An Extended Frequent Pattern Tree for Intertransaction Association Rule Mining

Sebastian Lühr Geoff West Svetha Venkatesh
Institute for Multi-Sensor Processing and Content Analysis
Department of Computing, Curtin University of Technology, Perth, Western Australia
{S.Luhr, G.West, S.Venkatesh}@curtin.edu.au

Technical Report 2005/1
March 2005; Revised October 2007

Abstract

We propose the Extended Frequent Pattern Tree (EFP-Tree) to address the problem of intertransaction association rule mining where the frequent occurrence of a large number of items results in a combinatorial explosion that limits the practical application of the existing Apriori inspired mining algorithms in a smart home environment. The EFP-Tree mining algorithm avoids candidate generation by employing a divide and conquer approach that recursively finds the set of frequent intertransaction association rules. Empirical results comparing the computational performance of the EFP-Tree with the First Intra Then Inter (FITI) algorithm on real world data from a smart home are presented. Experimental results show significant computational improvement of the EFP-Tree over FITI when a large number of rules is present in the data.

1 Introduction

*Inter*transaction association rule mining [7] extends the discovery of *intra*transaction associations to include relationships that span transactions in one or more domain specific dimensions. The dimensional attribute may be, for example, temporal in the prediction of stock market movements or spatial in a GIS application. In our work, we seek to find intertransaction associations from the event logs in a smart home environment for the detection and analysis of emergent human behaviours [8, 9]. Such an intelligent environment generates many sensor events over short periods of time which results in dense data sets where the number of frequently occurring events can be numerous. The existing algorithms [6, 12] for the mining of intertransaction associations, or itemsets, have adopted the Apriori [1] approach in which knowledge about the frequency of k length itemsets is used to restrict the search space in the discovery of $k+1$ length associations. These algorithms may generate a large number of candidate itemsets, especially at lower support thresholds and in dense data sets where the number of generated candidates may be vast. The computational complexity of testing the frequency of the candidates, many of which may be infrequent, can become intractable.

Addressing this issue, in this work we show how pattern growth may be employed as an alternative to the Apriori based approach for the mining of intertransaction associations using the Extended FP-Tree (EFP-Tree), an adaptation of the Frequent Pattern Tree (FP-Tree) [4] to the intertransaction association rule mining problem. The pattern growth approach in the FP-Tree is a more computationally efficient method for association rule mining that eliminates the need for candidate itemset generation by first transposing a transactional database into a intermediate form that aids subsequent mining. The FP-Tree structure and mining algorithm, however, only finds frequent intratransaction association rules and is not suitable to intertransaction rule mining.

Experimental results showing significantly improved computational performance over the FITI [12] algorithm on synthetic dense data and on real world data captured in a smart home are presented.

2 Related Work

The discovery of intertransaction association rules was first proposed with the E-Apriori and EH-Apriori algorithms [6]. These Apriori inspired approaches make multiple passes over the database to find the set of frequent as-

sociation rules, first generating and then searching for candidate k-itemset associations given known frequent $(k-1)$-itemsets discovered in the previous pass. The generation and then verification of the frequency of candidate itemsets by these algorithms restricts their scalability as the number of candidate itemsets generated may be prohibitive [4].

The First Intra Then Inter (FITI) algorithm [12] is a more efficient E-Apriori-like algorithm that initially finds the complete set of frequent intratransaction itemsets as a basis for transforming the database into a structure that aids subsequent mining of the intertransaction itemsets.

Intertransaction mining was reformulated in [5] to consider the problem of mining frequent continuities, a type of periodic pattern, from event sequences. The work introduces frequent continuity mining as a special case of intertransaction mining in which event sequences are represented as contiguous transactions of single event items. The proposed PROjected Window Lists (PROWL) algorithm was shown to outperform FITI when it was applied to this special case.

Similar to our work is the use of event trees, an FP-Tree like structure where event nodes are flagged as belonging either to the current transaction or a future transaction [10]. This was applied to finding relations between financial events and stock market movement using event episodes. In this work only the existence of events within the sliding window is important, the ordering or temporal relationship of events is not considered. In contrast, the EFP-Tree specifically encodes these temporal relationships.

The FP-Tree has also been applied to the prediction of rare events [2]. FP-Trees conditioned on the occurrence of specific rare events in a transaction database were constructed using a codebook whose alphabet encodes both an event and its temporal offset in relation to the event being predicted. The standard FP-Growth algorithm was used to mine the associations and the codebook used to recover the frequent intertransaction associations occurring prior to the event being conditioned on. This technique requires each FP-Tree to be conditioned on a single rare event and can not be efficiently applied to the general case intertransaction association rule mining problem.

3 Intertransaction Association Rules

Intertransaction association rules describe association relationships that span outside traditional "market basket" intratransaction items or events in one or more domain specific dimension. A video store, for example, may be interested in how future rentals are influenced by the items shoppers have borrowed in the past while a stock market analyst may wish to find associations of market events to predict movement in share prices.

Consider the set of all items $I = \{a^1 a^2 \dots a^i \dots a^M\}$ occurring in a database $DB = \langle T_1 T_2 \dots T_N \rangle$ of transactions T_i $(1 \leq i \leq N)$ such that $T_i\,(x) \in I\ \forall$ items x in T_i. At any transaction T_i the items are said to form the set $S_{T_i} = \{a^i_{T_i} \dots a^k_{T_i}\}$. For the case of a single intertransaction dimension attribute, an intertransaction sliding window of size w transactions is passed over the transactions in DB to extract the extended transaction items such that the extended transaction at T_i is $E_{T_i} = \{S_{T_i}, S_{T_i+1} \dots S_{T_i+w}\}$ and the set of all possible extended transaction items is $E = \{a^1_0 a^2_0 \dots a^i_d \dots a^M_w\}$. The mining problem reduces to the traditional intratransaction case when $w = 0$, that is, when only intratransaction items are included in an extended transaction itemset.

The superscript notation is dropped when the value of an item is known. For example, the extended transaction items retrieved with a sliding intertransaction window of size $w = 5$ starting at transaction ID 300 from the example database in Table 1 are C_0, B_0, A_0, C_2, E_2, B_3, E_3, B_4, and A_4, given that the dimensional attribute is the transaction time.

In this work we refer to the extended transaction items $\{a^0_1 \dots a^0_M\}$ as intraitems and the extended transaction items $\{a^1_1 \dots a^w_M\}$ as interitems.

Intertransaction association rules are implication rules such that $X \Rightarrow Y$ with the following properties [12]

$$X \subseteq E, Y \subseteq E \tag{1}$$

$$\exists a^0_i \in X \tag{2}$$

$$\exists a^d_i \in Y, d > 0 \tag{3}$$

$$X \cap Y = \emptyset \tag{4}$$

The support and confidence measures of an itemset are calculated as $\frac{|T_{xy}|}{N}$ and $\frac{|T_{xy}|}{|T_x|}$ respectively where $|T_{xy}|$ is the number of extended transactions containing all items in $X \cup Y$, $|T_x|$ is the number of extended transactions containing all items in X and N is the number of extended transactions.

4 Extended Frequent Pattern Tree

The proposed Extended Frequent Pattern Tree (EFP-Tree) is a tree structure of descending frequency ordered intraitem nodes with zero or one interitem Frequent Pattern Tree (FP-Tree) subtrees where the frequency ordering of the interitems is conditioned on the intratransaction item parent. Each node in the tree contains an item ID which maps to a codebook of item descriptors, a frequency counter, a link to its parent node, links to zero or more children and a link to the next node in the tree of the same item ID. Interitem nodes also carry the dimensional offset of the item in relation to its intratransaction parent.

Nodes are placed into the tree such that the entire set of frequent items for an arbitrary intertransaction can be restored by traversing the tree. The ordering of nodes into descending frequency increases the likelihood of items placed

Table 1. Example database with the unsorted and descending frequency ordered items. The time at which each transaction occurs is shown.

Trans. ID	Time	Raw Items	Ordered Items
100	1	A C B	B C A
200	2	B	B
300	3	C A B	B C A
400	5	E C	C E
500	6	B E	B E
600	7	A B	B A
700	9	C	C
800	10	C D B	B C D
900	11	C B A	B C A

Table 2. Extended transactions retrieved from the example database in Table 1 using a sliding intertransaction window of size $w = 5$.

Time	Extended transaction items
1	B_0 C_0 A_0 B_1 B_2 C_2 A_2 C_4 E_4 B_5 E_5
2	B_0 B_1 C_1 A_1 C_3 E_3 B_4 E_4 B_5 A_5
3	B_0 C_0 A_0 C_2 E_2 B_3 E_3 B_4 A_4
5	C_0 E_0 B_1 E_1 B_2 A_2 C_4 B_5 C_5 D_5
6	B_0 E_0 B_1 A_1 C_3 B_4 C_4 D_4 B_5 C_5 A_5
7	B_0 A_0 C_2 B_3 C_3 D_3 B_4 C_4 A_4
9	C_0 B_1 C_1 D_1 B_2 C_2 A_2
10	B_0 C_0 D_0 B_1 C_1 A_1
11	B_0 C_0 A_0

into the tree sharing common nodes, creating a compact representation of the database transactions that captures the associative relationship of the transaction items.

4.1 Tree Construction

Three passes, detailed in Algorithm 1, over a database are required to build the tree structure. As in the FP-Tree, the frequency of single items is gathered in an initial pass over the database to build the set of frequent single intraitems, or 1-itemsets, and the set of frequent interitems given a minimum support threshold. The intraitems are ordered by descending frequency to become the item lookup header table for the intraitem tree. The frequent items are those items whose frequency count is greater than or equal to the minimum support threshold α.

The intraitem FP-Tree is built and the conditional frequencies of the interitems are found in the second pass. The intraitems for each transaction T_i are first filtered to remove items not present in the known frequent intraitem set from the first pass and are sorted in order of descending frequency. The ordered item list is recursively inserted into the tree such that at each level l in the tree the child node with the ID of the l^{th} item in the ordered array is traversed and its frequency count is incremented. Children nodes that do not exist will be created prior to traversal and kept in codebook ID order so that binary search can be used when traversing the tree. The linked list of nodes of same item ID that originates from the root node header table is updated whenever a new node is created. The frequency of the interitems relative to T_i are incremented in the final intraitem node that is traversed. The ordered lists of frequent intraitems are stored for use in the third pass. The root node is said to be at level $l = 0$.

The third and final pass over the database builds the interitem sub-trees in the EFP-Tree structure. At each transaction T_i the cached ordered list of frequent intraitems are used to traverse the intraitem tree and locate the intraitem node that will become the root node of the interitem subtree. The extended items within the intertransaction sliding window at T_i are filtered to remove known infrequent interitems and the remaining items are sorted in order of local descending frequency given the intraitem parent. The ordered interitems are then recursively inserted into the interitem subtree as before.

The example database in Table 1 and Table 2 is used to demonstrate the construction of the EFP-Tree structure in Figure 1 using a minimum support threshold of $\alpha = 3$ and a sliding intertransaction window size of $w = 5$.

The frequent items are found in the first pass over the database from Table 2. The frequent intraitems found given the minimum support threshold $\alpha = 3$ are B_0:7, C_0:6 and A_0:4 with frequency counts of 7, 6 and 4 respectively. The frequent interitems meeting the same minimum support threshold are A_1:3, A_2:3, B_1:6, B_2:3, B_4:4, B_5:4, C_1:3, C_2:4, C_3:3 and C_4:4.

In the second pass, the intraitems B_0, C_0 and A_0 are added to the root node of an empty tree such that all nodes are recursively created. The frequency of item B_0 is incremented by the next transaction. The items B_0, C_0 and A_0 are then added again, the existing nodes are traversed and their frequency counts are each incremented. The next two transactions see the node C_0 created as the second child of the root node and an increment to the count of B_0. B_0 and A_0 are then added such that A_0 becomes the second child of B_0 and the count of B_0 is incremented once more. The count of the nodes representing the intratransaction associations C_0 and B_0, C_0 are incremented by the next two transactions. Finally, the counts of the items in the path B_0, C_0, A_0 are once again incremented.

The third and final pass over the example database now begins. The known frequent items found in the first pass

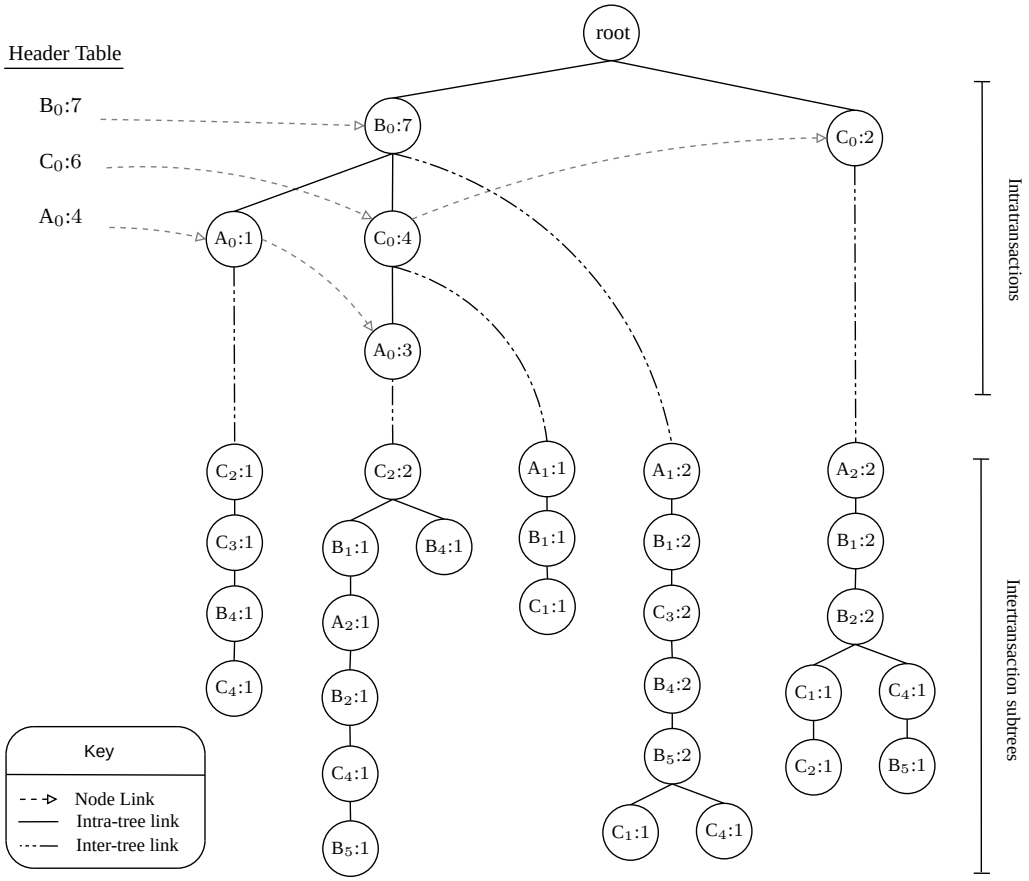

Figure 1. The Extended FP-Tree for the example database with $w = 5$ and $\alpha = 3$. Subscript numbers represent the dimensional offset, in this case the item time, relative to the intratransaction items while colon delineated numbers depict the node frequency. The header tables of the intertransaction item subtrees have been omitted for clarity.

allow the extended transaction items at time 1 to be reduced to B_0, C_0, A_0, B_1, A_2, B_2, C_2, C_4 and B_5 given that the items E_4 and E_5 are known to not meet the minimum support threshold. The interitems will be inserted at the node identified by following the path B_0, C_0, A_0. The insertion of the ordered list of interitems from the first extended transaction will create the children interitem nodes C_2, B_1, A_2, B_2, C_4 and B_5. Next, the extended items at time 2 are reduced to A_1, B_1, C_1, C_3, B_4 and B_5. The items are ordered and inserted as new children of B_0. The nodes B_0, C_0 and A_0 will again be traversed at time 3. The child node C_2 will then be incremented and a new node for item B_4 will be inserted as a child of C_2.

The interitems from Table 2 at time 5 are reduced to B_1, A_2, B_2, C_4 and B_5 and inserted as new children of C_0. Next, the interitems at time 6 are filtered and inserted as children of B_0 such that the counts of A_1, B_1, C_3, B_4 and B_5 are each incremented and the node C_4 is inserted as a new child of B_5. At time 7 the intraitems B_0 and A_0 are traversed and the interitems C_2, C_3, B_4 and C_4 are inserted as children of A_0. The next transaction at time 9 sees the count of the nodes A_2, B_1 and B_2, as children of C_0, incremented and the interitems C_1 and C_2 inserted as children of B_2. Finally, the nodes B_0, C_0 are traversed and the frequent interitems from the extended transaction at time 10 are appended.

No interitems exist in the extended items in Table 2 at time 11 so no further action is required.

4.2 Rule Mining

A method to extract the intertransaction associations present in the EFP-Tree structure and to generate the IARs is now required. As in the FP-Tree, retrieval of association rules from the EFP-Tree is made possible by the pattern growth property [3, 4]. Pattern growth uses a divide and conquer approach that recursively builds the entire set of frequent associations by constructing trees conditioned on known frequent base rules and taking the dot product of the

Algorithm 4.1: EFP-Tree Construction

Input: Transaction Database *DB*, sliding intertransaction window size *maxSpan*, tree building support threshold *minSupport*
Output: EFP-Tree *tree*

Method:

// First Pass
Count the frequency of single items in *DB* to build the set of frequent items *F* such that support(F_i) $\geq$ *minSupport* ($1 \leq i \leq |F|$)

// Second Pass
Create the intratransaction root node *tree*
For each intratransaction *t* in *DB* **do**
  A $\leftarrow$ intratransaction items such that $A_i \in F$ ($1 \leq i \leq |A|$) ordered by descending frequency
  I $\leftarrow$ count of intertransaction items | *t*, *maxSpan*
  recursively insert the nodes *A* onto *tree* finishing at node *interParent*
  interParent.interFreq $\leftarrow$ *interParent.interFreq* + *I*
end

// Third Pass
For each intratransaction *t* in *DB* **do**
  A $\leftarrow$ intratransaction items such that $A_i \in F$ ($1 \leq i \leq |A|$) ordered by descending frequency
  interParent $\leftarrow$ intraitem node in *tree* corresponding to *A*
  E $\leftarrow$ intertransaction items such that $E_i \in F$ ($1 \leq i \leq |E|$) ordered by descending frequency | *leaf.interFreq*
  create the root node *interParent.interTree*
  recursively insert the nodes *E* onto *interParent.interTree*
end

Algorithm 4.2: EFP-Growth

Input: EFP-Tree N, mining support threshold α, sliding intertransaction window size *maxSpan*
Output: Set of frequent rules
Method:

minedRules $\leftarrow \emptyset$;
For each item a_i in header table of N from least to most frequent such that support $(a_i) \geq \alpha$ **do**
  Find the conditional prefix path and the extended items for a_i, propagate the intertransaction items of each occurrence of a_i to its parent and build the conditional EFP-Tree T_c;
  If T_c contains a single intratransaction path P such that no non-leaf node contains an intertransaction subtree **then**
    $T_c \leftarrow T_c$ with P removed;
    singlePathRules $\leftarrow$ all combinations of intratransaction nodes in P;
    singlePathRules $\leftarrow$ *singlePathRules* $\times$ rules returned by call to FP-Growth (leaf node of P, null) as in [4];
  end
  returnedRules $\leftarrow$ call EFP-Growth (T_c, α, *maxSpan*);
  Build the intertransaction item FP-Tree T_e using the extended items from a_i;
  interRules $\leftarrow$ call FP-Growth (T_e, null) as in [4];
  ruleSet $\leftarrow$ *returnedRules* $\cup$ *interRules* $\cup$ *singlePathRules*;
  For each rule $R \in$ *ruleSet* **do**
    add a_i to R with
    support (R) = min (support (R), support (a_i));
  end
  add a_i to *ruleSet* with support = frequency of a_i in N;
  minedRules $\leftarrow$ *minedRules* $\cup$ *ruleSet*;
end
Return *minedRules*

frequent items in the conditional tree and the conditional base itemset to produce new rules. These new rules then become the conditional base for the next set of conditional trees to be mined.

The Frequent Pattern Growth (FP-Growth) algorithm from the FP-Tree differs to the Extended FP-Growth (EFP-Growth) algorithm used to mine the EFP-Tree in that the latter must consider intertransaction relationship inheritance along the intraitem nodes.

Property 4.1. (Intertransaction inheritance property) Intratransaction item nodes inherit the intertransaction item relationships of their intratransaction item children.

Interitems are inserted into a subtree whose root node is the last intraitem node traversed to when an extended transaction itemset is sorted into descending frequency order and placed into the EFP-Tree. As a given interitem subtree can only be reached by traversing the intraitem tree in the presence of all parent intraitem nodes it follows that the relationship between an intraitem node and the items in the subtree must apply to all nodes traversed to reach the intertransaction item subtree.

Example 4.1. The extended transaction $L = \mathrm{A}_0, \mathrm{B}_0, \mathrm{C}_0, \mathrm{B}_1, \mathrm{C}_3$ is inserted as an ordered item list into an empty tree. The item nodes A_0, B_0 and C_0 are created as a single branch in the intraitem tree and the items B_1 and C_3 are in turn inserted as interitem nodes as children of C_0. Given Property 4.1 we can infer that there exists a relation $\mathrm{A}_0 \Rightarrow B_1$ which is known to exist as both $\mathrm{A}_0 \in L$ and $\mathrm{B}_1 \in L$. □

The EFP-Growth algorithm, detailed in Algorithm 4.2, will now be described. Starting with an EFP-Tree T and an empty conditional base, or rule suffix, EFP-Growth iterates over the set of intraitems I in T to build a conditional tree T_c conditioned on I for each frequent I. At each recursion, I is prepended to the conditional base to generate, or grow, a new association rule and to build the conditional tree for the next recursive step. No candidate generation is necessary as the frequency of the items is stored in the tree structure and all generated rules are guaranteed to be frequent.

Two types of conditional tree are used in EFP-Growth, a conditional EFP-Tree T_c used for finding the related intraitems and interitems that can be used to extend the present intraitem rule suffix and a FP-Tree T_e of the interitems inherited by the conditional base. This latter tree is used to find the interitem associations for a given intraitem rule suffix and is required as not all interitems inherited by the conditional base may be included in T_c.

Given a tree T, the conditional tree T_c conditioned on some I is found by collecting the set of extended transactions formed through the union of the prefix path and the inherited interitems for each node in T whose item ID is I and whose immediate parent is an intraitem node. The pre-

fix path for any given node is the set of its parent nodes and corresponding frequencies as stored in the EFP-Tree. All nodes in T of item ID I are found by following the linked list of "same item" ID nodes, the head of which is stored in the intraitem header table of T. The extended transactions are then used to build T_c as described in Section 4.1.

The conditional interitem tree T_e for a given conditional base is found by constructing an FP-Tree of the interitem transactions inherited by the conditional base rule and using FP-Growth to mine the resulting tree. Taking the dot product of the conditional base and the set of interitem associations returned by FP-Growth produces the entire set of intertransaction associations related to the conditional base.

This process continues recursively until no more conditional trees are built or until only a single intraitem path exists in T_c. If T_c contains a single intraitem path we can avoid recursion and find the complete set of rules given the conditional base by finding the dot product of the intraitem combinations in T_c and the interitem associations returned when calling FP-Growth on the interitem subtree.

Although the example EFP-Tree T in Figure 1 was built with a minimum support level of $\alpha = 3$, the support threshold will be set to $\alpha = 2$ for mining the tree in order to demonstrate the mining process in finer detail than is possible at the original support setting.

Conditioning T on A_0, we find the conditional prefix paths $\langle B_0{:}1\rangle$ and $\langle B_0{:}3\ C_0{:}3\rangle$. The intertransaction items $\langle C_2{:}1\ C_3{:}1\ B_4{:}1 C_4{:}1\rangle$ are related to $\langle B_0{:}1\rangle$ and the interitems $\langle C_2{:}1\ B_1{:}1\ A_2{:}1\ B_2{:}1\ C_4{:}1\ C_5{:}1\rangle$ and $\langle C_2{:}1\ B_4{:}1\rangle$ are found for $\langle B_0{:}3\ C_0{:}3\rangle$. The conditional tree $T_c|A_0$ with $\alpha = 2$ is shown in Figure 2(a) and the interitem FP-Tree $T_e|A_0$ in Figure 2(b).

The set of intertransaction rules associated with the conditional base A_0 is found by taking the dot product of A_0 and the interitem associations returned by FP-Growth from the tree in Figure 2(b). The resulting rules are $A_0 \Rightarrow C_2{:}3$, $A_0 \Rightarrow B_4{:}2$, $A_0 \Rightarrow C_4{:}2$, $A_0C_2 \Rightarrow B_4{:}2$ and $A_0C_2 \Rightarrow C_4{:}2$.

Recursing into the conditional tree in Figure 2(a), EFP-Growth grows the conditional base by finding the least frequent intraitem whose support meets the support threshold for mining. This item, C_0, is prepended to A_0 to create the new conditional base C_0A_0 and the intratransaction rule $C_0 \Rightarrow A_0{:}3$. The single interitem prefix path $\langle B_0{:}4\rangle$ and its inherited interitems $\langle C_2{:}1\ B_4{:}1\rangle$ and $\langle C_2{:}1\ C_4{:}1\rangle$ form to create the single intraitem path conditional tree shown in Figure 2(c). The inherited items of C_0A_0 are used to build a single node FP-Tree T_e containing $C_2{:}2$ resulting in the generation of a single rule $C_0A_0 \Rightarrow C_2{:}2$ when the dot product of the interitem associations found in $T_e|C_0A_0$ and the conditional base is found. Recursively mining the tree in Figure 2(c) generates the rules $B_0C_0 \Rightarrow A_0{:}2$ and $B_0C_0A_0 \Rightarrow C_2{:}2$.

The mining now returns to $T_c|A_0$ to create the next conditional base B_0A_0 and generate its respective rule $B_0 \Rightarrow A_0{:}4$. No prefix path of B_0A_0 exists in Figure 2(a) so no conditional tree $T_c|B_0A_0$ needs to be built. The mining of the interitem FP-Tree $T_e|B_0A_0$, the same as for $T_e|A_0$ in Figure 2(b), generates the rules $B_0A_0 \Rightarrow C_2{:}3$, $B_0A_0 \Rightarrow B_4{:}2$, $B_0A_0 \Rightarrow C_4{:}2$, $B_0A_0C_2 \Rightarrow B_4{:}2$ and $B_0A_0C_2 \Rightarrow C_4{:}2$.

Upon return from a recursive call EFP-Growth will update the immediate parent of each node whose item ID is I such that the interitems are inherited and ready for conditioning on the next frequent item. It is for this reason that the mining algorithm grows rules by recursing into trees conditioned on the least frequent intraitems first.

Returning to the original tree T in Figure 1 the recursive mining technique will be applied in turn to the conditional bases C_0 and B_0. The conditional trees $T_c|C_0$ and $T_e|C_0$ are given in Figure 2(d) and Figure 2(e) respectively.

IARs generated through EFP-Growth may be required to adhere to constraints present in the application domain. For example, the association $B_0C_4 \Rightarrow C_2{:}1$ makes little sense when the intertransaction attribute of the items is temporal and the rules are to be applied to prediction. In this case we can reorder the items to produce the association $B_0C_2 \Rightarrow C_4{:}1$ without affecting the accuracy of the support measure. The confidence measure of a reordered rule can be calculated by returning extra information during the recursive mining step.

5 Benchmark Comparisons

Both synthetic data, employed to model the best and worst case scenarios for association rule mining, and real world data sets, to indicate the practical application of the mining algorithms, were used to compare the computational performance and peak memory requirements of EFP-Growth with FITI. These performance measures are important as they empirically demonstrate the scalability of the algorithms on input data of varying characteristics. For each data set the ability of the algorithms to scale with respect to the length of the intertransaction window and a decreasing minimum support threshold is observed. [12] have previously shown FITI to be computationally more efficient than EH-Apriori and so the latter algorithm is not considered in this experiment.

The real world data used [11] are event logs from an array of state-change sensors installed in the homes of two volunteer subjects, a thirty year old working professional and an eighty year old retiree, over a period of fourteen days. The sensors, 77 in the first subject's home and 84 in the second, were fitted to a variety of appliances, containers and furniture to log the times of use. These events were discretised for mining into transactions of five minute intervals to produce 658 transactions for the first subject and

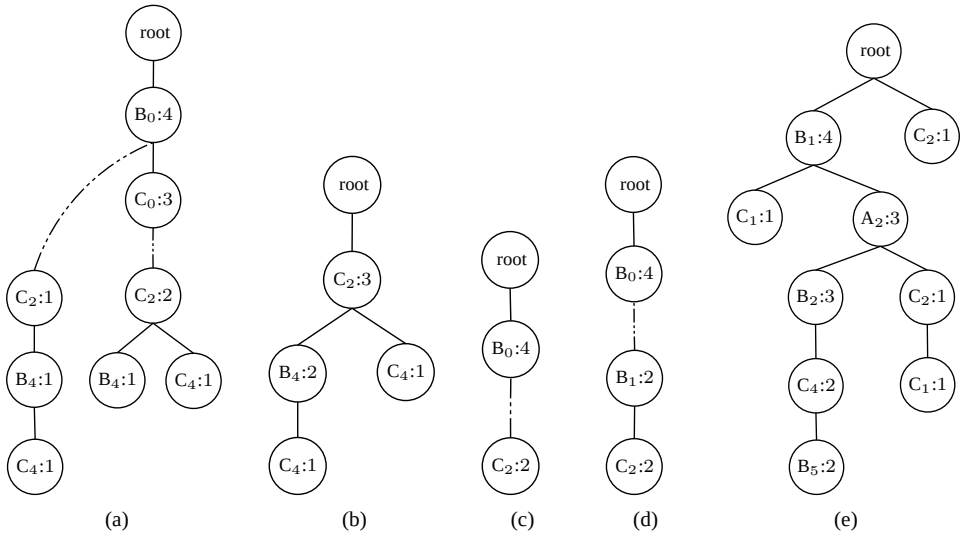

Figure 2. The conditional EFP-Tree tree T_c and conditional interitem FP-Tree T_e for (a) $T_c|\mathrm{A}_0$, (b) $T_e|\mathrm{A}_0$ and $T_e|\mathrm{B}_0\mathrm{A}_0$, (c) $T_c|\mathrm{C}_0\mathrm{A}_0$, (d) $T_c|\mathrm{C}_0$ and (e) $T_e|\mathrm{C}_0$ when the EFP-Tree in Figure 1 is mined with a minimum support threshold of $\alpha = 2$.

748 transactions for the second. Unique sensor IDs were stripped from the event logs to reduce the sensor information to only include the sensor state and its room and object context. For example, multiple sensors installed on the doors of a cabinet are reduced to `Kitchen/Cabinet true` and `Kitchen/Cabinet false` events. The event codebooks contained 76 and 80 entries for the first and second subjects respectively.

Two synthetic data sets representing sparse and dense data were generated using the method described in [6, 12], the same method used to compare the EH-Apriori algorithm to FITI. The data synthesis method is a two step process that first generates a pool of candidate intertransaction associations and then uses this pool to populate the transactional data set. The characteristics and features of the generated data is defined by several parameters that guide the generation process. These parameters include the size of the intertransaction pool, the mean and maximum length of the intertransaction associations, the maximum number of unique items that may be in the data set and the maximum interval span of the associations. Table 3 lists the parameters used to create the data sets used in the experimentation.

Intertransaction association rule mining in FITI occurs only after the set of frequent intratransaction associations have been found. Knowledge of these rules is then used to transform the database into a lookup structure that aids intertransaction mining. For this experiment, FITI was implemented using the FP-Tree and FP-Growth algorithm for the initial mining phase. This was necessary in order for a fair comparison of the algorithms to be made, it having previously been shown that FP-Growth performs an order of magnitude faster than the Apriori algorithm used in the original FITI implementation [3].

The algorithms were implemented in Ruby, an interpreted language, and benchmarked on a 3.2GHz Pentium 4 running FreeBSD.

5.1 Limitations of the Benchmark Environment

Before discussing results, execution time irregularities should be noted in the EFP-Growth curve in Figure 5(a) at $w = 2$ and at $w = 7$. Irregularities also appear for FITI in Figure 4(a) at the 1.1% support threshold, in Figure 4(b) at 1% support and in Figure 5(b) at $w = 4$.

Profiling revealed these irregularities are caused by an erratic garbage collector in the Ruby interpreter. When triggered, the garbage collector will spend a disproportionally long time seeking memory to free.

This behaviour was consistently reproduced on the FreeBSD 5.3, Linux 2.6 and Windows XP platforms using the 1.6 and 1.8 Ruby interpreter series. This behaviour is independent of the algorithm being run and was found present in the implementations of the EH-Apriori, FITI, FP-Growth and EFP-Growth algorithms. The garbage collector behaved normally for all other points on the graphs and hence the irregularities found do not invalidate the results obtained.

Table 3. Parameters used in the generation of the synthetic sparse and synthetic dense data sets.

Parameter	Sparse	Dense
Number of intratransactions	500	200
Size of the intertransaction source pool	50	200
Average length of intratransactions	5	25
Maximum length of intratransactions	10	50
Average length of intertransactions	5	8
Maximum length of intertransactions	10	20
Maximum number of unique items in the data	500	100
Maximum interval span of intertransactions	4	6

5.2 Minimum Support Threshold

For the first set of results, the support threshold was gradually lowered from 1.6% to 0.6% with a fixed intertransaction window size of 4 and from 13% to 8% with a fixed window size of 6 for the synthetic sparse and synthetic dense data sets respectively.

The plot in Figure 3(a) shows FITI outperforming EFP-Growth until the 1% support threshold is reached. FITI has an advantage at the higher support thresholds as it is able to remove unnecessary data prior to counting. This benefit is reduced as the number of candidates generated by FITI increases when the support threshold is lowered. EFP-Growth outperforms FITI at the lower support thresholds and especially at the 0.6% level where an explosion in the number of rules, as seen in Figure 3(e), results in an exponential increase in the number of candidate itemsets generated and counted by FITI. The plot of the memory requirements of the two algorithms in Figure 3(c) shows that although FITI has greatly reduced memory needs compared to EFP-Growth at the higher support levels, it is the latter algorithm that displays more stable memory use as the number of rules increases exponentially. FITI has lower memory requirements at the higher support levels because it is able to discard many known infrequent associations which results in a low number of candidates being generated.

We begin to see an order of magnitude difference in the algorithm execution times on the dense data in Figure 3(b). Although FITI marginally outperforms EFP-Growth at the 12.5% and 13% support threshold, FITI is overwhelmed by the number of candidate itemsets generated at the lower thresholds. The memory requirements for the dense data set in Figure 3(d) shows FITI has an advantage at all but the 9% and lower support levels. Here the memory usage continues to increase rapidly for FITI whereas the peak memory requirement of EFP-Growth remains stable. The number of rules discovered at each support level are shown in Figure 4(f).

The execution times in Figure 4(a) and peak memory usage in Figure 4(c) for the first real world data set compares the algorithms' performance as the support threshold is lowered from 1.5% to 0.7%. An order of magnitude difference in the running times exists at the lower support levels due to the large number of discovered rules and a high number of FITI generated candidates. An exponential increase in the number of rules discovered, shown in Figure 5(e), is reflected in a jump in the peak memory use of the two algorithms. Both algorithms use similar amounts of memory up until this point.

Figure 4(b) depicts the execution time of EFP-Growth and FITI on the second real world data set over a support threshold range of 0.4% to 1.3%. EFP-Growth is able to maintain its computational advantage over FITI at all support levels. Memory use, shown in Figure 4(d), sees FITI again having an advantage only at the higher support levels where the number of rules, shown in Figure 6(f), and hence the number of candidates generated remains low.

5.3 Intertransaction Sliding Window Size

The intertransaction window size in Figure 5 is incremented from $w = 0$ to $w = 10$ for the sparse data and $w = 8$ for the dense data with fixed minimum support thresholds of 1% and 10% respectively.

Figure 5(a) shows that EFP-Growth has only a marginal computational advantage on the sparse data set, the number of rules found and the number of candidates generated by FITI remaining relatively low. The memory requirements in Figure 5(c) are seen to be increasing at a similar pace with FITI requiring slightly less memory than EFP-Growth until $w = 9$. Figure 5(e) depicts the number of rules retrieved with each window size. FITI has similar execution times to EFP-Growth on the dense data in Figure 5(b) set until the intertransaction size $w = 5$. The curves begin to diverge at this point, the FITI execution time eventually being an order of magnitude greater than EFP-Growth at $w = 8$. The FITI memory needs, from Figure 5(d), are overall lower than that for EFP-Growth but are growing exponentially with respect to the sliding window size due to the number of candidate itemsets being created. The number of rules mined at each

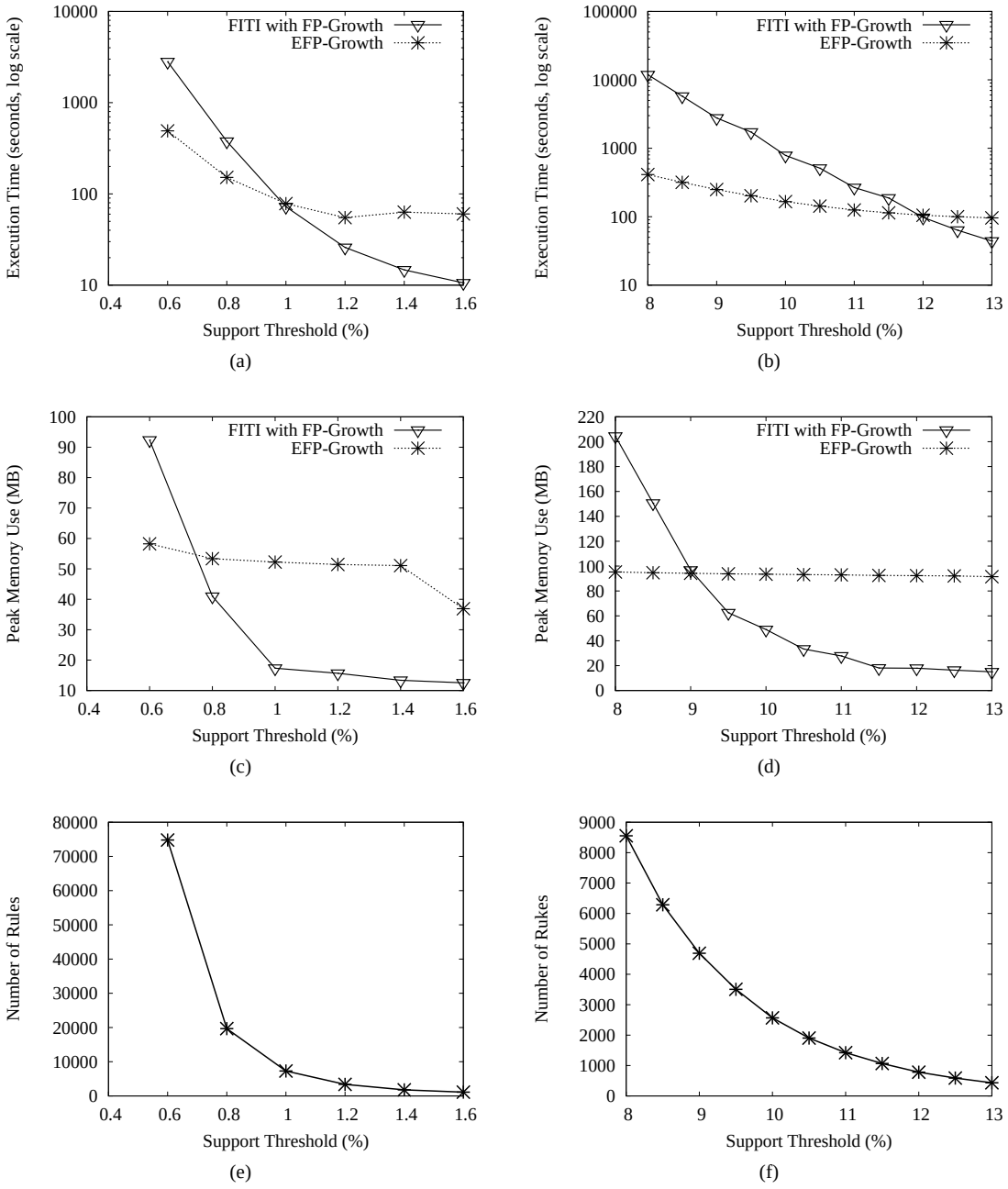

Figure 3. The execution time (a), memory use (c) and the number of rules found (e) for the synthetic sparse data set with the intertransaction window size fixed at $w = 4$ and the execution time (b), memory use (d) and the number of rules found (f) for the synthetic dense data set with $w = 6$ as the minimum support threshold is adjusted.

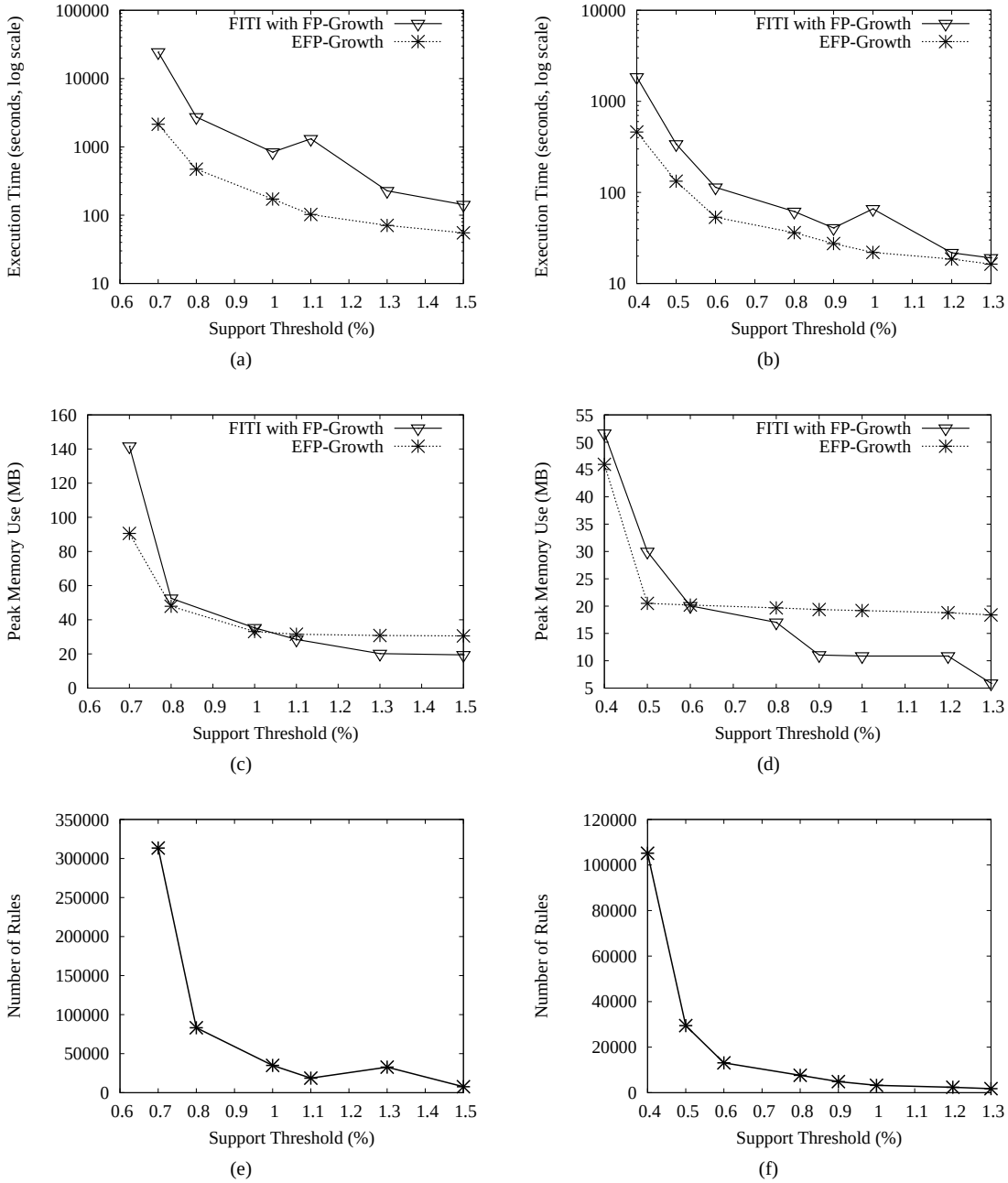

Figure 4. The execution time (a), memory use (c) and the number of rules found (e) for the working professional subject and the execution time (b), memory use (d) and the number of rules found (f) for the retiree subject as the minimum support threshold is adjusted. The intertransaction window size is fixed at $w = 6$ with an interval size of 300 seconds for both data sets.

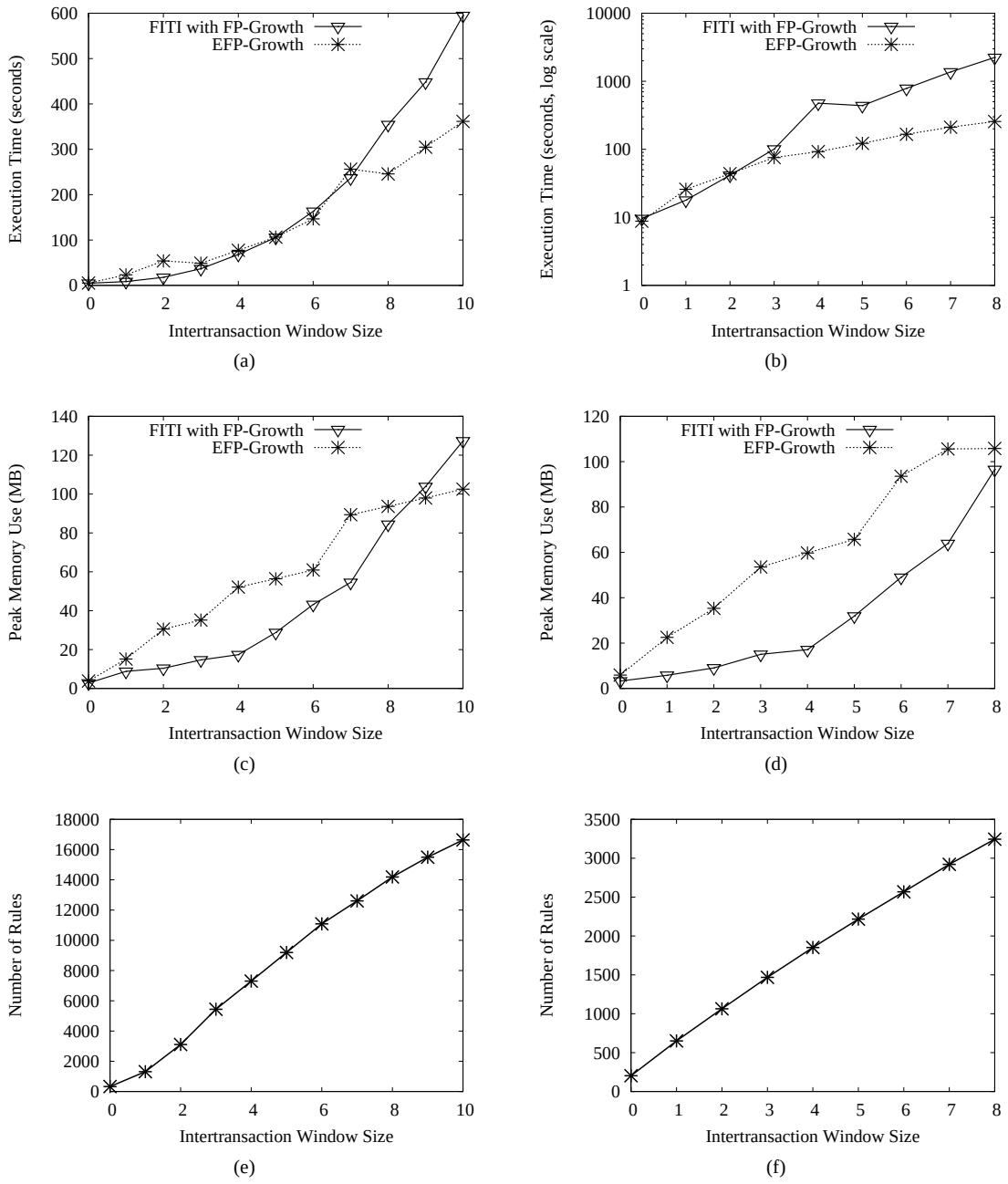

Figure 5. The execution time (a), memory use (c) and the number of rules found (e) for the synthetic sparse data set and the execution time (b), memory use (d) and the number of rules found (f) for the synthetic dense data set as the intertransaction window size is increased. The minimum support level was fixed at 1% and 10% for the sparse and dense data sets respectively.

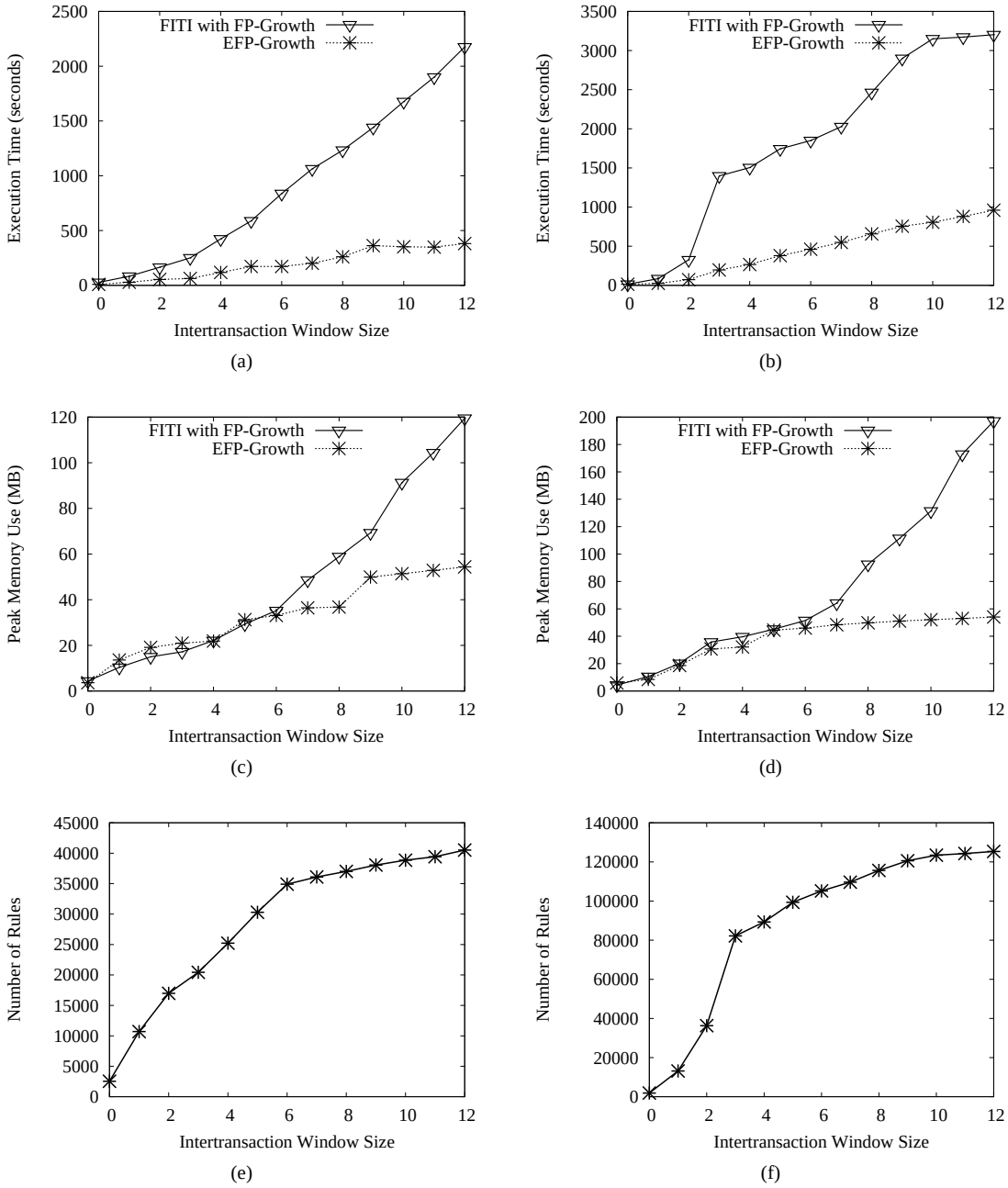

Figure 6. The execution time (a), memory use (c) and the number of rules found (e) for the working professional subject with a minimum support threshold $\alpha = 1\%$ and the execution time (b), memory use (d) and the number of rules found (f) for the retiree subject with $\alpha = 0.4\%$ as the intertransaction window size is increased.

window size are given in Figure 5(f).

Performance on the real world data is compared by incrementing the sliding window size up to $w = 12$ to find associations spanning up to an hour. The support thresholds are fixed at 1% and 0.4% for the first and second data sets.

EFP-Growth outpaced FITI computationally in both real world data sets in Figure 6(a) and Figure 6(b). A sudden increase in the execution time of the FITI algorithm is seen in Figure 6(b) when the sliding window size is increased from $w = 2$ to $w = 3$. This increase is caused by a sudden large jump in the number of rules being discovered as can be seen in Figure 6(e). The memory requirements of the algorithms remain similar until $w = 8$ for both Figure 6(c) and Figure 6(d). The EFP-Growth memory use remains stable while FITI continues to increase linearly as the number of rules, seen in Figure 6(e), being discovered begins to taper off at this point.

6 Conclusion

The EFP-Tree and EFP-Growth algorithms for intertransaction association rule mining have been introduced and experimental results comparing the computational performance of EFP-Growth to the First Intra Then Inter (FITI) algorithm have been presented.

EFP-Growth was shown to scale well, particularly with the synthetic dense data set and data from the home of a real world retiree where the presence of many frequent items results in a combinatorial explosion of candidate itemsets at the lower support thresholds and with the larger intertransaction window sizes. The benefit gained by FITI by pruning known infrequent intratransaction item combinations prior to intertransaction mining is diminished when the number of discovered rules and hence the number of candidate itemsets that are required to be counted grow too large.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int'l Conf. Very Large Data Bases*, pages 487–499, New York City, New York, USA, August 1994.

[2] C. Berberidis, L. Angelis, and I. Vlahavas. PREVENT: An algorithm for mining intertransactional patterns for the prediction of rare events. In *Proc. Second Starting AI Researchers' Symposium*, volume 9 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2004.

[3] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM SIGMOD Int'l Conf. Management of Data*, pages 1–12, 2000.

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

[5] K.-Y. Huang, C.-H. Chang, and K.-Z. Lin. PROWL: An efficient frequent continuity mining algorithm on event sequences. In *Proc. Int'l Conf. Data Warehousing and Knowledge Discovery*, volume 3181 of *Lecture Notes in Computer Science*, pages 351–360. Springer-Verlag, Nov 2004.

[6] H. Lu, L. Feng, and J. Han. Beyond intra-transaction association analysis: Mining multi-dimensional inter-transaction association rules. *ACM Trans. Information Systems*, 18(4):423–454, 2000.

[7] H. Lu, J. Han, and L. Feng. Stock movement prediction and n-dimensional inter-transaction association rules. In *Proc. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 12:1–12:7, Seattle, Washington, USA, June 1998.

[8] S. Lühr, S. Venkatesh, and G. West. Emergent intertransaction association rules for abnormality detection in intelligent environments. In *Int'l Conf. Intelligent Sensors, Sensor Networks and Information Processing*, pages 343–347, December 2005.

[9] S. Lühr, G. West, and S. Venkatesh. Recognition of emergent human behaviour in a smart home: A data mining approach. *Pervasive and Mobile Computing*, 3(2):95–116, March 2007.

[10] A. Ng and A. W. Fu. Mining frequent episodes for relating financial events and stock trends. In *Proc. Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining*, volume 2637 of *Lecture Notes in Computer Science*, pages 27–39. Springer-Verlag, 2003.

[11] E. M. Tapia, S. S. Intille, and K. Larson. Activity recognition in the home using simple and ubiquitous sensors. In *Int'l Conf. Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 158–175. Springer-Verlag, April 2004.

[12] A. K. H. Tung, H. Lu, J. Han, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Trans. Knowledge and Data Engineering*, 15(1):43–56, 2003.