

Discovering Itemset Interactions*

Ping Liang¹, John F. Roddick¹, Aaron Ceglar^{1,2},
Anna Shillabeer^{1,3} and Denise de Vries¹

¹ School of Computer Science, Engineering and Mathematics
Flinders University
PO Box 2100, Adelaide, South Australia 5001
Email: {ping.liang, roddick, ceglar, denise.devries}@csem.flinders.edu.au

² Defence Science and Technology Organisation
PO Box 1500, Edinburgh, South Australia 5111

³ Heinz School Australia
Carnegie Mellon University
Torrens Building, 220 Victoria Square
Adelaide, South Australia 5000
Email: anashill@andrew.cmu.edu

Abstract

Itemsets, which are treated as intermediate results in association mining, have attracted significant research due to the inherent complexity of their generation. However, there is currently little literature focusing upon the interactions between itemsets, the nature of which may potentially contain valuable information. This paper presents a novel tree-based approach to discovering itemset interactions, a task which cannot be undertaken by current association mining techniques.

Keywords: Itemset interaction, relative support, FP-tree.

1 Introduction

Since the seminal work of Agrawal, Srikant *et al.* (Agrawal *et al.* 1993, Agrawal & Srikant 1994) association mining has become a mature field of research applied in a variety of domains including commerce, defence, health, manufacturing and engineering. Within the classic support-confidence framework, association mining first finds all itemsets satisfying a predefined minimum frequency (support), and then extracts strong association rules from them. In this process, itemsets are treated as intermediaries, the generation of which has attracted significant research due to its inherent complexity (Ceglar & Roddick 2006). However, the nature of the inferred relationship between a rule's participant itemsets is rarely considered during association mining due to two factors:

- The need to constrain itemset generation through the specification of a pruning heuristic, such as support. As support is reduced, the number of itemsets generated typically increases significantly.

*This research was funded through an ARC Linkage Grant LP0667714 in collaboration with ASERNIP/Royal Australasian College of Surgeons and the Australian Computer Society Inc, SA Branch. We would like to acknowledge their generous support.

- The setting of a minimum support threshold eliminates infrequent itemsets, making it difficult to discover those relationships involving an itemset of low frequency.

This paper presents Finding Itemset Interactions (FITIN), a method for finding itemset interactions, a novel technique in the field of association mining. FITIN adopts a tree-based approach to represent itemset interaction patterns, such as Competitors, Catalysts or Clusters (see Section 3) and searches for matched instances within an FP-tree (see Section 4). Experimental results on both synthetic and real datasets demonstrate the effectiveness of this approach (see Section 6).

The paper is organized as follows. In the next section, a review of related work is provided. Section 3 contains definitions of the terminology used and Section 4 provides a theoretical discussion of the FITIN approach. Section 5 discusses algorithm development and Section 6 presents a proof-of-concept implementation and experimental evaluation. Section 7 concludes the paper with a discussion of future work.

2 Related Work

Association mining was initially proposed by Agrawal *et al.* (1993) and aims to find interesting relationships between items (Han & Kamber 2000). The majority of research to date has focused upon the efficient discovery of itemsets as its level of complexity is significantly greater than that of inference generation (Agrawal & Srikant 1994, Bayardo 1998, Brin, Motwani, Ullman & Tsur 1997, Gunopulos *et al.* 1997, Han *et al.* 1997, 2000, Liu *et al.* 2002, Park *et al.* 1995, Toivonen 1996, Zaki & Hsiao 2002).

Association rules do not capture interesting dependencies between items (Aggarwal & Yu 1998, Brin, Motwani & Silverstein 1997). For example, customers who buy coffee might not often buy tea in the same transaction. The behaviour of buying coffee and buying tea is not associated but (negatively) correlated. Correlation rules, which aim to discover the correlations in the association rules, were proposed by Brin, Motwani & Silverstein (1997) and have attracted substantial attention (Lee *et al.* 2003, Liu *et al.* 1999, Piatetsky-Shapiro 1991). Our work differs from correlation mining in two ways. First, correlation relates to the dependency between items while interaction is a class of relationship focused on the effects, with an unknown cause, between different itemsets. Second,

Transaction ID	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
Items	a,b,c,d,e,f	e,f	a,b,c,d,g	a,b,m,n	c,d,e,f	a,b,c,d	a,b,c,d	e,f,g	c,e,f	a,b,c,d,g

Table 1: Transaction dataset D

Interaction ID	Itemsets		Trans e_A	Trans e_B	Trans e_A	Pattern
	e_A	e_B	without e_B	without e_A	with e_B	Revealed
1	{a,b}	{c,d}	T_4	T_5	$T_1T_3T_6T_7T_{10}$	Facilitator
2	{a,b}	{e,f}	$T_3T_4T_6T_7T_{10}$	$T_2T_5T_8T_9$	T_1	Competitor

Table 2: Sample two itemset interaction patterns

the participants in an interaction are itemsets rather than items.

Recently, the absence of itemsets has attracted a lot of research. For instance, a store manager may like to be informed that customers who do not buy product X are likely to buy product Y . The problem of identifying negative associations, in the form $A \Rightarrow \neg B$, $\neg A \Rightarrow B$ and $\neg A \Rightarrow \neg B$, has been widely explored (Antonie & Zaiane 2004, Daly & Taniar 2004, Savasere et al. 1998, Wei & Chen 2000, Xu et al. 2004, Yan et al. 2004, Yuan et al. 2002). However, the relationships between itemsets are not considered. In our work, we do not generate negative itemsets, but rather search for the effect one itemset has on another.

Teng (2002) denotes the dissociation rule as $AB \not\Rightarrow C$, which indicates the presence of A and B is a good predictor of the absence of C . This approach is able to discover competitors in an itemset which in some ways is similar to our work. However, itemset interactions can be generalised to reveal various patterns rather than the simply Competitor patterns, such as Catalysts and Clusters, which cannot be handled by approaches such as Teng’s.

3 Itemset Interaction

The nature of the relationship between interacting itemsets is potentially useful. Given the sales transaction data D shown in Table 1 (which is necessarily a small sample for illustration), Table 2 presents two itemset interactions of potential interest. The first interaction of interest occurs between the itemsets {a,b} and {c,d}, each occurring without the other only once in T_4 and T_5 respectively. However, they occur together in 5 transactions ($T_1, T_3, T_6, T_7, T_{10}$), revealing that they might *facilitate* each other, or that, the sales of {a,b} or {c,d} may be increased because of the presence of the other. The second interaction, between {a,b} and {e,f}, shows individual occurrence of {a,b} in 5 transactions ($T_3, T_4, T_6, T_7, T_{10}$) and individual occurrence of {e,f} in 4 transactions (T_2, T_5, T_8, T_9) and co-existence in only a single transaction (T_1), revealing that they *compete* with each other. In other words, the sales of {a,b} or {e,f} are decreased in the presence of the other.

Table 3 represents a descriptive list of some potential types of interaction relationships in market basket data (qv. Roddick et al. (2008)). However, such patterns may be exhibited within many domains. For example, different medicines can interact resulting in distinct treatment outcomes, while in human resources, the productivity of a team can be affected by the interaction between its members.

In addition, the itemset interaction patterns shown in Table 3 may combine to form more complex patterns, i.e., one Cluster may be a Catalyst to another Cluster and one itemset may compete with more than two itemsets. The conjunction of itemset interaction patterns is beyond the scope of this paper.

3.1 Definition of Itemset Interaction

Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of *items*, and D be a dataset containing a set of transactions, where each transaction t_i is a set of items such that $t_i \subseteq I$ (Agrawal & Srikant 1994). Each transaction set may have subsets which are called itemsets. For example, given $t_i = \{i_1, i_2, i_3\}$, the resultant itemsets are $\{i_1\}$, $\{i_2\}$, $\{i_3\}$, $\{i_1, i_2\}$, $\{i_1, i_3\}$, $\{i_2, i_3\}$ and $\{i_1, i_2, i_3\}$ excluding ϕ . An important property of an itemset, x , is support σ , which is the ratio of the number of transactions containing x to the number of transactions in D , defined as $\sigma(x) = \frac{P(x)}{|D|}$, where $P(x)$ denotes the number of transactions containing x .

An itemset interaction (Π) is a relationship between multiple itemsets, which results in a change in behaviour in at least one of the participating itemsets, and is denoted as $\Pi = \{E, \mathfrak{u}\}$, where E is the set of all participating itemsets and \mathfrak{u} is a union set, or u-set, i.e., $\mathfrak{u} = \bigcup_{e_i \in E} e_i$. For each $e_i \in E$, e_i is called

an e-set. Given E , we define $\Pi = \{E, \mathfrak{u}\}$ and the following conditions hold:

- Multiple e-sets: $|E| > 1$. An Π contains at least two distinct e-sets since an e-set cannot interact with itself.
- Atomic: For every $e_i, e_j \in E$, neither $e_i \subset e_j$ nor $e_j \subset e_i$ since a set cannot interact with its own subset or superset. For example, given $e_i = \{a, b\}$, $e_j = \{a, b, c, d\}$, we have $e_i \subset e_j$ and hence no interaction can occur between e_i and e_j .
- Existing Interaction. The u-set (\mathfrak{u}) exists in at least one transaction. Otherwise, no interaction is possible. For example, given $E = \{e_i, e_j\}$, $e_i = \{a, b\}$, $e_j = \{c, d\}$, if there is no transaction containing $\mathfrak{u} = e_i \cup e_j = \{a, b, c, d\}$, there is no possible interaction between e_i and e_j .

Given an $\Pi = \{E, \mathfrak{u}\}$, we define a relative support¹ σ_R for an e-set $e_i \in E$ as follows:

$$\sigma_R(e_i) = \frac{P(e_i) - Q(e_i)}{|D|} \quad (1)$$

where $Q(e_i)$ denotes the number of transactions containing other e-sets in E in all transactions containing e_i .

Relative support represents the occurrence of an e-set independent of other e-sets in D . For example, using the data from Table 1, one itemset interaction from Table 2 is $\Pi = \{E, \mathfrak{u}\}$, where $E = \{\{a, b\}, \{c, d\}\}$, $\mathfrak{u} = \{a, b\} \cup \{c, d\} = \{a, b, c, d\}$. Since e-set $\{a, b\}$ exists in 6 transactions ($T_1, T_3, T_4, T_6, T_7, T_{10}$), we have $P(\{a, b\}) = 6$. Among those transactions, e-set $\{c, d\}$ exists in $T_1, T_3, T_6, T_7, T_{10}$ and we have $Q(\{a, b\}) = 5$. Thus the relative support of $\{a, b\}$ is:

¹This concept builds on the work by Shillabeer & Pfitzner (2007).

Pattern	Description	Example
Competitor	An itemset competes with another itemset.	Itemset {Chips, Cola} competes with itemset {Chips, Lemonade} Desc: Customers tend to buy Chips and Cola or Chips and Lemonade individually, but they seldom buy Chips, Cola and Lemonade together.
Catalyst	An itemset facilitates another itemset.	Itemset {milk, butter} facilitates itemset {bread, jam}. Desc: There are fewer customers who buy milk and butter or bread and jam individually. More customers buy them together.
Cluster	Three or more itemsets frequently occur together but seldom occur individually.	Itemsets {milk},{bread} and {eggs} form a cluster. Desc: These three itemsets are seldom purchased by customer individually but always together.

Table 3: Sample types of itemset interaction patterns in market basket data

Π	e-sets				u-set		e_a		e_b		u-set status	Behavioural change of e_a	Behavioural change of e_b
	e_a	$\sigma(e_a)$	e_b	$\sigma(e_b)$	u	σ	σ_R	status	σ_R	status			
Π_1	{a,b}	0.6	{c,d}	0.6	{a,b,c,d}	0.5	0.1	Low	0.1	Low	High	Low \rightarrow High	Low \rightarrow High
Π_2	{a,b}	0.6	{c,f}	0.5	{a,b,e,f}	0.1	0.5	High	0.4	High	Low	High \rightarrow Low	High \rightarrow Low

Table 4: Sample itemsets behavioural changes ($min_{HF} = 0.4$, $min_{LF} = 0.1$)

$$\sigma_R(\{a,b\}) = \frac{P(\{a,b\}) - Q(\{a,b\})}{|D|} = \frac{(6-5)}{10} = 0.1.$$

This means that $\{a,b\}$ occurs independently of $\{c,d\}$ in 10% of transactions.

Given two user specified thresholds for support min_{HF} (Minimum High Frequency) and max_{LF} (Maximum Low Frequency), where $0 < max_{LF} \leq min_{HF}$, let an itemset x represent a u-set or an e-set. x has a *High* status when $\sigma(x) \geq min_{HF}$ or $\sigma_R(x) \geq min_{HF}$, and a *Low* status when $\sigma(x) \leq max_{LF}$ or $\sigma_R(x) \leq max_{LF}$. x has a *Normal* status when $max_{LF} < \sigma(x) < min_{HF}$ or $max_{LF} < \sigma_R(x) < min_{HF}$. For brevity, the status of *High*, *Low* and *Normal* is denoted as H, L and N respectively, e.g., $(\{a,b\}, H)$ indicates that itemset $\{a,b\}$ has a *High* status.

The behavioural changes of an e-set can be described by using this concept of itemset status. For example, using the data from Table 1 and given that $min_{HF} = 0.4$ and $max_{LF} = 0.1$, the behavioural changes of the itemsets in the two itemset interactions from Table 2 are shown in Table 4. Taking Π_1 as an example, itemsets $\{ab\}$ and $\{cd\}$ both have *Low* status when they occur individually since $\sigma_R(\{a,b\}) = 0.1 \leq max_{LF}$ and $\sigma_R(\{c,d\}) = 0.1 \leq max_{LF}$. When they interact, the u-set $\{a,b,c,d\}$ has a *High* status since $\sigma(\{a,b,c,d\}) = 0.5 \geq min_{HF}$, thus revealing that both itemsets change their status from *Low* to *High* through the interaction.

Given min_{HF} and max_{LF} , a valid itemset interaction pattern (P_Π) relates to an itemset interaction (Π) where the status of all e-sets and u-set are known. Given a database D and a P_Π , we will use Finding Itemset Interactions (FITIN) to find all matches of P_Π in D .

4 The FITIN Approach

Our initial approach to finding itemset interactions was naive, generating all itemsets and then merging them together to find all matches of a given P_Π . This was a good solution while the number of itemsets was small. However, it became inefficient as the number of itemsets increased. Given n frequent items in a database, the possible itemsets generated are $2^n - 1$ and the number of combinations of itemset interactions is significantly larger, $|I| \gg 2^n - 1$. It therefore

becomes intractable to generate all itemset interactions when n is large.

4.1 Overview

To overcome the scalability issue, FITIN adopts an alternative tree-based approach which extends the concepts discussed in the presentation of FP-trees (Han et al. 2000). As an FP-tree stores the pertinent items in a structured way that facilitates the discovery of interactions between itemsets, it becomes more efficient to find matches of a given P_Π within the FP-tree than to search for them in individual itemsets. Firstly, there is no need to generate all itemsets as a preliminary step, which significantly reduces the computational complexity and temporal overhead. Secondly, once an FP-tree has been built, there is no pruning process to remove infrequent itemsets. Information held in the FP-tree is complete.

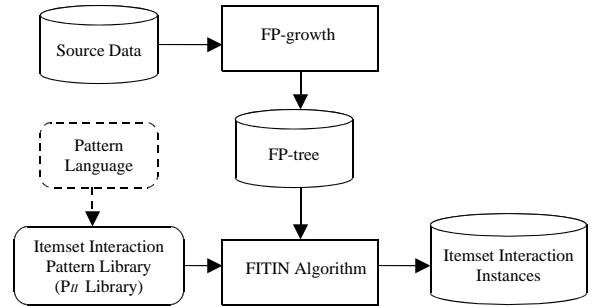


Figure 1: FITIN framework

There are three key parts to the FITIN framework: an FP-tree, an P_Π library with an associated pattern language and a pattern search algorithm, as presented in Figure 1. The FP-tree is generated to concisely represent the pertinent dataset information. The P_Π library contains a set of IIP-trees (as discussed in Section 4.3), each of which represents a P_Π . The integration of a pattern library and its associated pattern language allows users to retrieve, modify and create new patterns in FITIN based on their requirements and their datasets. At the core of FITIN, there is a search algorithm to find all matches of a given P_Π held in an FP-tree.

4.2 FP-trees

FP-trees, first proposed by Han et al. (2000), are a compact data structure that contains the complete set of information held in a database relevant to frequent pattern mining. The FP-tree stores a single item at each node, and includes a support count and additional links to facilitate processing. These links start from a header table and link together all nodes in the FP-tree which store the same item. Readers may refer to (Han et al. 2000) for more details.

4.3 The IIP-tree

FITIN represents itemset interaction patterns using a novel IIP-tree (Itemset Interaction Pattern tree), which makes one assumption.

Assumption: All items of a P_{Π} must themselves be frequent, although the participating itemsets may not be.

Rationale: The FP-tree target contains only frequent items (Han et al. 2000). Therefore, if a P_{Π} instance contains infrequent items, it is impossible to find matches in FP-trees. Thus infrequent items have no significance in the search of the FP-tree and would be ignored.

An IIP-tree has a set of prefix subtrees as the children of the root. Each node consists of three fields: the item, a node link and status, where the node link points to the next node containing the same item in the IIP-tree.

Algorithm 4.1 IIP-tree Construction

```

1: Input: An itemset interaction pattern  $P_{\Pi}$ 
2: Output: An IIP-tree
3:  $T = \text{Root}(P_{\Pi})$ 
4: sort all items in  $P_{\Pi}.u$  in support descending order
5: insert-tree( $P_{\Pi}.u, P_{\Pi}.T, 0$ )
6: for each e-set  $e_i$  do
7:   sort all items in  $e_i$  according to their order in  $P_{\Pi}.u$ 
8:   insert-tree( $e_i, P_{\Pi}.T, 0$ )
9: end for
10: insert-tree(itemset  $e$ , treeNode  $node$ , int index)
11: if index <  $e.length$  then
12:   if  $node.hasChild()$  then
13:     if  $node.child.item-name \neq e[index].item-name$  then
14:        $node.addChild(e[index])$ ,
          $linkset.add(e[index])$ 
15:     else
16:        $node.child.addStatus(e.status)$ 
17:     end if
18:   else
19:      $node.addChild(e[index])$ ,
        $linkset.add(e[index])$ 
20:   end if
21:   insert-tree( $e, node.child, index++$ )
22: end if

```

The algorithm to construct an IIP-tree is outlined in Algorithm 4.1, which firstly sorts all items in the u-set in support descending order. The u-set is then inserted into the tree. During the remaining insertion process, items in each e-set are inserted according to their order in the u-set. If a node on the insertion path already exists, the item stored in the node appears in multiple itemsets and will have multiple status. Multiple status is denoted as $mHnLpN$, where $n > 0, m > 0, p > 0$. For example, notation ‘2L2H’ represents two *Low* status and two *High* status. If

$n = 1$ or $m = 1$ or $p = 1$, the number ‘1’ is omitted in the notation for brevity, e.g., ‘2LH’ represents two *Low* status and one *High* status.

After construction, a language is required to describe the IIP-tree. An IIP-tree is defined as a collection of tuples $\langle node, parent, [Nchildren], [Nstatus] \rangle$, where $Nchildren$ is the collection of the node’s children from left to right and $Nstatus$ is the collection of status of its children from left to right. In addition, a *label*, which is an instantiated description of the pattern, can be imposed over the definition.

Given an P_{Π} containing two e-sets: $(\{a,b\}, H)$, $(\{a,c\}, H)$ and the u-set $(\{a,b,c\}, L)$, the IIP-tree construction process and its description are shown in Fig. 2. Fig. 2(a) creates the root and inserts the u-set $\{a,b,c\}$ and (b) shows the effect of inserting $(\{a,b\}, H)$. Fig. 2(c) illustrates the effect of inserting $(\{a,c\}, H)$ with a link created between the two ‘c’ nodes. Fig. 2(d) shows the tuples which are constructed for all non-leaf IIP-tree nodes. For brevity, leaf nodes are not listed as they do not have children.

5 Itemset Interaction Search

Given the structure of the two trees, IIP-tree and FP-tree, the FITIN algorithm searches the FP-tree for all matches of the IIP-tree (shown in Algorithm 5.1).

Algorithm 5.1 FITIN Algorithm

```

1: Input: FP-tree  $fp$ , IIP-tree  $ip$ 
2: Output: Matched instances of  $ip$ 
3: mineITIN( $ip.root.next$ )
4: mineITIN(IIPtreeNode  $node$ )
5: if  $node \neq null$  then
6:    $P = \text{arrayOfPath}(node)$ 
7:    $S = \text{set of status of } P$ 
8:   for each  $item$  in  $fp.header$  do
9:      $C = \text{getCountList}(item, P)$ 
10:    for each  $C[i]$  do
11:      if  $!C[i].fitStatus(S[i])$  then
12:        return
13:      end if
14:    end for
15:    runMiner( $node.child$ )
16:  end for
17: end if
18: getCountList(FPtreeNode  $node$ , Array  $P$ )
19: countList = new int[P.size]
20: while  $node \neq null$  do
21:   if  $node.prefix.hasPath(P[i])$  then
22:     countList[i] = countList[i] +  $node.count$ 
23:   end if
24:    $node = node.link$ 
25: end while
26: return countList
27: arrayOfPath(IIPtreeNode  $node$ )
28: path = new Array()
29: while  $node \neq null$  do
30:   path.add( $node.prefix$ )
31:    $node = node.link$ 
32: end while
33: return path

```

FITIN provides a mechanism for tree searching, which substitutes IIP-tree nodes with the items from the FP-tree header table. The complexity of the process is $O(\frac{n!}{(n-k)!k!})$, where n is the number of frequent items in an FP-tree and k is the number of distinct items in an IIP-tree.

As the IIP-tree’s leftmost branch contains all items in an P_{Π} (u-set), FITIN requires a single pass over this branch for search purposes. For each node in this u-set branch, all other nodes representing the same item

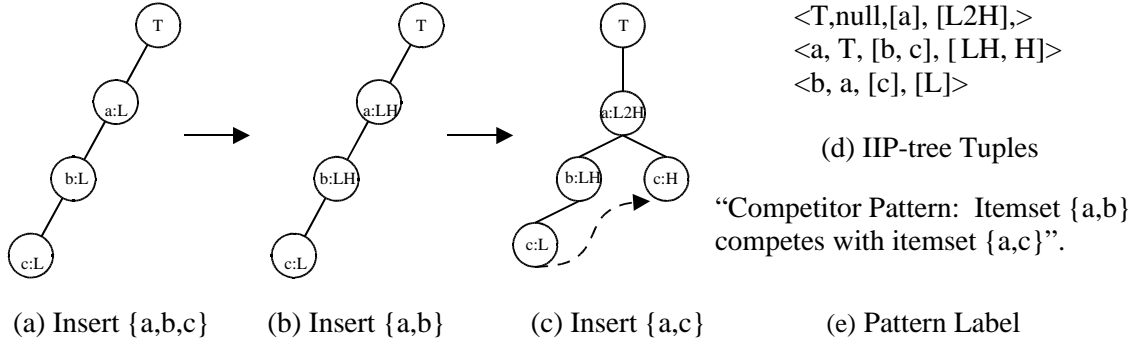


Figure 2: Sample IIP-tree construction and description

are accessible through the node links. The collection of itemsets that terminate with a specific item is efficiently obtained through this structure. During the process to calculate support, FITIN visits the relevant FP-tree branches by following the link of the specific item in the header table. If a visited branch contains the u-set, the support of the u-set increases by adding the support count of the visited node in the branch. If there is no u-set in the branch and it contains a single itemset (e-set), the relative support of the itemset (e-set) increases by adding the support count of the visited node too. If there is no u-set in the branch and the branch contains more than one e-sets, those e-sets are not independent of each other in that branch and the support count of the visited node will not be counted according to the definition of relative support.

Once a support is calculated from the FP-tree branches, it will be checked against the stored status in the associated IIP-tree node. Given min_{HF} and max_{LF} , let $\sigma(cnode)$ represent calculated support of an IIP-tree node, $cnode$, and $mHnL$ be its stored status, where $m \geq 0, n \geq 0$. A rule to validate $\sigma(cnode)$ is as follows. If $m \neq 0$, which means $cnode$ has at least one *high* status, it is valid when $\sigma(cnode) \geq m \times min_{HF}$. If $m = 0$ and $n \neq 0$, which means $cnode$ has no *high* status but has at least one *low* status, it is valid when $\sigma(cnode) \leq n \times max_{LF}$.

If $cnode$ has a parent node, $pnode$, with status $pHqL$, where $p \geq m, q \geq n$. The difference in the status of the two nodes is defined as $St_{diff} = pHqL - mHnL = (p - m)H(q - n)L$, while the difference in their support is defined as $\sigma_{diff} = \sigma(pnode) - \sigma(cnode)$. If $p - m \neq 0$ or $q - n \neq 0$, σ_{diff} will be checked against St_{diff} based on the same rule as above.

To illustrate, consider the example shown in Fig. 3, which contains an IIP-tree and an FP-tree. Given that $min_{HF} = 0.4$ and $max_{LF} = 0.2$, the algorithm starts with node 'x' in the IIP-tree. Each node in the IIP-tree will be replaced by its corresponding value from the FP-tree header table.

Starting with item 'a', only one branch ends with 'a', and a support of 1.0 is obtained from the header table, $\sigma(\{a\}) = 1.0$, which is checked against the stored status $2HL$. Since $2 \times min_{HF} = 0.8 < \sigma(\{a\})$, $\sigma(\{a\})$ is valid and the process continues, moving to the next node in the traversal, 'y'. The node 'y' is replaced by all items in the header table, except 'a'.

Starting with item 'b', a branch is generated from the IIP-tree that ends with 'b' and also contains 'a', which is $\{a,b\}$. Following the FP-tree link from 'b', all branches that contain $\{a,b\}$ are found and $\sigma(\{a,b\}) = 0.6$, which is checked against the stored status HL . Since $\sigma(\{a,b\}) > 1 \times min_{HF}$, it is valid. The algorithm continues to check its parent node, which is node 'a' with status $2HL$. The differences in the status

and support of the two nodes are calculated respectively, i.e., $St_{diff} = 2HL - HL = (2 - 1)H(1 - 1)L = H$, $\sigma_{diff} = \sigma(a) - \sigma(b) = 1.0 - 0.6 = 0.4$. The value of σ_{diff} is then checked against St_{diff} . Since $\sigma_{diff} \geq 1 \times min_{HF}$, it is valid. The process advances to 'z', which is replaced with each item in the FP-tree header table except 'a' and 'b'.

Starting with 'c', generate a list of branches in the IIP-tree that contains 'c' and have 'a' or 'b' in their parental path, specifically $\{a,b,c\}$ and $\{a,c\}$. The support of $\{a,b,c\}$ and relative support of $\{a,c\}$ is calculated, $\sigma(\{a,b,c\}) = 0.2$ and $\sigma_R(\{a,c\}) = 0.4$. The algorithm firstly checks the IIP-tree node 'c' in the u-set branch, which has a L status. Since $\sigma(\{a,b,c\}) = 0.2 < max_{LF}$, it is valid. The differences in the status and support of the node and its parent node (node 'b' with status HL) are calculated, i.e., $St_{diff} = HL - L = (1 - 0)H(1 - 1)L = H$, $\sigma_{diff} = \sigma(b) - \sigma(c) = 0.6 - 0.2 = 0.4$. Since $\sigma_{diff} \geq 1 \times min_{HF}$, it is valid. Similarly, the difference in the status and support of the other 'c' node with its parent node (node 'a' with status $2HL$) are generated and checked. Since $St_{diff} = 2HL - H = (2 - 1)H(1 - 0)L = HL$, $\sigma_{diff} = \sigma(a) - \sigma(c) = 1.0 - 0.4 = 0.6$, $\sigma_{diff} \geq 1 \times min_{HF}$, it is also valid. As they are all valid, a matched instance of the IIP-tree has been found.

The algorithm then progresses to 'd'. Branches containing 'd' that have 'a' or 'b' in their parental path are $\{a,b,d\}$ and $\{a,d\}$ which have $\sigma(\{a,b,d\}) = 0.2$ and $\sigma_R(\{a,d\}) = 0.1$. Since $\sigma_R(\{a,d\}) = 0.1 \leq min_{HF}$, it is invalid and thus is pruned. The process continues until all nodes in the leftmost branch of the pattern tree have been substituted.

6 Evaluation of Proof-of-Concept System

To demonstrate the concept, a prototype of FITIN was implemented in Java and several experiments were conducted on both synthetic and real datasets. All tests are done on a 2.6 GHz PC with 1.00 GB of main memory running Windows XP (2002). This implementation is shown to be tractable and able to reveal itemset interactions of potential interest that would otherwise not be reported.

6.1 Synthetic Data

A synthetic data generator was built based on the work reported by Agrawal & Srikant (1994) to produce large quantities of transactional data. Table 5 shows the parameters for data generation, along with their default values and the range of values on which experiments were conducted. Table 6 presents the details of the generated synthetic data.

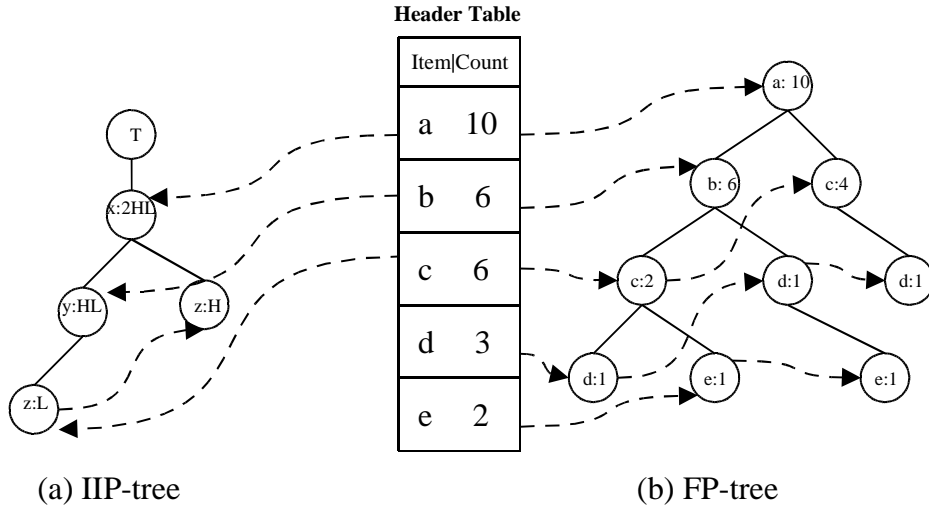


Figure 3: Illustration of FITIN algorithm

Name	Description	Default Value	Range of Values
$ I $	Number of Items	10	10-100
$ T $	Number of Transactions	5K	5K-200K
$ P $	Number of Patterns	50	50-500
TS	Average Size of Transaction	5	5-10
PS	Average Size of Pattern	5	5-10

Table 5: Synthetic data parameters

Data	$ I $	$ T $	$ P $	TS	PS
Syn1	100	10,000	20	5	5
Syn2	200	50,000	150	10	10
Syn2	300	100,000	250	10	15

Table 6: Synthetic data

6.2 Real Data

Three datasets were used to test FITIN, details of which are given in Table 7. The Retail Data is supplied by an anonymous Belgian retail supermarket store (Brijs et al. 1999). The data are collected over three non-consecutive periods between 1999 and 2000. The two datasets BMS-WebView-1 and BMS-WebView-2 are taken from KDDCUP 2000 (Kohavi et al. 2000). They contain several months' worth of click stream data from two e-commerce web sites.

6.3 Tested Pattern

Fig. 4 shows the patterns tested. Fig. 4(a) and Fig. 4(d) represent two Competitor patterns involving two pairs of e-sets: $\{x,y\}$ and $\{x,z\}$ in (a) and $\{x,y\}$ and $\{m,n\}$ in (d). Fig. 4(b) represents a Catalyst pattern, where two e-sets $\{x,y\}$ and $\{x,z\}$ facilitates each other resulting in status changes from *Low* to *High* for both of them. Fig. 4(c) represents a Cluster pattern showing that three e-sets $\{x\}$, $\{y\}$ and $\{z\}$ have *Low* status individually but *High* status together.

Data	Retail -Data	BMS -WebView-1	BMS -WebView-2
NumberOfTrans	88,163	59,602	77,512
Distinction Items	16,470	497	3,340
MaxTransSize	67	267	161
AverageTransSize	15	2.5	5.0

Table 7: Real datasets

6.4 Results and Evaluation

The experimental results demonstrate that FITIN provides a sound and useful means of finding complex and random IIP-tree patterns within an FP-tree.

Test results, as shown in Table 8, demonstrate that FITIN is able to reveal itemset interactions of potential interest. Pattern (a), (b) and (d) exist in both the real and synthetic datasets, while pattern (c) exists in two synthetic datasets. Presented below are some itemset interaction examples discovered from Retail-Data (each item is denoted as a character c plus a number):

- $\{c39, c2925\}$ [$\sigma_R = 1.1\%$], $\{c39, c1146\}$ [$\sigma_R = 1.1\%$], $\{c39, c2925, c1146\}$ [$\sigma = 0.009\%$]
Description: Itemset $\{c39, c2925\}$ competes with $\{c39, c1146\}$.
- $\{c14945, c101\}$ [$\sigma_R = 0.5\%$], $\{c271, c270\}$ [$\sigma_R = 0.8\%$], $\{c14945, c101, c271, c270\}$ [$\sigma = 0.005\%$]
Description: Itemset $\{c14945, c101\}$ competes with $\{c271, c270\}$.
- $\{c39\}$ [$\sigma_R = 24.4\%$], $\{c48\}$ [$\sigma_R = 14.7\%$], $\{c39, c48\}$ [$\sigma = 33.1\%$]
Description: Itemset $\{c39\}$ facilitates itemset $\{c48\}$.

As shown in Fig. 5, the number of matched instances is affected by the setting of min_{HF} and max_{LF} . The higher min_{HF} is, the more itemsets with lower support are pruned out, and therefore, the fewer matched instances found. Similarly, the higher max_{LF} is, the fewer significant status changes in an itemset interaction will occur, and therefore, more matches can be identified.

7 Conclusions and Future Work

This paper outlines an approach to finding itemset interactions in large databases. It represents an itemset interaction pattern (P_{II}) as a tree structure and searches an FP-tree for its matched instances. The experimental results demonstrate the capacity of this approach to find itemset interactions of potential interest that cannot be identified by other data mining techniques.

The focus of the proof-of-concept implementation was not on performance but on proving that the design decision was sound. A more efficient algorithm is planned to cope with more complex itemset interaction patterns.

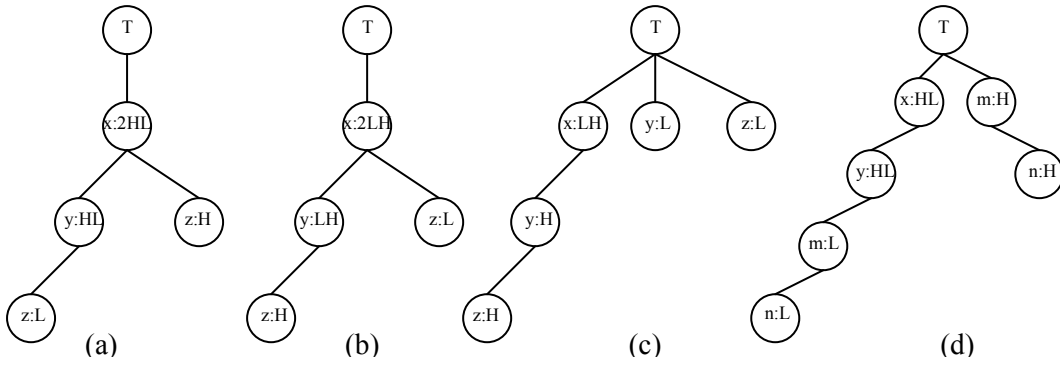


Figure 4: Patterns in tests

Test Datasets	Min -sup	FP-tree Info			Pattern (a)			Pattern (b)			Pattern (c)			Pattern (d)		
		Max Depth	NumOf Branch	NumOf Nodes	min -HF	max -LF	#	min -HF	max -LF	#	min -HF	max -LF	#	min -HF	max -LF	#
RetailData	0.01	12	12,142	31,037	1.0	0.01	6	1.0	1.0	1	1.0	0.5	0	0.5	0.5	48
BMS-WebView1	0.01	31	5,584	16,909	0.5	0.5	11	0.1	0.8	52	0.5	0.5	0	0.8	0.2	10
BMS-WebView2	0.005	28	14,044	48,571	0.5	0.5	5	0.5	0.5	0	0.5	0.5	0	0.8	0.2	8
Syn1	0.2	46	5,842	96,159	1.0	0.5	54	2.0	0.5	32	1.0	0.5	44	1.0	0.5	10
Syn2	0.05	23	17,426	124,763	0.1	1.0	144	0.2	0.2	132	0.0	1.0	3	0.8	0.2	2
Syn3	0.1	20	11,699	52,897	0.5	0.5	97	0.5	0.5	2	0.5	0.5	0	0.5	0.5	270

Table 8: Test results

The evaluation of interestingness of a matched itemset interaction pattern instance is currently based on status changes, which are also enhancing. For example, it would potentially be useful to detect itemset interactions with the most significant status changes.

References

- Aggarwal, C. & Yu, P. (1998), A new framework for itemset generation, in ‘ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems’, pp. 18–24.
- Agrawal, R., Imielinski, T. & Swami, A. (1993), Mining association rules between sets of items in large databases, in P. Buneman & S. Jajodia, eds, ‘ACM SIGMOD International Conference on the Management of Data’, Washington D.C., U.S.A, pp. 207–216.
- Agrawal, R. & Srikant, R. (1994), Fast algorithms for mining association rules, in J. Bocca, M. Jarke & C. Zaniolo, eds, ‘20th International Conference on Very Large Data Bases, VLDB’94’, Morgan Kaufmann, Santiago, Chile, pp. 487–499.
- Antonie, M. & Zaiane, O. (2004), Mining positive and negative association rules: an approach for confined rules, in ‘Intl. Conf. on Principles and Practice of Knowledge Discovery in Databases’, pp. 27–38.
- Bayardo, R. (1998), Efficiently mining long patterns from databases, in ‘ACM SIGMOD’, Seattle, WA, pp. 85–93.
- Brijs, T., Swinnen, G., Vanhoof, K. & Wets, G. (1999), The use of association rules for product assortment decisions: a case study, in ‘Fifth International Conference on Knowledge Discovery and Data Mining’, San Diego, USA, pp. 254–260.
- Brin, S., Motwani, R. & Silverstein, C. (1997), Beyond market basket: Generalizing association rules to correlations, in ‘ACM SIGMOD Int. Conf. Management of Data (SIGMOD97)’, pp. 265–276.
- Brin, S., Motwani, R., Ullman, J. & Tsur, S. (1997), Dynamic itemset counting and implication rules for market basket data, in ‘ACM SIGMOD’, Tucson, AZ, pp. 255–264.
- Ceglar, A. & Roddick, J. F. (2006), ‘Association mining’, *ACM Computing Surveys* **38**(2).
- Daly, O. & Taniar, D. (2004), ‘Exception rules mining based on negative association rules’, *Lecture Notes in Computer Science* **3046**, 543–552.
- Gunopulos, D., Mannila, H. & Saluja, S. (1997), Discovering all most specific sentences by randomised algorithms extended abstract, in F. Afrati & P. Kolaitis, eds, ‘the 6th International Conference on Database Theory’, Springer Verlag, Delphi, Greece, pp. 229–251.
- Han, E., Karypis, G. & Kumar, V. (1997), Scalable parallel data mining for association rules, in ‘ACM SIGMOD’, Tucson, AZ, pp. 277–288.
- Han, J. & Kamber, M. (2000), *Data Mining Concepts and Techniques*, Morgan Kaufmann.
- Han, J., Pei, J. & Yin, Y. (2000), Mining frequent patterns without candidate generation, in ‘ACM SIGMOD’, Dallas, TX, pp. 1–12.
- Kohavi, R., Brodley, C., Frasca, B., Mason, L. & Zheng, Z. (2000), Kdd-cup 2000 organizers’ report: Peeling the onion, Vol. 2, SIGKDD Exploration, pp. 86–93.
- Lee, Y.-K., Kim, W.-Y., Cai, Y. D. & Han, J. (2003), Comine: Efficient mining of correlated patterns, in ‘Int. Conf. Data Mining (ICDM’03)’, pp. 581–584.
- Liu, H., Lu, H., Feng, L. & Hussain, F. (1999), Efficient search of reliable exceptions, in ‘Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD99)’, pp. 194–203.
- Liu, J., Pan, Y., Wang, K. & Han, J. (2002), Mining frequent item sets by opportunistic projection, in ‘Knowledge Discovery in Databases’, Vol. 31, ACM Press, Edmonton, Canada, pp. 97–102.
- Park, J., Chen, M. & Yu, P. (1995), An effective hash-based algorithm for mining association rules, in ‘ACM SIGMOD’, San Jose, CA, pp. 175–186.
- Piatetsky-Shapiro, G. (1991), ‘Discovery, analysis and presentation of strong rules’, *Knowledge Discovery in Databases* pp. 229–248.

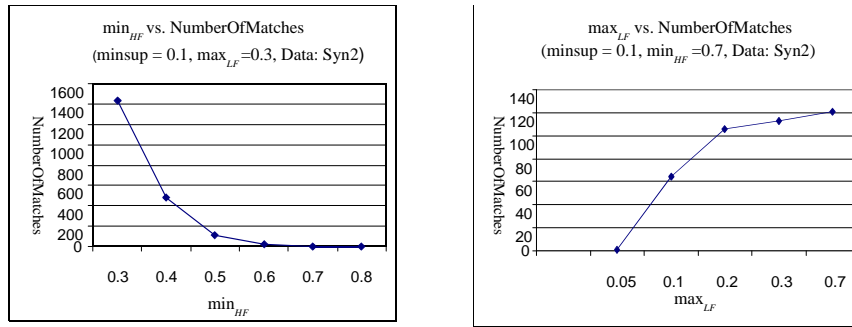


Figure 5: Performance of FITIN

- Roddick, J. F., Spiliopoulou, M., Lister, D. & Ceglar, A. (2008), 'Higher order mining', *SIGKDD Explorations* **10**(1), 5–17.
- Savasere, A., Omiecinski, E. & Navathe, S. (1998), Mining for strong negative associations in a large database of customer transactions, in 'Intl. Conf. on Data Engineering', pp. 494–502.
- Shillabeer, A. & Pfitzner, D. (2007), Determining pattern element contribution in medical datasets, in 'Australasian Workshop on Health Knowledge Management and Discovery (HKMD 2007)', Vol. 68 of *CRPIT*, ACS, Ballarat, Australia.
- Teng, C. (2002), Learning from dissociations, in Y. Kambayashi, W. Winiwarter & M. Arikawa, eds, 'DaWaK 2002', pp. 11–20.
- Toivonen, H. (1996), Sampling large databases for association rules, in 'VLDB', Bombay, India, pp. 134–145.
- Wei, Q. & Chen, G. (2000), Association rules with opposite items in large categorical database, in 'Intl. Conf. on Flexible Query Answering Systems', pp. 507–514.
- Xu, X., Zhang, C. & Zhang, S. (2004), 'Efficient mining of both positive and negative association rules', *ACM Transactions on Information Systems* **22**(3), 381–405.
- Yan, P., Chen, C., Cornelis, C., De Cock, M. & Kerre, E. (2004), 'Mining positive and negative fuzzy association rules', *Lecture Notes in Computer Science* **3213**, 270–276.
- Yuan, X., Buckles, B., Yuan, Z. & Zhang, J. (2002), Mining negative association rules, in 'Seventh Intl. Symposium on Computers and Communication', Italy, pp. 623–629.
- Zaki, M. J. & Hsiao, C.-J. (2002), Charm: An efficient algorithm for closed itemset mining, in 'the Second SIAM International Conference on Data Mining', ACM Press, Arlington, Vancouver, pp. 457–473.