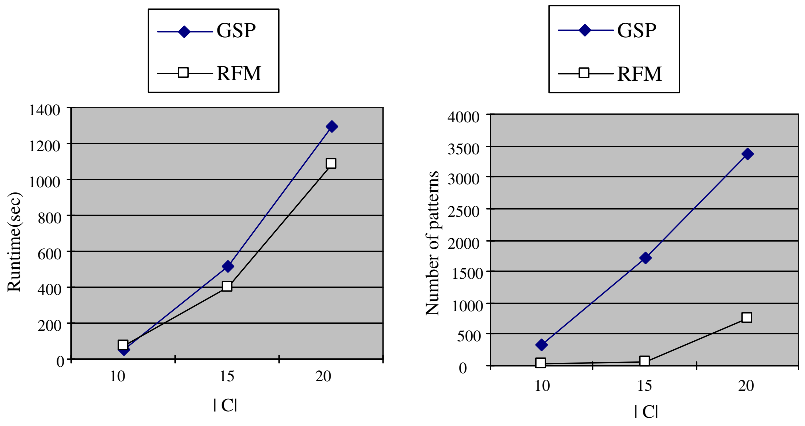

Fig. 5. Scalability test with respect to the average number of transactions per customer.

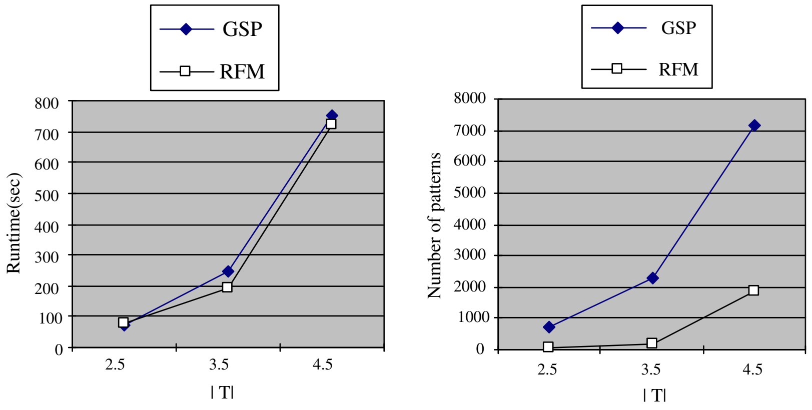

Fig. 6. Scalability test with respect to the average number of items per transaction.

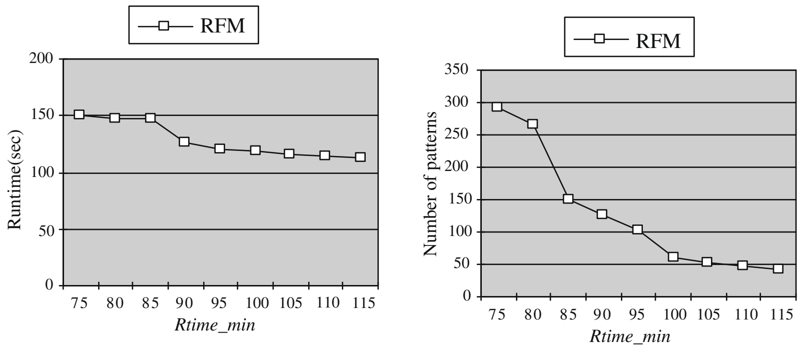

Fig. 7. Runtime and number of patterns vs. *Rtime_min*.

terns, RF patterns are patterns that are subject to the recency constraint, and RFM-patterns are subject to both monetary and recency constraints. Table 8 shows the percentage of F patterns that are RF patterns and RFM-patterns. The results from the SC-POS dataset indicate that, if we only consider the recency constraint, we can filter out about 90% of F patterns. If we consider both constraints, however, we can filter out about 99% of F patterns.

5.3. Comparison between RFM-patterns and R^AFM patterns

In this section, we compare RFM and R^AFM patterns using real dataset SC-POS. We first compare the runtime and the number of patterns of the RFM and R^AFM algorithms under selected values of support thresholds, and then compare the numbers of RFM patterns and R^AFM patterns under two recency settings. We set *min-sup_max* = 1, *M_min* = 50, and *M_max* = 10,000 in these comparisons.

Figs. 9 and 10 give the runtimes and numbers of patterns of the RFM and R^AFM algorithms under selected values of support thresholds ranging from 0.18% to 0.22%. The results show that both algorithms require roughly the same amount of runtime, which is expected because computing recency values does not consume much time. Further, Fig. 10 shows that the number of R^AFM patterns is always larger than that of RFM patterns. We use Fig. 11 to explain this result, where the numbers of RFM and R^AFM patterns are obtained for two “recency windows.”

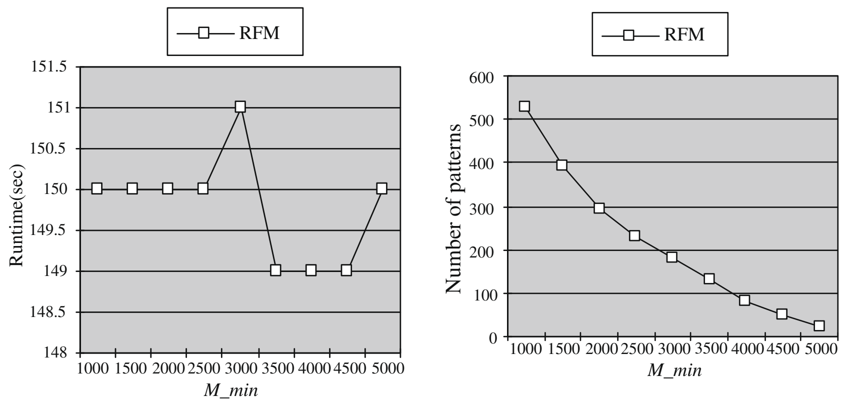

Fig. 8. Runtime and number of patterns vs. *M_min*.

Table 8
The proportions of patterns in datasets.

*C*10-*T*2.5-*S*4-*I*1.25	RF		RFM		F	
Name	# of patterns	%	# of patterns	%	# of patterns	%
D = 25, *minsup_min* = 0.075	186	22.28	48	5.75	835	100
D = 50, *minsup_min* = 0.075	189	23.74	23	2.89	796	100
D = 75, *minsup_min* = 0.075	187	23.58	23	2.9	793	100
C = 10, *minsup_min* = 0.015	4	4.04	0	0	99	100
C = 15, *minsup_min* = 0.015	78	21.67	5	1.39	360	100
C = 20, *minsup_min* = 0.015	293	29.27	36	3.6	1001	100
T = 2.5, *minsup_min* = 0.008	152	21.71	43	6.14	700	100
T = 3.5, *minsup_min* = 0.008	639	28.11	204	8.97	2273	100
T = 4.5, *minsup_min* = 0.001	1122	24.64	455	9.99	4554	100
SC-POS, *minsup_min* = 0.015	168	9.86	14	0.82	1704	100

The first recency window is between *Rtime_min* = 120 and *Rtime_max* = 240, denoted as [120,240], and the second is [240,360]. Note these two windows have the same length, but the second is more "recent." The results indicate that we tend to obtain more RFM patterns when the window is more recent, and more R^AFM patterns when the window is less recent. This is because the average recency value is always smaller than the latest recency value. Therefore, when the window has earlier starting time, we would obtain more R^AFM patterns.

5.4. Applications and managerial implications

In this section, we first show how the proposed algorithm can be used for pattern segmentation, just as RFM has-been used for market segmentation. To illustrate it, we divide the ranges of *R*, *F*, and *M* of the SC-POS dataset into five equal segments (as shown in Table 9) to create 125 groups for the RFM-patterns. Some of the groups are shown in Table 10, where group a-b-c contains patterns with the *F*, *R*, and *M* values in segments *a*, *b*, and *c*, respectively. For example, a pattern in group 1-1-1 has an *F* value between 0.007 and 0.008, an *R* value between 0 and 75, and an *M* value between 0 and 100.

The information in Table 10 is interesting in several ways. For example, the patterns in group 1-1-5 have a high monetary value in the past, but a low frequency. We may check if these patterns still have a high monetary value now, and if not, we should try to find the reason. Another example is the patterns in group 5-5-1, which become very active recently but have a low-value. For these patterns, we may attempt to increase their values by, for example, setting higher prices or lowering their costs for the items in the pattern. If these options are not available or attractive, we may consider removing these items from our selling list.

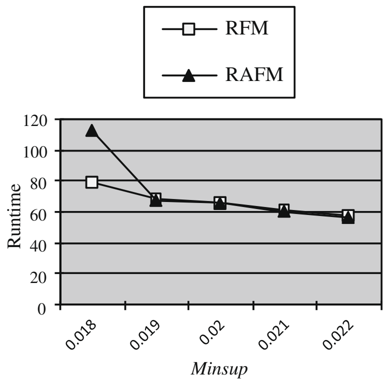

Fig. 9. Runtime vs. Minsup.

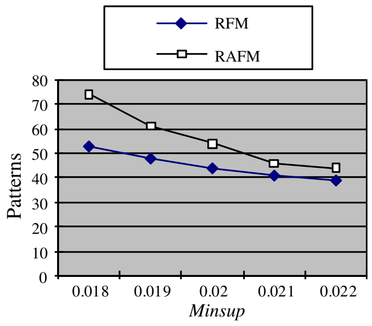

Fig. 10. Pattern number vs. Minsup.

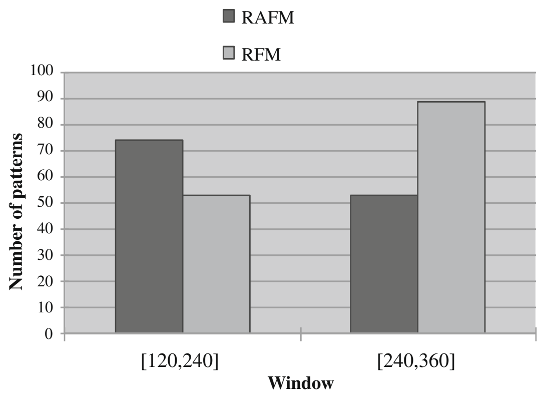

Fig. 11. Numbers of patterns generated by RFM and R^AFM in two time windows.

Table 9
Five segments for each of the selection criteria.

Divisions	R	F	M
Segment 1	0–75	0.007–0.008	0–100
Segment 2	75–150	0.008–0.009	100–200
Segment 3	150–225	0.009–0.01	200–300
Segment 4	225–300	0.01–0.02	300–400
Segment 5	300–360	0.02–1	400

Table 10
RFM-pattern segmentation.

RFM-groups	No. of patterns
1-1-1	50
3-3-3, 1-1-5	0
5-5-5	3
5-1-1	40
1-5-1	97
1-5-5	17
5-1-5, 5-3-5	0
5-5-1	22
3-5-5	4
5-5-3	3

Using the framework, we can perform additional analyses to study possible changes in patterns over time. For example, by fixing the values for *F* and *M* at appropriate levels, and examining the patterns over the *R* segments, we can identify “emerging patterns” and “inactive patterns,” which are only found, respectively, in recent and early segments. Furthermore, we may find cyclic or seasonal patterns by the same analysis. However, the *R* segments have to be set on a monthly, quarterly, or yearly basis.

Furthermore, we may fix the *M* value and examine the patterns in different combinations of *R* and *F* segments to see whether their frequencies are stable, increase, or decrease over time. Similarly, we may study the monetary values over time by fixing the *F* value. These analyses provide a retailer with valuable information for decision-making.

6. Conclusions

Most existing algorithms for mining sequential patterns focus on frequency as the sole selection criterion. Although frequency is an essential criterion in determining the value of a pattern, there could be other important ones. For example, the values of products in product recommendation could be a critical criterion for on-line retailers. Otherwise, we may obtain a large number of patterns with low values.

In this paper, we include two such criteria, recency and monetary, and propose the recency, frequency, and monetary (RFM) pattern for SPM. Although RFM variables have been considered by several researchers in developing data-mining methods, this paper is the first in applying RFM in SPM. The RFM-patterns, as traditional sequential patterns, could be applied in various EC applications, such as cross-selling, product recommendation, personalized marketing. E-catalog design, and product bundle design. (For details, refer to (Han and Kamber 2007).

We modify the traditional Apriori algorithm to produce a novel algorithm, named RFM-Apriori, for generating all RFM-patterns. Extensive experiments using nine synthetic datasets and one real-life dataset are carried out to evaluate the proposed algorithm.

In addition to the contributions of introducing the RFM concept to SPM and developing an efficient algorithm, we propose a framework for generating valuable information on customer purchasing behavior for managerial decision-making. Specifically, we suggest a segmentation method by partitioning the patterns into groups based on the RFM indices such that a retailer can further compare, contrast, and aggregate these groups of patterns to find, for example, possible changes in purchasing patterns over time.

Because of the difficulty of obtaining e-retailing clickstream data, we use a real dataset collected by an off-line retailer. There are differences between on-line and off-line retailing data. For instance, the conversion rate in on-line retailing is normally lower than off-line, making it more difficult to reach sufficient frequency for most items. Furthermore, the scale of visiting frequency varies drastically between on-line and off-line – we can safely use days as the time unit for off-line firms, which may not necessarily be adequate for on-line firms. In addition, the patterns of customer visits/purchases could be very different. In view of these problems, further research is necessary to study how the differences in the data influence the patterns generated.

There are several possible extensions for future research. In this paper, we use crisp (exact) constraints to discover patterns. Fuzzy constraints may be used instead, which would allow the thresholds to be flexible. Furthermore, since it may be difficult for a retailer to properly set threshold values for the selection criteria, developing a systematic mechanism for making such a determination would be an interesting issue for further study. Finally, this work can be extended by considering additional criteria/attributes, such as time and location, in defining sequential patterns, which would involve generalizing the GSP algorithm for finding patterns possessing required attributes and/or satisfying selection criteria.

Appendix A. Proofs of analytical results

Proof of Lemma 1. Suppose $B = \langle I_1 I_2 \rangle$ is in LI_2^{rf}. Then, I_1 and I_2 must satisfy the frequency constraint and I_2 must satisfy the recency constraint. Therefore, I_1 must be in LI_1^{f}, and I_2 must be in LI_1^{rf}. Consequently, all patterns in LI_2^{rf} can be found from LI_1^{f} joining LI_1^{rf}.

Similarly, we generate CI_k, where $k > 2$, by joining LI_{k-1}^{rf} and LI_{k-1}^{rf}. In other words, if $B_1 = \langle I_a, I_2, I_3, \ldots, I_k \rangle$, where $B_1 \in LI_k^{rf}$, and $B_2 = \langle I_b, I_2, I_3, \ldots, I_k \rangle$, where $B_2 \in LI_k^{rf}$, then we will have $\langle I_a, I_b, I_2, I_3,, \ldots, I_k \rangle$ and $\langle I_b, I_a, I_2, I_3, \ldots, I_k \rangle$ in CI_k.

Proof of Lemma 2. Suppose pattern $B = \langle I_1, I_2, \ldots, I_{k-1}, I_k \rangle$ is in LI_k^{rf}. Then, any subsequence of $B = \langle J_1, J_2, \ldots, J_s \rangle$ with $J_s = I_k$ will also be in LI_s^{rf}.

Proof of Lemma 3. According to Lemma 2, if $B = \langle I_1, I_2, \ldots, I_{k-1}, I_k \rangle$ is in LI_k^{rf}, then $B_1 = \langle I_1, I_3, \ldots, I_{k-1}, I_k \rangle$ and $B_2 = \langle I_2, I_3, \ldots, I_{k-1}, I_k \rangle$ must be in LI_{k-1}^{rf}. Since the candidate patterns generated by joining B_1 and B_2 belong to CI_k, CI_k must contain all patterns in LI_k^{rf}.

Proof of Theorem 1. The algorithm first uses the traditional Apriori algorithm to find LI_1^f, and then scans the database to find LI_1^{rf} and LI_1^{rfm} by computing their *rf*-supports and *rfm*-supports. Since the time complexity of this step is dominated by those of the succeeding steps, we skip analyzing its time complexity. In the second step, the algorithm repeats K iterations to find the patterns of all different lengths. In iteration k, $1 \leqslant k \leqslant K$, the algorithm performs the following operations: (1) It generates CI_k; (2) Build the inverse candidate tree from CI_k: and (3) Scan every transaction of the database by procedure *Traverse* to determine the supports of all candidate patterns in CI_k.

To generate CI_k, it needs to join two sets, either LI_1^f join LI_1^{rf} or LI_{k-1}^{rf} join LI_{k-1}^{rf}. It is difficult to estimate the exact size of CI_k because many patterns in the set are redundant and can be removed. However, almost all previous studies about Apriori-like algorithms have the same finding that the largest candidate size usually happens at small k. Therefore, we treat the largest size of $|CI_k|$ as a parameter, and the time needed to generate CI_k is bounded from above by $O(\rho)$.

Next, we store CI_k in a candidate tree. To this end, we first create an empty tree and then insert every candidate pattern in CI_k into the tree. Thus, the tree can be constructed in time $O(\rho \times R)$, because we insert at most ρ candidate patterns into the tree, where each candidate pattern has a length no more than R.

Having constructed the candidate tree, we need to scan the database by procedure *Traverse* to compute the supports of candidate patterns in CI_k. Although procedure *Traverse* seems a little complicated, its function can be summarized as: it generates all subsequences of length k from the data sequence one after another and check if the generated subsequence can match a leaf node, a candidate pattern, in the tree. If so, we then further check if it only satisfies the recency constraint or satisfies both the recency and monetary constraints. By checking these constraints, the procedure can determine the three supports of a pattern. In this step, please note that the number of subsequences with length k in a data sequence is no more than C_R^n. Further, a subsequence takes $O(R)$ steps to traverse from the root to a leaf node. Once reaching the leaf node, we then check the recency and monetary constraints to determine the three supports, which can be done in time $O(R)$. Therefore, procedure *Traverse* takes time $O(C_R^n \times R)$ to process a data sequence. Accordingly, it spends time $O(m \times C_R^n \times R)$ to process all data sequences in phase k. Summing the times for these three parts, we obtain the total time for these K iterations is $O(((m \times C_R^n \times R) + (\rho \times R)) \times K)$. In a normal situation, it is expected that $O(m \times C_R^n) \geqslant O(\rho)$, and, as a result, the time complexity can be expressed as $O((m \times C_R^n \times R) \times K)$.

Appendix B. Description of the datasets used in the experiments

We applied the algorithm proposed by Agrawal and Srikant (1995) to generate the synthetic datasets. Each transaction in the datasets contains a sequence of itemsets. However, the transaction data are extended such that the items in different itemsets are assigned different time values and that those in the same itemsets are assigned the same time values. A value w is generated randomly from a Poisson distribution with mean T_I for each customer, which is used as the average time interval between successive itemsets in the sequence of this particular customer. Then, the intervals between successive itemsets of this customer are randomly generated by a Poisson distribution with mean w. All items are divided into three price levels: high, medium, and low. The type of an item is randomly determined according to the proportions specified beforehand. Then, an item's price is randomly generated by a Poisson distribution with mean *H_price*, *M_price*, or *L_price*, depending on its type. We also assume that the quantity of an item purchased in a transaction is inversely related to its price. According to this assumption, the purchased quantities of high-, medium-, and low-priced items are generated by a Poisson distribution with mean *H_quantity*, *M_quantity*, and *L_quantity*, respectively. In this experiment, we set 50% of items at a low price level, 30% at a medium price level, and the last 20% at a high price level.

A real dataset, called SC-POS, is also used in the experiment. The data was supplied by the *Songchine* supermarket chain in Taiwan, consisting of transactional records from its twenty branches between 2001/12/27 and 2002/12/31. Each transactional record contains the date, time, item names, prices, and purchased quantities associated with the transaction. A series of data pre-processing and cleaning tasks were performed, including combining the sequence data from the 20 branches, removing records without information on a membership card, and merging the transactions of the same customer into one data-sequence. After the data preparation, we sort all transactions of the same customer according to the transaction times. The final SC-POS dataset contains 17,685 product items and 33,500 customers' data-sequences.

References

Agrawal, R., and Srikant, R. Mining sequential patterns. In *Proceedings of the 1995 International Conference Data Engineering*, 1995, 3–14.

Bult, J. R., and Wansbeek, T. J. Optimal selection for direct mail. *Marketing Science*, 14, 4, 1995, 378–394.

Chen, M. S., Park, J. S., and Yu, P. S. Efficient data mining for path traversal patterns. *IEEE Transactions on Knowledge and Data Engineering*, 10, 2, 1998, 209–221.

Chen, Y. L., Tang, K., Shen, R. J., and Hu, Y. H. Market basket analysis in a multiple-store environment. *Decision Support Systems*, 40, 2, 2005, 339–354.

Chen, Y. L., Chen, S. S., and Hsu, P. Y. Mining hybrid sequential patterns and sequential rules. *Information Systems*, 27, 5, 2002, 345–362.

Chen, Y. L., Chiang, M. C., and Kao, M. T. Discovering time-interval sequential patterns in sequence databases. *Expert Systems with Applications*, 25, 3, 2003, 343–354.

Chen, Y. L., and Huang, C. K. Discovering fuzzy time-interval sequential patterns in sequence databases. *IEEE Transactions on Systems, Man and Cybernetics – Part B*, 35, 5, 2005, 959–972.

Cheng, C. H., and Chen, Y. S. Classifying the segmentation of customer value via RFM model and RS theory. *Expert Systems with Applications*, 36, 3, 2009, 4176–4184.

Cui, G., Wong, M. L., and Lui, H. K. Machine learning for direct marketing response models: Bayesian networks with evolutionary programming. *Management Science*, 52, 4, 2006, 597–612.

Etzion, O., Fisher, A., and Wasserkrug, S. e-CLV: A modeling approach for customer lifetime evaluation in e-Commerce domains, with an application and case study for online auction. *Information Systems Frontiers*, 7, 4–5, 2005, 421–434.

Han, J., Dong, G., and Yin, Y. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the 1999 International Conference on Data Engineering*, 1999, 106–115.

Han, J., and Kamber, M. *Data Mining: Concepts and Techniques*, 2nd edition. Morgan Kaufman, San Francisco, CA, 2007.

Hu, H. L., and Chen, Y. L. Mining typical patterns from databases. *Information Sciences*, 178, 19, 2008, 3683–3696.

Lawrence, R. D., Almasi, G. S., Kotlyar, V., Viveros, M. S., and Duri, S. S. Personalization of supermarket product recommendations. *Data Mining and Knowledge Discovery*, 5, 1–2, 2001, 11–32.

Lee, J., Podlaseck, M., Schonberg, E., and Hoch, R. Visualization and analysis of clickstream data of online stores for understanding Web merchandising. *Data Mining and Knowledge Discovery*, 5, 1–2, 2001, 59–84.

Lin, Q. Y., Chen, Y. L., Chen, J. S., and Chen, Y. C. Mining inter-organizational retailing knowledge for an alliance formed by competitive firms. *Information and Management*, 40, 5, 2003, 431–442.

Lin, C., and Hong, C. Using customer knowledge in designing electronic catalog. *Expert Systems with Applications*, 34, 1, 2008, 119–127.

Mahdavi, I., Cho, N., Shirazi, B., and Sahebjamnia, N. Designing evolving user profile in e-CRM with dynamic clustering of Web documents. *Data and Knowledge Engineering*, 65, 2, 2008, 355–372.

Mannila, H., Toivonen, H., and Inkeri Verkamo, A. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1, 3, 1997, 259–289.

Mozer, M. C., Wolniewicz, R., Grimes, D. B., Johnson, E., and Kaushansky, H. Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry. *IEEE Transactions on Neural Networks*, 11, 3, 2000, 690–696.

Pei, J., Han, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M. C. FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the 2000*

International Conference on Knowledge Discovery and Data Mining, 2000, 355–359.

Pei, J., Han, J., Pinto, H., Chen, Q., Dayal, U., Hsu, and M. C. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 2001 International Conference on Data Engineering*, 2001, 215–224.

Schafer, J. B., Konstan, J. A., and Riedl, J. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5, 1–2, 2001, 85–114.

Srikant, R., and R. Agrawal, Mining sequential patterns: generalizations and performance improvements. In *Proceedings of the Fivth International Conference on Extending Database Technology*, 1996, 3–17.

Spiliopoulou, M., and Pohle, C. Data mining for measuring and improving the success of Web sites. *Data Mining and Knowledge Discovery*, 5, 1–2, 2001, 85–114.

Tang, K., Chen, Y. L., and Hu, H. W. Context-based market basket analysis in a multiple-store environment. *Decision Support Systems*, 45, 1, 2008, 150–163.

Toroslu, I. H. Repetition support and mining cyclic patterns. *Expert Systems with Applications*, 25, 3, 2003, 303–311.

Turban, E., Lee, J., King, D., and Chung, H. M. *Electronic Commerce: A Managerial Perspective*, 4th edition. Prentice Hall, Upper Saddle River, NJ, 2006.

Weiss, G. M. Data mining in the telecommunications. In O. Maimon and L. Rokach (eds.), *Data mining and knowledge discovery handbook: a complete guide for practitioners and researchers*, Kluwer Academic Publishers, 2005, 1189–1201.

Yu, C. C., and Chen, Y. L. Mining sequential patterns from multi-dimensional sequence data. *IEEE Transactions on Knowledge and Data Engineering*, 17, 1, 2005, 136–140.

Zaki, M. J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42, 1–2, 2001, 31–60.