行政院國家科學委員會專題研究計畫 成果報告

# 跨交易關聯規則演算法之研究
研究成果報告(精簡版)

計 畫 類 別	：個別型
計 畫 編 號	：NSC 95-2416-H-002-053-
執 行 期 間	：95 年 08 月 01 日至 96 年 07 月 31 日
執 行 單 位	：國立臺灣大學資訊管理學系暨研究所

計畫主持人：李瑞庭

計畫參與人員：博士班研究生-兼任助理：王春笙
碩士班研究生-兼任助理：李梓煜、葉榮忠、陳嘉信、林明志、許家銘、高嘉興

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 96 年 08 月 19 日

行政院國家科學委員會補助專題研究計畫 ☑ 成 果 報 告
☐期中進度報告

（計畫名稱）

跨交易關聯規則演算法之研究

A study on mining inter-transaction association rules

計畫類別：☑ 個別型計畫　　☐ 整合型計畫
計畫編號：NSC 95-2416-H-002-053
執行期間：　95　年　8　月　1　日至　96　年　7　月　31　日

計畫主持人：李瑞庭
共同主持人：
計畫參與人員：王春笙、李梓煜、葉榮忠、陳嘉信、林明志、許家銘、高嘉興

成果報告類型(依經費核定清單規定繳交)：☑精簡報告　☐完整報告

本成果報告包括以下應繳交之附件：
☐赴國外出差或研習心得報告一份
☐赴大陸地區出差或研習心得報告一份
☐出席國際學術會議心得報告及發表之論文各一份
☐國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
☐涉及專利或其他智慧財產權，☐一年☐二年後可公開查詢

執行單位：國立臺灣大學資訊管理學系暨研究所
中　華　民　國　九十五 年 八 月 二十 日

1. Introduction

Association rule mining is an important problem in the data mining field [1, 2, 4, 5, 8, 10, 11, 12, 13, 15, 16, 20, 22, 25, 26]. Traditional association analysis is intra-transactional because it focuses on association relationships among itemsets within a transaction. For example, an intra-transaction association rule might be: "If the stock prices of Microsoft and IBM go up, the price of Intel is likely to go up on the same day." However, intra-transactional approaches cannot capture a rule like: "If the stock prices of Microsoft and IBM go up, the price of Intel is likely to go up two days later." Inter-transaction association rule mining [6, 7, 14, 17, 18, 23, 24] extends the association rules to describe association relationships among itemsets across several transactions.

Many algorithms for mining inter-transaction association rules have been proposed. Lu et al. [17] and Feng et al. [6] applied inter-transaction association rule mining algorithms to the prediction of trends in meteorological and stock market data. Lu et al. [17, 18] proposed the EH-Apriori algorithm, which uses the Apriori algorithm to discover frequent inter-transaction itemsets. To enhance their algorithm's efficiency, the authors used a hashing technique to reduce the number of candidate itemsets of length two. Feng et al. [7] used templates to reduce the number of rules. More recently, Tung et al. [23, 24] developed an algorithm, called FITI (First Intra Then Inter), which discovers frequent intra-transaction itemsets and uses them to generate frequent inter-transaction itemsets.

All the algorithms for mining inter-transaction association rules developed thus far have been based on Apriori-like breadth-first search (BFS) approaches that search for frequent itemsets level by level. At each level, a database must be scanned once to determine the support for each candidate itemset. It has been shown that Apriori-like approaches [19, 22] perform well in finding frequent intra-transaction itemsets when the itemsets are short. However, when mining long frequent itemsets, or using very small support thresholds, the performance of such algorithms often deteriorates dramatically. The reason is that a frequent itemset of length *k* implies the presence of 2^k-2 additional frequent sub itemsets, each of which must be examined. Moreover, since Apriori-like approaches may generate a large number of candidate patterns at each level, they are prone to memory shortage during the mining process. We observe that Apriori-like methods for finding frequent inter-transaction itemsets have the same drawbacks as those for finding frequent intra-transaction itemsets.

Therefore, in this paper, we propose an efficient method for mining a complete set of frequent inter-transaction itemsets (patterns). The method consists of two phases. First, we find all frequent items. For each frequent item found, we construct a dat-list that records the item information required for finding the frequent inter-transaction patterns. Then, we devise a data structure, called an ITP-tree, to store the discovered frequent inter-transaction patterns. In the second phase, we propose an algorithm, called ITP-miner, to efficiently find all frequent inter-transaction patterns in a depth-first search manner. By using the ITP-tree and dat-lists to mine the frequent inter-transaction patterns, the ITP-Miner algorithm requires only one database scan and can localize joining, pruning, and support counting to a small number of dat-lists. Therefore, it is more efficient than the FITI algorithm.

The remainder of this paper is organized as follows. In Section 2, we describe the problem of mining frequent inter-transaction patterns. Section 3 introduces our proposed algorithm, ITP-Miner, for mining frequent inter-transaction patterns. We explain the basic concept of the algorithm and give an example to illustrate how it works. Section 4 describes the performance evaluation. Finally, we present the conclusions in Section 5.

2. Problem Description

In this section, we introduce the notations and describe the problem of mining frequent inter-transaction patterns.

Definition 1. Let I be a set of data items, and N be a set of non-negative integers called domain attributes. A transaction database consists of a set of transactions, where a transaction is in the form of $<t, s>$, $t \in N$ and $s \subseteq I$; t is called a dimensional attribute (or dat), and s is called an itemset.

Table 1. A transaction database.

dat	itemset
1	{a,b}
2	{a,c,d}
3	{a}
4	{a,b,c,d}
5	{a,b,d}
6	{a,d}

The dimensional attribute describes the properties associated with the data items, such as time and location. As it is an ordinal, it can be divided into intervals of equal length. For example, time can be divided into days, weeks, etc. These intervals can be represented by non-negative integers 0, 1, 2, and so on. An itemset is denoted by $\{u_1, u_2, \ldots, u_k\}$, where u_i is an item, $1 \le i \le k$; and items in an itemset are listed in alphabetical order, i.e., we write $\{a,c,d\}$ instead of $\{c,a,d\}$. Table 1 shows a transaction database containing 6 transactions.

Definition 2. An itemset $s=\{u_1, u_2, \ldots, u_k\}$ at the dimensional attribute t is called an *extended itemset* and denoted by $\triangle_t s=\{u_1(t), u_2(t), \ldots, u_k(t)\}$.

Before mining inter-transaction association rules, we need to find the frequent inter-transaction itemsets that span several transactions. Since an inter-transaction itemset can span many intervals, discovering all such itemsets would require a lot of resources, but a user may only be interested in rules that span a certain number of intervals. Therefore, to avoid wasting resources by mining unwanted rules, we introduce a parameter called *maxspan*. When mining for inter-transaction association rules, we only mine rules whose span is equal to or less than the *maxspan* intervals.

Definition 3. Let $\triangle_{t_0}\alpha_0$, $\triangle_{t_1}\alpha_1$, …, $\triangle_{t_n}\alpha_n$ be extended itemsets, where $t_0 < t_1 < \ldots < t_n$, and t_n-t_0 is not greater than a user-specified maximum span threshold *maxspan*. Then, let $\alpha = \triangle_0\alpha_0 \cup \triangle_{t_1-t_0}\alpha_1 \cup \ldots \cup \triangle_{t_n-t_0}\alpha_n$, where $\cup$ is an operator that combines multiple extended itemsets into single unit, α is called an inter-transaction itemset (or a pattern) and t_0 is called the reference point. An item in the inter-transaction itemset is called an *extended item*. Since α takes t_0 as the reference point, we say that α starts from t_0. Note that the reference point of an inter-transaction itemset is defined as the smallest dimensional attribute of the extended itemsets in the inter-transaction itemset.

For example, let us consider the database as shown in Table 1, where {*a*(1), *b*(1)} is the extended itemset for the first transaction. Let *maxspan* = 1. The first two transactions form an inter-transaction itemset, {*a*(0), *b*(0), *a*(1), *c*(1), *d*(1)}, where we take the dimensional attribute of the first transaction as the reference point.

Definition 4. Let $u(i)$ and $v(j)$ be two extended items. If ($i = j$ and $u = v$), we can say that $u(i) = v(j)$. Also, if ($i = j$ and $u < v$) or ($i < j$), we say that $u(i) < v(j)$.

Definition 5. Let $\alpha = \{u_0(0), u_1(i_1), u_2(i_2), \ldots, u_n(i_n)\}$ and $\beta = \{v_0(0), v_1(j_1), v_2(j_2), \ldots, v_n(j_n)\}$ be two patterns, where $n \geq 1$. We say that $\alpha = \beta$ if $u_i(i_i) = v_i(j_i)$ for $0 \leq i \leq n$, where $i_0 = j_0 = 0$. We also say that $\alpha < \beta$, if 1) $u_0(0) < v_0(0)$; or 2) there exists k (≥ 0) such that $u_i(i_i) = v_i(j_i)$ for $0 \leq i \leq k < n$, and $u_{k+1}(i_{k+1}) < v_{k+1}(j_{k+1})$.

Definition 6. Let $\alpha = \{u_0(0), u_1(i_1), u_2(i_2), \ldots, u_n(i_n)\}$ and $\beta = \{v_0(0), v_1(j_1), v_2(j_2), \ldots, v_m(j_m)\}$ be two patterns, where $1 \leq n \leq m$. α is a *subset* of β if we find n extended items $v_{k1}(j_{k1})$, $v_{k2}(j_{k2})$, …, $v_{kn}(j_{kn})$ in β such that $u_0(0) = v_0(0)$, $u_1(i_1) = v_{k1}(j_{k1})$, $u_2(i_2) = v_{k2}(j_{k2})$, …, and $u_n(i_n) = v_{kn}(j_{kn})$. We can also say that β contains α.

For example, both {*b*(0), *a*(1), *c*(3)} and {*a*(0), *d*(1)} are subsets of {*a*(0), *b*(0), *a*(1), *d*(1), *b*(3), *c*(3)}.

Definition 7. In a transaction database *D*, the number of transactions is denoted by $|D|$. Let α be a pattern, and T_α be all inter-transaction itemsets in *D*, where every inter-transaction itemset in T_α contains α. The *count* of α, *count*(α), is denoted by $|T_\alpha|$ and the *support* of α, *sup*(α), is denoted by *count*(α)/$|D|$. If *sup*(α) is not less than the user-specified minimum support threshold *minsup*, α is called a *frequent inter-transaction pattern* (or *frequent pattern* for short).

Definition 8. The number of extended items in a pattern is called the *length* of the pattern. A pattern of length *k* is called a *k*-pattern.

For example, the length of {*a*(0), *a*(1), *d*(1), *b*(3), *c*(3)} is equal to 5.

Definition 9. An *inter-transaction association rule* is written in the form of $\alpha \rightarrow \beta$, where both α and $\alpha \cup \beta$ are frequent patterns, $\alpha \cap \beta = \phi$, *conf*($\alpha \rightarrow \beta$) is not less than the user-specified minimum confidence, and the confidence of the rule is defined as *sup*($\alpha \cup \beta$)/*sup*(α).

Suppose that *minsup*=50% and *maxspan*=1. Let us consider the database *D* shown in Table 1, where the database contains six transactions (i.e., $|D|$=6). To compute the support of {*a*(0), *b*(0), *d*(1)}, we first form an inter-transaction itemset {*a*(0), *b*(0), *a*(1), *c*(1), *d*(1)} by using the first two transactions and find that it contains {*a*(0), *b*(0), *d*(1)}. Thus, count({*a*(0), *b*(0), *d*(1)})=1. Next, we form an inter-transaction itemset {*a*(0), *b*(0), *c*(0), *d*(0), *a*(1), *b*(1), *d*(1)} by using the fourth and fifth transactions, and find that it contains {*a*(0), *b*(0), *d*(1)}. Thus, the count of {*a*(0), *b*(0), *d*(1)} is increased by one. Finally, we form an inter-transaction itemset {*a*(0), *b*(0), *d*(0), *a*(1), *d*(1)} by using the last two transactions and find that it contains {*a*(0), *b*(0), *d*(1)}. Thus, the count of {*a*(0), *b*(0), *d*(1)} is increased by one. That is, *count*({*a*(0), *b*(0), *d*(1)})=3, and *sup*({*a*(0), *b*(0), *d*(1)})=3/6=0.5. Consequently, {*a*(0), *b*(0), *d*(1)} is a frequent pattern. Similarly, we find that *sup*({*a*(0), *b*(0)})= 0.5. That is, {*a*(0), *b*(0)} is also a frequent pattern. Therefore, we have an inter-transaction association rule {*a*(0), *b*(0)}$\rightarrow${*d*(1)} with the confidence = 0.5/0.5 = 100%.

The objective of mining frequent inter-transaction patterns is to find a complete set of frequent patterns in a transaction database with respect to user-specified *minsup* and *maxspan* thresholds.

3. The ITP-Miner Algorithm

We now introduce the ITP-Miner algorithm. In Section 3.1 we describe a data structure, called a dat-list, and a search tree, called an ITP-tree. We then explain the main concept of our proposed algorithm in Section 3.2, and describe it in detail in Section 3.3. Finally, we give an example to demonstrate how the algorithm works in Section 3.4.

3.1 The dat-list and ITP-tree

We have devised a data structure, called the dimensional attribute list (dat-list), to store the dimensional attributes of a frequent pattern during the mining process. Let $\alpha<t_1, t_2, \ldots, t_n>$ be a dat-list, where α is a pattern and $<t_1, t_2, \ldots, t_n>$ is a list of dats in which the inter-transaction itemset starting from t_i contains α, $1 \leq i \leq n$. For example, {*a*(0), *d*(1)}<1,3,4,5> is a dat-list containing 4 dats, 1, 3, 4, and 5; i.e., the inter-transaction itemsets starting from the first, third, fourth, and fifth transactions in the database shown in Table 1 contain the pattern {*a*(0), *d*(1)}. Table 2 shows all dat-lists of the frequent patterns found in the database shown in Table 1, where *minsup* = 50% and *maxspan* = 1. Note that the count of a pattern is equal to the number of dats contained in its dat-list. For example, the support counts of {*a*(0), *d*(1)}<1,3,4,5> and {*a*(0), *b*(0), *a*(1), *d*(1)}<1,4,5> are 4 and 3, respectively.

Next, we introduce a data structure called an ITP-tree, which stores the dat-lists generated during the mining process.

Definition 10. An ITP-tree is a search tree denoted by *T*. A node in *T* represents a dat-list, $\alpha<t_1,t_2,...,t_n>$, where $n\geq 1$, α is a frequent pattern, and the extended items in α are listed in increasing order. The parent of node $\alpha<t_1,t_2,...,t_n>$ is the dat-list $\beta<t'_1,t'_2,...,t'_m>$, where $m\geq n$, and β is the pattern obtained by removing the last extended item from α. The root of *T* is a null pattern, i.e., {}. Siblings with the same parent are sorted in increasing order.

Table 2. The dat-lists.

Length	Dat-list
1	{a(0)}<1,2,3,4,5,6>, {b(0)}<1,4,5>, {d(0)}<2,4,5,6>
2	{a(0),b(0)}<1,4,5>, {a(0),d(0)}<2,4,5,6>, {a(0),a(1)}<1,2,3,4,5>, {a(0),d(1)}<1,3,4,5>, {b(0),a(1)}<1,4,5>, {b(0),d(1)}<1,4,5>, {d(0),a(1)}<2,4,5>
3	{a(0),b(0),a(1)}<1,4,5>, {a(0),b(0),d(1)}<1,4,5>, {a(0),d(0),a(1)}<2,4,5>, {a(0),a(1),d(1)}<1,3,4,5>, {b(0),a(1),d(1)}<1,4,5>
4	{a(0),b(0),a(1),d(1)}<1,4,5>

Consider the complete ITP-tree shown in Fig. 1, where the database contains only two items, *a* and *b*; the *maxspan* is equal to 1; and the list of dats associated with each dat-list is omitted. Note that all possible inter-transaction patterns of the ITP-tree in Fig. 1 can be enumerated in a depth-first search manner.

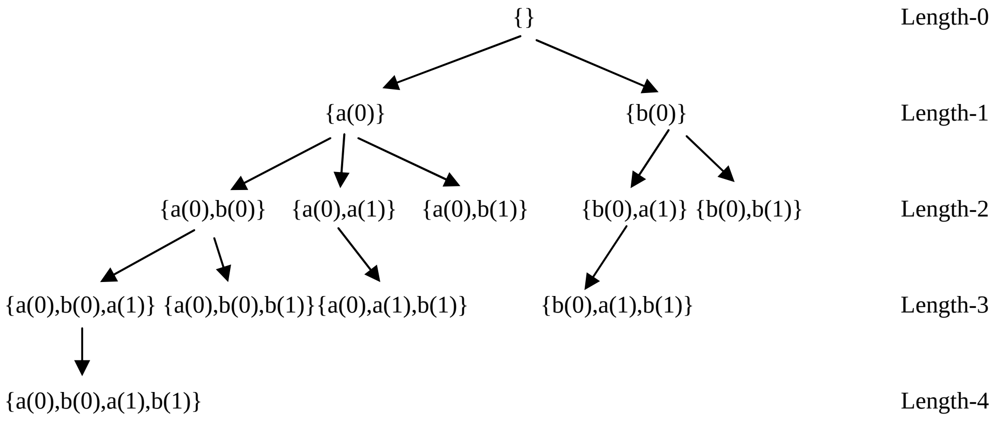


Fig. 1. An ITP-tree.

3.2 Main concept

First, we consider the generation of a candidate 2-pattern by joining two frequent 1-patterns.

Definition 11. If both α and β are frequent 1-patterns, α is *joinable* to β.

Let t_1 and t_2 be the dats in the dat-lists of α={*u*(0)} and β={*v*(0)}, respectively. If ($u < v$ and $0 \leq t_2-t_1 \leq$ *maxspan*) or ($u \geq v$ and $0 < t_2-t_1 \leq$ *maxspan*), then: 1) we generate a candidate 2-pattern $\{u(0),v(t_2-t_1)\}$ if it has not been generated already; 2) we add t_1 to the candidate's dat-list; and 3) we increase the count of $\{u(0),v(t_2-t_1)\}$ by one. For each candidate 2-pattern generated, we check that its support is not less than the *minsup*. If this is the case, it is a frequent 2-pattern and we store its dat-list in the ITP-tree. Thus, for a frequent 2-pattern $\{u(0),v(w)\}$, $0 \leq w \leq$ *maxspan*, we store its dat-list as a child of α's dat-list in the ITP-tree.

Fig. 2 illustrates how we find a frequent 2-pattern by joining the dat-lists of two frequent 1-patterns {*a*(0)}<1,2,3,4,5,6> and {*b*(0)}<2,4,5,6>, where *minsup*=50% and *maxspan*=1. Here, we find two frequent 2-patterns, {*a*(0),*b*(0)} and {*a*(0),*b*(1)}. In Fig 2(a), the dats used to generate pattern {*a*(0),*b*(0)} are connected by black arrows, while the dats used to generate pattern {*a*(0),*b*(1)} are connected by dashed arrows. Fig. 2(b) shows the ITP-tree formed by the dat-lists of the patterns found. Note that there are six dats in {*a*(0)}<1,2,3,4,5,6>; however, there are only four in {*a*(0),*b*(0)}<2,4,5,6> and {*a*(0),*b*(1)}<1,3,4,5>.

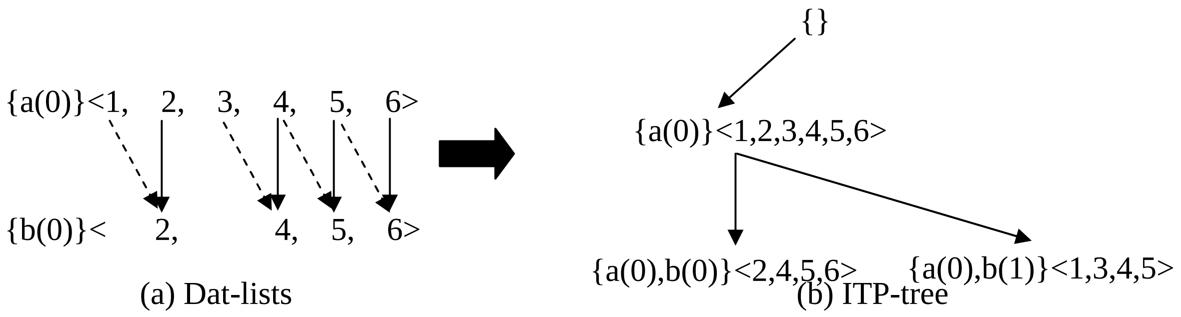


Fig. 2. Generating 2-patterns by joining two 1-patterns.

Definition 12. Let $\alpha = \{u_0(0), u_1(i_1), \ldots, u_{k-1}(i_{k-1})\}$ and $\beta = \{v_0(0), v_1(j_1), \ldots, v_{k-1}(j_{k-1})\}$ be two frequent *k*-patterns, $k > 1$. α is *joinable* to β if the first *k*-1 extended items of α are equal to those of β, and the last extended item of α is less than that of β. The joined pattern is obtained by appending the last extended item of β to α; that is, the joined pattern is $\{u_0(0), u_1(i_1), \ldots, u_{k-1}(i_{k-1}), v_{k-1}(j_{k-1})\}$.

We now consider how to find frequent (k+1)-patterns by joining two joinable k-patterns. Assume there are two dat-lists of 2-patterns, {a(0), b(0)}<1,4,5> and {a(0), a(1)} <1,2,3,4,5>. Note that the first extended item of {a(0), b(0)} is equal to that of {a(0), a(1)}. To join both 2-patterns, we match the dats in {a(0), b(0)}<1,4,5> with the dats in {a(0),a(1)}<1,2,3,4,5>. If a dat in the former is equal to a dat in the latter, the count of the joined pattern ({a(0),b(0),a(1)}) is increased by one. By matching every dat in {a(0), b(0)}<1,4,5> with its counterpart in {a(0), a(1)}<1,2,3,4,5>, we find that the count of {a(0),b(0),a(1)} is equal to 3. Thus, it is a frequent 3-pattern. We store {a(0),b(0),a(1)}<1,4,5> in the ITP-tree as shown in Fig. 3, where <1,4,5> are the matched dats between <1,4,5> and <1,2,3,4,5>.

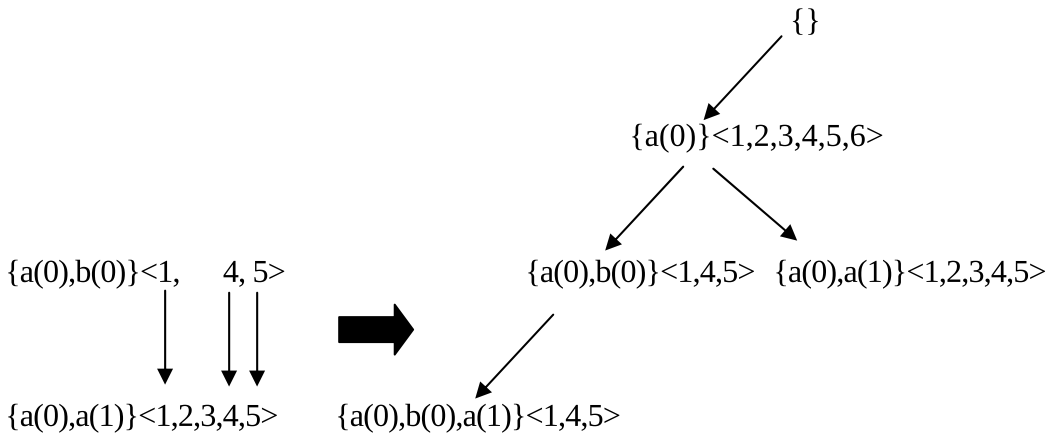


Fig. 3. Generating a 3-pattern by joining two frequent 2-patterns.

The ITP-miner algorithm generates the frequent patterns in a depth-first search manner. We use the frequent patterns obtained at this level (frequent k-pattern**)** to generate the frequent patterns at the next level (frequent (k+1)-pattern). To do so, we first define the *joinable* and *extended* groups for a frequent k-pattern. A frequent k-pattern, α, only needs to join the patterns in its joinable group to generate the frequent (k+1)-pattern extended from α. Generating the frequent (k+1)-pattern in this manner can localize joining, pruning, and support counting to a small number of dat-lists.

Definition 13. Let α be a frequent k-pattern, where $k \geq 1$. The *joinable group* of α is $J(\alpha)$ = {α_1, α_2, ..., α_n}, where α is joinable to α_i, which is a frequent k-pattern, $1 \leq i \leq n$. Also, the *extended group* of α is $E(\alpha)$ = {β_1, β_2, ..., β_m}, where every β_j is a frequent (k+1)-pattern extended from α, $1 \leq j \leq m$. Since β_j is extended from α, its dat-list will be a child of α's dat-list in the ITP-tree.

Lemma 1. Let α be a frequent k-pattern; and β be a frequent (k+1)-pattern that is extended from α, where $k \geq 1$. Then, we have: 1) $E(\alpha)$={θ|θ=(α join γ), θ is frequent, and $\gamma \in J(\alpha)$}; and 2) $J(\beta)$={θ|θ is greater than β, and $\theta \in E(\alpha)$}.

We can use Lemma 1 to generate frequent patterns and store their dat-lists in the ITP-tree, where the frequent patterns are generated in a depth-first search manner. The generation of frequent patterns consists of two steps. Let α be a frequent k-pattern. First, α is joined to each γ in $J(\alpha)$, after which we can find all the frequent (k+1)-pattern patterns extended from α, referred to as $E(\alpha)$. Next, for each frequent (k+1)-pattern β in $E(\alpha)$, $J(\beta)$ is the set containing the patterns in $E(\alpha)$ that are greater than β. We can therefore join β to every γ in $J(\beta)$ to find all the frequent (k+2)-patterns extended from β, referred to as $E(\beta)$. Let $\alpha = \beta$ and k=k+1. We perform the second step recursively in a depth-first search manner until no more frequent patterns can be generated. This is the main concept of our proposed method.

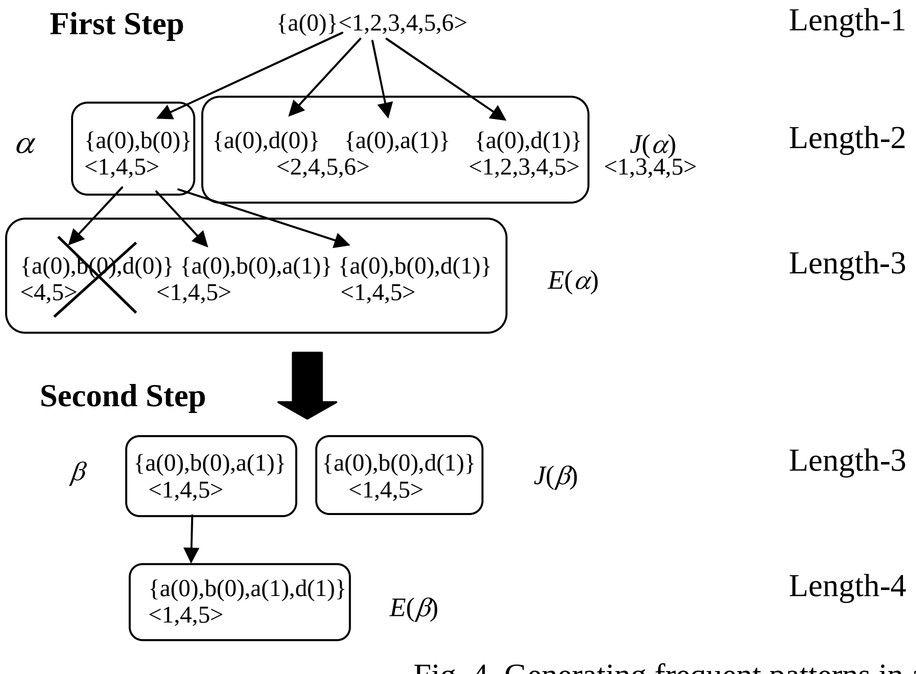


Fig. 4. Generating frequent patterns in an ITP-tree.

Let us consider the example shown in Fig 4, where $\alpha = \{a(0),b(0)\}$ and $\beta = \{a(0),b(0),a(1)\}$. From the figure, we know that $J(\alpha) = \{\{a(0),d(0)\}, \{a(0),a(1)\}, \{a(0),d(1)\}\}$; $E(\alpha) = \{\{a(0),b(0),a(1)\}, \{a(0),b(0),d(1)\}\}$; $J(\beta) = \{\{a(0),b(0),d(1)\}\}$; and $E(\beta) = \{\{a(0),b(0),a(1),d(1)\}\}$. The figure shows that $E(\alpha)$ can be obtained by joining α with every pattern in $J(\alpha)$, while $E(\beta)$ can be obtained by joining β with every pattern in $J(\beta)$.

3.3 The ITP-Miner algorithm

The ITP-Miner algorithm is comprised of three functions: Join2, DFS, and JoinK. The algorithm and its functions are shown in Figs. 5, 6, 7, and 8, respectively. The Join2 function is used to find frequent 2-patterns by joining two frequent 1-patterns, while the JoinK function is used to find frequent (k+1)-pattern by joining two frequent k-patterns.

```
Algorithm: ITP-Miner
Input: transaction database $D$, minimum support $minsup$, maximum span $maxspan$.
Output: all frequent inter-transaction patterns $FP$.
Method:
1   Scan $D$ to find all frequent 1-patterns and their dat-lists. Construct a hash table, $H2$, and insert the dat-lists
        into the ITP-tree $T$;
2   for each frequent 1-pattern $\alpha$ do
3       Insert $\alpha$ into $FP$;
4       for each frequent 1-pattern $\gamma$ in $J(\alpha)$ do call Join2($\alpha$, $\gamma$, $T$, $minsup$, $maxspan$, $|D|$) to get $E(\alpha)$;
5       for each frequent 2-pattern $\beta \in E(\alpha)$ do call DFS($\beta$, $FP$, $T$, $minsup$, $maxspan$, $|D|$);
6   end for
7   Output $FP$.
```

Fig. 5. The ITP-Miner algorithm

In Step 1 of Fig.5, the database is scanned to find all frequent 1-patterns and their dat-lists, after which a hash table, $H2$, is constructed and the dat-lists are inserted into the ITP-tree, T. In Step 4, two frequent 1-patterns, α and γ, are joined to generate frequent 2-patterns and obtain $E(\alpha)$. Note that the joinable group of a frequent 1-pattern is equal to the set containing all frequent 1-patterns. In Step 5, for each frequent 2-pattern β, the DFS function is applied recursively in a depth-first search manner to find all frequent inter-transaction patterns.

When running the ITP-Miner algorithm, we use the following pruning strategies to reduce the search space and speed up the mining process.

```
Function: Join2($\alpha$, $\gamma$, $T$, $minsup$, $maxspan$, $|D|$)
Input: two frequent 1-patterns $\alpha = \{u(0)\}$ and $\gamma = \{v(0)\}$, the ITP-tree $T$, minimum support $minsup$, maximum
        span $maxspan$, and the number of transactions in the database $|D|$.
Output: the ITP-tree $T$.
1   if there exists $w$, $0 \le w \le maxspan$, such that $H2[((u * (maxspan+1) + w) * N1 + v) \bmod Hashsize] \ge minsup$
    $* |D|$ then
2       for each dat $t_\alpha \in$ $\alpha$'s dat-list do
3           for each dat $t_\gamma \in$ $\gamma$'s dat-list do
4               if ($u < v$ and $0 \le t_\gamma - t_\alpha \le maxspan$) or ($u \ge v$ and $0 < t_\gamma - t_\alpha \le maxspan$) then
5                   Generate a candidate 2-pattern $\{u(0),v(t_\gamma - t_\alpha)\}$ if it has not been generated already;
6                   Add $t_\alpha$ to its dat-list;
7               end if
8           end for
9       end for
10      for each candidate 2-pattern generated do
11          if it is frequent then
12              Insert its dat-list into $T$ to be a child of $\alpha$'s dat-list;
13          end if
14      end for
15  end if;
```

Fig. 6. The Join2 Function

```
Function: DFS($\beta$, $FP$, $T$, $minsup$, $maxspan$, $|D|$)
Input: a frequent pattern $\beta$, all frequent inter-transaction patterns $FP$, the ITP-tree $T$, minimum support $minsup$,
        maximum span $maxspan$, and the number of transactions in the database $|D|$.
Output: the ITP-tree $T$.
1   Insert $\beta$ into $FP$;
2   for each pattern $\gamma \in J(\beta)$, call JoinK($\beta$, $\gamma$, $T$, $minsup$, $maxspan$, $|D|$) to get $E(\beta)$;
3   for each pattern $\theta \in E(\beta)$, call DFS($\theta$, $FP$, $T$, $minsup$, $maxspan$, $|D|$);
4   Remove $\beta$'s dat-list from $T$;
```

Fig. 7. The DFS Function.

(1) **Infrequent 2-pattern pruning**. In Step 1 of the algorithm (Fig. 5), if we generate n 1-patterns in Step 1, we will perform n^2 join operations in Step 4. Since many 2-patterns are infrequent, it is wasteful to perform

so many operations. To resolve the problem, we use a hashing approach [24] to check if a pair of 1-patterns can generate a frequent 2-pattern before joining them. In Step 1, therefore, we construct a hash table, *H2*, when searching for 1-patterns and dat-lists. Every bucket of *H2* aggregates the counts of all length-2 candidates hashed to it. Given a candidate 2-pattern generated by items *u*' and *v*' with dats $t_{u'}$ and $t_{v'}$, respectively, its hash value *g* is computed by $g=((u' * (maxspan+1) + (t_{v'} - t_{u'})) * N1 + v') \bmod Hashsize$, where $0 \leq t_{v'} - t_{u'} \leq maxspan$, *N1* is the number of distinct items in the database, and *Hashsize* is the size of the hash table. The count of *H2*[*g*] is then incremented by one. In Step 1 of Fig. 6, before joining two 1-patterns, i.e., $\alpha=\{u(0)\}$ and $\gamma=\{v(0)\}$, we compute all possible hash values for both patterns by using $h = ((u * (maxspan+1) + w) * N1 + v) \bmod Hashsize$, where $0 \leq w \leq maxspan$. If *H2*[*h*] is less than *minsup**|*D*| for each possible hash value, *h*, we do not need to join α and γ to generate a 2-pattern.

(2) **Infrequent *k*-pattern pruning**. Before joining two *k*-patterns when $k > 1$, we use the hash table *H2* to filter out the join operations. In Step 1 of Fig. 8, before joining two *k*-patterns, $\beta=\{u_0(0),u_1(i_1),...,u_{k-1}(i_{k-1})\}$ and $\gamma=\{v_0(0),v_1(j_1),...,v_{k-1}(j_{k-1})\}$, we know that β joining γ will yield $\{u_0(0),u_1(i_1),...,u_{k-1}(i_{k-1}),v_{k-1}(j_{k-1})\}$. Since $\{u_{k-1}(i_{k-1}),v_{k-1}(j_{k-1})\}$ is a subset of the joined pattern, it can be transformed into a 2-pattern, $\{u_{k-1}(0),v_{k-1}(j_{k-1}-i_{k-1})\}$, by taking i_{k-1} as the reference point. Thus, we can compute its hash value using $h = ((u_{k-1} * (maxspan+1) + (j_{k-1}-i_{k-1})) * N1 + v_{k-1}) \bmod Hashsize$. If *H2*[*h*] is less than *minsup**|*D*|, we do not need to join β and γ to generate a (*k*+1)-pattern.

(3) **Unmatched dat pruning**. In the ITP-Miner algorithm, we join two joinable *k*-patterns to generate frequent (*k*+1)-patterns and construct a dat-list for each of the latter. In Step 6 of Fig. 6 and Step 4 of Fig. 8, when we construct the dat-list for a pattern, we only add the matched dats of both *k*-patterns to the dat-list of the newly generated (*k*+1)-pattern.

Function: JoinK(β, γ, *T*, *minsup*, *maxspan*, |*D*|)
Input: two frequent *k*-patterns $\beta=\{u_0(0),u_1(i_1),...,u_{k-1}(i_{k-1})\}$ and $\gamma=\{v_0(0),v_1(j_1),...,v_{k-1}(j_{k-1})\}$, the ITP-tree *T*, minimum support *minsup*, maximum span *maxspan*, and the number of transactions in the database |*D*|.
Output: the ITP-tree *T*.

```
1  if $H2[((u_{k-1} * (maxspan+1) + (j_{k-1} - i_{k-1})) * N1 + v_{k-1}) \bmod Hashsize] \geq minsup * |D|$, then
2       Let $\theta = \{u_0(0),u_1(i_1),...,u_{k-1}(i_{k-1}),v_{k-1}(j_{k-1})\}$;
3       for each dat $x \in \beta$'s dat-list do
4           Check if any dat $y \in \gamma$'s dat-list such that $y = x$. If yes, increase $\theta$'s count by one and add $x$ to $\theta$'s
                dat-list;
5       end for
6       if $\theta$ is frequent then
7           Insert its dat-list into $T$ to be a child of $\beta$'s dat-list;
8       end if
9  end if;
```

Fig. 8. The JoinK Function.

3.4 An example

Let us consider the example as shown in Table 1, where *minsup*=50% and *maxspan*=1. The frequent inter-transaction patterns and their dat-lists generated during the mining processes are shown in Fig. 9. During the database is scanned, we compute the count for each 1-pattern. The counts of patterns {*a*(0)}, {*b*(0)}, {*c*(0)}, and {*d*(0)}, are equal to 6, 3, 2, and 4, respectively. Since the *minsup* is 50%, a pattern is frequent if its count is not less than 3. Consequently, {*a*(0)}<1,2,3,4,5,6>, {*b*(0)}<1,4,5>, and {*d*(0)}<2,4,5,6> are inserted into the ITP-tree as shown in Fig. 9. Note that, since we have three frequent 1-patterns: {*a*(0)}, {*b*(0)}, and {*d*(0)}, there are three joinable groups: *J*({*a*(0)})={{*a*(0)},{*b*(0)},{*d*(0)}}, *J*({*b*(0)})={{*a*(0)},{*b*(0)}, {*d*(0)}}, and *J*({*d*(0)})= {{*a*(0)},{*b*(0)},{*d*(0)}}.

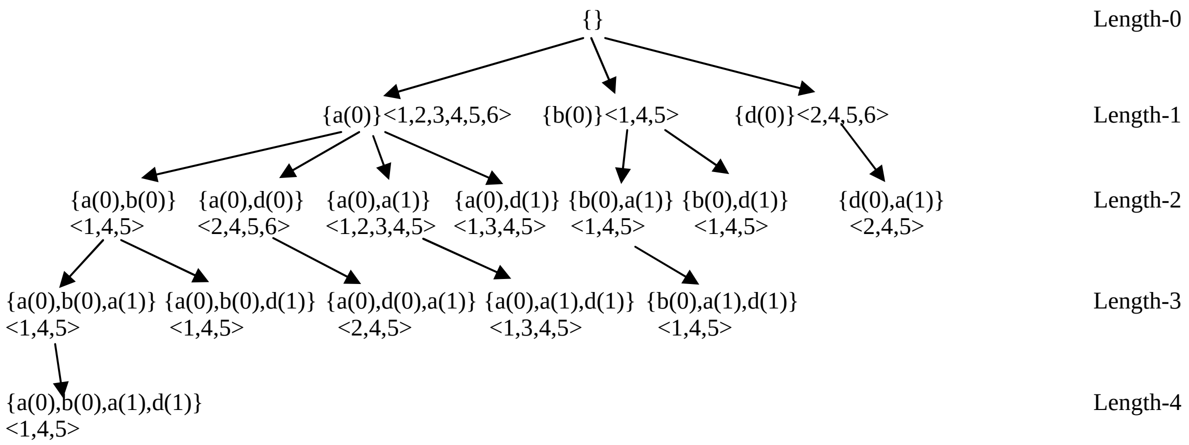


Fig. 9. The ITP-tree for the database shown in Table 1.

We now describe how to join {*a*(0)} to every pattern in *J*({*a*(0)}) to find frequent 2-patterns. The dat-lists of {*a*(0)} and {*b*(0)} share three dats, <1,4,5>; therefore, by joining them we obtain a frequent 2-pattern {*a*(0),*b*(0)} with a list of dats, <1,4,5>. Similarly, by joining {*a*(0)}<1,2,3,4,5,6> and {*d*(0)}<2,4,5,6>, we obtain {*a*(0),*d*(0)}<2,4,5,6>. Moreover, {*a*(0),*a*(1)}<1,2,3,4,5> can be generated, since {*a*(0)}'s dat-list contains five dats

<1,2,3,4,5>, each of which is one less than {*a*(0)}'s dats <2,3,4,5,6>; {*a*(0),*d*(1)}<1,3,4,5> can be generated in the same manner. Consequently, *E*({*a*(0)})={{*a*(0),*b*(0)}, {*a*(0),*d*(0)}, {*a*(0),*a*(1)}, {*a*(0),*d*(1)}}, as shown in Fig. 9. Note that according to the "unmatched dat pruning" strategy, we only add the matched dats of both 1-patterns to the dat-list of the newly generated 2-pattern.

Since we have *E*({*a*(0)}), we can join every 2-pattern β in *E*({*a*(0)}) to every 2-pattern in *J*(β) to get frequent 3-patterns. Consider the pattern {*a*(0),*b*(0)}. We know that the patterns joinable to {*a*(0),*b*(0)} are those in *E*({*a*(0)}) that are larger than {*a*(0),*b*(0)}. In other words, *J*({*a*(0),*b*(0)})={{*a*(0),*d*(0)},{*a*(0),*a*(1)},{*a*(0),*d*(1)}}. In this stage, the join operation is performed by matching the dat-lists of two joinable patterns, so we join {*a*(0),*b*(0)}<1,4,5> and {*a*(0),*a*(1)}<1,2,3,4,5> to get {*a*(0),*b*(0),*a*(1)}<1,4,5>; and join {*a*(0),*b*(0)}<1,4,5> and {*a*(0),*d*(1)}<1,3,4,5> to get {*a*(0),*b*(0),*d*(1)}}<1,4,5>. Therefore we get *E*({*a*(0),*b*(0)}) = {{*a*(0),*b*(0),*a*(1)}, {*a*(0),*b*(0),*d*(1)}}, as shown in Fig. 9. After obtaining *E*({*a*(0),*b*(0)}), we join every 3-pattern β in *E*({*a*(0),*b*(0)}) and every 3-pattern in *J*(β) to find frequent 4-patterns. According to the "infrequent k-pattern pruning" strategy, for joining {*a*(0),*b*(0),*a*(1)} and {*a*(0),*b*(0),*d*(1)}, the last extended item of the former and that of the latter can be combined to form a 2-pattern, {*a*(1),*d*(1)}, which is a subset of the joined pattern, {*a*(0),*b*(0),*a*(1),*d*(1)}. Pattern {*a*(1),*d*(1)} can be transformed into {*a*(0),*d*(0)} by taking dat=1 as the reference point. We join {*a*(0),*b*(0),*a*(1)}<1,4,5> and {*a*(0),*b*(0),*d*(1)}<1,4,5> to find frequent 4-patterns, and obtain {*a*(0),*b*(0),*a*(1),*d*(1)}<1,4,5>, as shown in Fig. 9. The other two 3-patterns, {*a*(0),*d*(0),*a*(1)} and {*a*(0),*a*(1),*d*(1)}, can be obtained in the same way.

A similar procedure can be applied to the nodes containing {*b*(0)} and {*d*(0)} by calling the DFS function recursively. Fig. 9 shows the complete set of frequent patterns and their dat-lists. Starting from the node containing {*b*(0)}, the patterns found include {*b*(0)}, {*b*(0),*a*(1)}, {*b*(0),*d*(1)}, and {*b*(0),*a*(1),*d*(1)}. Meanwhile, starting from the node containing {*d*(0)}, the patterns found include {*d*(0)}, and {*d*(0),*a*(1)}. The mining process is terminated once the node containing {*d*(0),*a*(1)} has been processed, since no more frequent patterns can be generated.

4. Performance Evaluation

We evaluated the performance of the ITP-Miner algorithm on two synthetic datasets. The datasets were generated with different parameters synthetically. We compared the ITP-Miner algorithm with the FITI algorithm [24]. All experiments were performed on an IBM Compatible PC with an Intel Pentium IV CPU 2.8GHz, 1G-byte main memory running on Microsoft Windows XP. Both algorithms were implemented using Microsoft Visual C++ 6.0.

4.1 Generation of synthetic data

We use the same method as that in [24] to generate synthetic transactions. The process involves two steps: first, we generate potential frequent itemsets, after which we generate transactions in the database from those itemsets. The parameters used to generate synthetic data are shown in Table 3.

Table 3. Parameters.

Parameter	Meaning
|*D*|	Total number of transactions
A	Average length of the transactions
F	Number of potential frequent inter-transaction itemsets
MaxT	Maximum length of the transactions
L	Length of each potential frequent inter-transaction itemset
MaxI	Maximum length of the potential frequent inter-transaction itemsets
S	Number of distinct items
M	Maxspan

Table 4. Parameter settings for the two synthetic datasets.

Parameter	Dataset1	Dataset2
|*D*|	10,000	1,000
A	5	100
F	1,000	1,000
MaxT	10	200
L	5	6
MaxI	10	10
S	500	500
M	3	5

The potential frequent inter-transaction itemsets, L_{item}, may span several transactions. The length of each itemset follows a Poisson distribution whose mean is equal to *L*, and the maximum length of an itemset is *MaxI*. Every itemset in L_{item} is associated with a relative distance ranging from 0 to *M* (*maxspan*). Hence, we can generate |*D*| transactions from L_{item}. The length of each transaction is selected from a Poisson distribution whose mean is equal to *A*, and the maximum length of any transaction is *MaxT*.

We generated three synthetic datasets using the parameters shown in Table 4, the first two are similar to the datasets used in [24]. The first dataset, Dataset1, had 10,000 transactions, with an average of 5 items per transaction. Dataset2, on the other hand, had only 1,000 transactions, but each one contained 100 items on average.

4.2 Experiments on synthetic data

In this section, we compare the ITP-Miner algorithm with the FITI algorithm by varying one parameter, while maintaining the other parameters at the default values shown in Table 4.

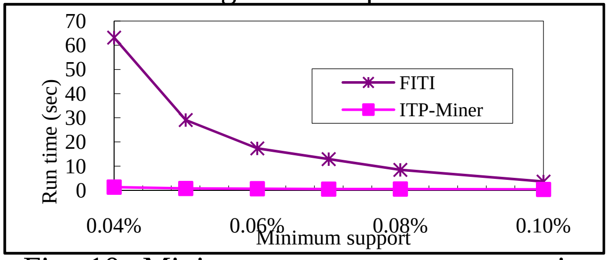


Fig. 10. Minimum support versus run time, Dataset1 with maxspan=3.

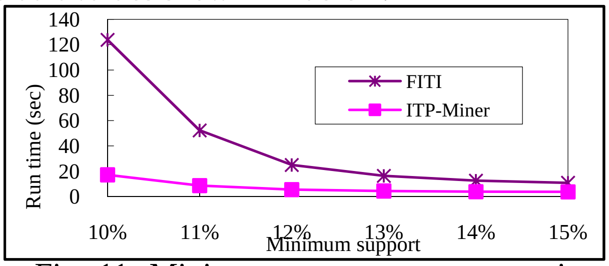
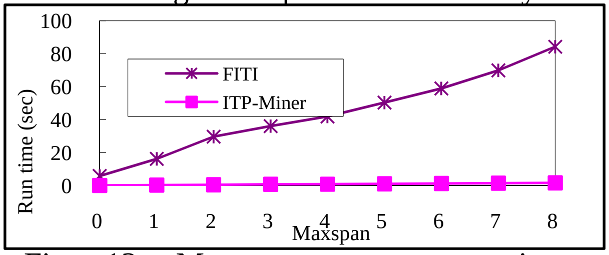


Fig. 11. Minimum support versus run time, Dataset2 with maxspan=5.

Fig. 10 shows the minimum support versus the run time for Dataset1. The minimum support varies between 0.04% and 0.1%, and the ITP-Miner algorithm runs 10-50 times faster than the FITI algorithm. Fig.11 illustrates the minimum support versus the run time for Dataset2. In this case, the support varies between 10% and 15%, and the ITP-Miner algorithm runs 3-8 times faster than the FITI algorithm. The ITP-Miner algorithm outperforms the FITI algorithm because the latter generates a huge number of candidates when the support is small. To count the support of candidates, the FITI uses the Apriori algorithm to count the support of intra-transaction candidates by repeatedly scanning the database. Whenever the FITI algorithm reads a new transaction from the FIT tables [24], each inter-transaction candidate must be matched with *maxspan*+1 consecutive transactions. Moreover, support counting of the FITI algorithm is quite time-consuming because it needs to perform subset matching. In contrast, the ITP-Miner algorithm counts the supports by joining the dat-lists of joinable patterns, and only needs to scan the database once; thus, it is more efficient than the FITI algorithm.

Next, we illustrate the impact of the *maxspan* on the performance of the algorithms for three datasets in Figs. 12 and 13. As the *maxspan* increases, more candidates and patterns are generated. Thus, the run-time of both algorithms increases. We observe that the run-time of the ITP-Miner algorithm increases slowly as the *maxspan* increases. For the FITI algorithm, as the *maxspan* increases, the range of the FIT tables [24] that must be scanned to count the candidates' supports increases. In addition, the number of generated candidates increases rapidly in a breadth-first search manner. Hence, the time required to count the candidates' supports increases. In contrast, by joining patterns in the ITP-tree, the ITP-Miner algorithm uses a depth-first search manner to avoid having to scan the database and count the support for a large number of candidates. Thus, the ITP-Miner algorithm performs efficiently when the *maxspan* increases.



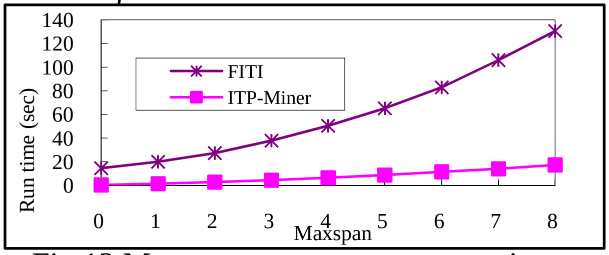


Fig. 12. Maxspan versus run time, Dataset1 with minsup=0.05%.

Fig.13.Maxspan versus run time, Dataset2 with minsup=11%.

In summary, since the ITP-Miner algorithm employs the ITP-tree and dat-lists to mine frequent patterns, it only requires one database scan and can localize candidate joining, pruning, and support counting to joinable patterns. As the "infrequent 2-pattern pruning" and "infrequent *k*-pattern pruning" strategies are used to check if both patterns are joinable, the ITP-Miner algorithm avoids many unnecessary join operations. It outperforms the FITI algorithm by one order of magnitude and requires less main memory storage.

5. Conclusions and Future Work

This work has been published in Information Sciences Journal [27]. In this study, we have proposed an efficient method for mining all frequent inter-transaction patterns. The method consists of two phases. First, we design a data structure, called a dat-list, to record the dimensional attribute information of a frequent item in a database. Then, we devise a data structure, called an ITP-tree, to store the frequent patterns found. Second, we propose an efficient algorithm, called ITP-Miner, to mine the frequent patterns in a depth-first search manner. By using the ITP-tree and dat-lists to mine frequent patterns, our proposed algorithm only requires one database scan and can localize joining, pruning, and support counting to a small number of dat-lists. Therefore, our proposed algorithm is more efficient than the FITI algorithm. The experiment results show that ITP-Miner outperforms the FITI algorithm by one order of magnitude and requires less main memory storage space.

Although we have shown that the ITP-Miner algorithm can efficiently mine frequent inter-transaction patterns, there are still some issues to be addressed in future research. First, we may also extend the algorithm from 1-dimensional transaction databases to higher dimension databases. Second, without generalization, too many patterns may be mined and they may be too detailed. However, by generalizing with a concept hierarchy, we may be able to obtain patterns or rules that are more abstract and meaningful.

References

[1] Agrawal, R., Imielinski, T., Swami, A., Mining association rules between sets of items in large databases, In Proceedings of ACM SIGMOD, 1993, pp. 207-216.

[2] Agrawal, R., Srikant, R., Fast algorithms for mining association rules, In Proceedings of International Conference on Very Large Data Bases, 1994, pp. 487-499.

[3] Bayardo, R.J., Efficiently mining long patterns from databases, In Proceedings of ACM SIGMOD, 1998, pp. 85-93.

[4] Chen, G., Wei, Q., Fuzzy association rules and the extended mining algorithms, Information Sciences, Vol. 147, No. 1-4, 2002, pp. 201-228.

[5] Chiu, D.K.Y., Wong, A.K.C., Multiple pattern associations for interpreting structural and functional characteristics of biomolecules, Information Sciences, Vol. 167, No, 1-4, 2004, pp. 23-39.

[6] Feng, L., Dillon T.S., Liu, J., Inter-transactional association rules for multi-dimensional contests for prediction and their application to studying meteorological data, Data and Knowledge Engineering, Vol. 37, No. 1, 2001, pp. 85-115.

[7] Feng, L., Yu, J.X., Lu, H., Han, J., A template model for multidimensional inter-transactional association rules, The VLDB Journal, Vol. 11, No. 2, 2002, pp. 153-175.

[8] Hsu, P.Y., Chen, Y.L., Ling, C.C., Algorithms for mining association rules in bag databases, Information Sciences, Vol. 166, No. 1-4, 2004, pp. 31-47.

[9] Kohavi, R., Broadley, C., Frasca, B., Mason, L., Zheng, Z., KDD-cup 2000 organizers' report: peeling the onion, SIGKDD Explorations, Vol. 2, No. 2, 2000, pp. 86-98.

[10] Lee, A.J.T., Hong, R.W., Ko, W.M., Tsao, W.K., Lin, H.H, Mining spatial association rules in image databases, Information Sciences, Vol. 177, No. 7, 2007, pp. 1593-1608.

[11] Lee, A.J.T., Lin, W.C., Wang, C.S., Mining association rules with multi-dimensional constraints, The Journal of Systems and Software, Vol. 79, No. 1, 2006, pp. 79-92.

[12] Lee, A.J.T., Wang, Y.T., Efficient data mining for calling path patterns in GSM networks, Information Systems, Vol. 28, No. 8, 2003, pp. 929-948.

[13] Lee, G.., Yang, W., Lee, J.M., A parallel algorithm for mining multiple partial periodic patterns, Information Sciences, Vol. 176, No. 24, 2006, pp. 3591-3609.

[14] Li, Q., Feng, L., Wong, A., From intra-transaction to generalized inter-transaction: landscaping multidimensional contexts in association rule mining, Information Sciences, Vol. 172, No. 3-4, 2005, pp. 361-395.

[15] Littau, D., Boley, D., Streaming data reduction using low-memory factored representations, Information Sciences, Vol. 176, No. 14, 2006, pp. 2016-2041.

[16] Li, Y., Zhu, S., Wang, X.S., Jajodia, S., Looking into the seeds of time: discovering temporal patterns in large transaction sets, Information Sciences, Vol. 176, No. 8, 2006, pp. 1003-1031.

[17] Lu, H., Han, J., Feng, L., Stock movement prediction and n-dimensional inter-transaction association rules, In Proceedings of ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge, 1998, pp. 1-7.

[18] Lu, H., Feng, L., Han, J., Beyond intratransaction association analysis: mining multidimensional inter-transaction association rules, ACM Transactions on Information Systems, Vol. 18, No. 4, 2000, pp. 423-454.

[19] Savasere, A., Omiecinski, E., Navathe, S., An efficient algorithm for mining association rules in large databases, In Proceedings of International Conference on Very Large Data Bases, 1995, pp. 432-443.

[20] Shen, Li, Shen, Hong, Cheng Ling, New algorithms for efficient mining of association rules, Information Sciences, Vol. 118, No. 1-4, 1999, pp. 251-268.

[21] Shenoy, P., Haritsa, J.R., Sudarshan, S., Bhalotia, G.., Bawa, M., Shah, D., Turbo-charging vertical mining of large databases, In Proceedings of ACM SIGMOD, 2000, pp. 22-33.

[22] Tsay, Y.J., Chiang, J.Y., An efficient cluster and decomposition algorithm for mining association rules, Information Sciences, Vol. 160, No. 1-4, 2004, pp. 161-171.

[23] Tung, A.K.H., Lu, H., Han, J., Feng, L., Breaking the barrier of transactions: mining inter-transaction association rules, In Proceedings of ACM International Conference on Knowledge and Data Discovery, 1999, pp. 423-454.

[24] Tung, A.K.H., Lu, H., Han, J., Feng, L., Efficient mining of intertransaction association rules, IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 1, 2003, pp. 43-56.

[25] Wang, C.Y., Tseng, S.S., Hong, T.P., Flexible online association rule mining based on multidimensional pattern relations, Information Sciences, Vol. 176, No. 12, 2006, pp. 1752-1780.

[26] Yu, J.X., Chong, Z., Lu, H., Zhang, Z., Zhou, A., A false negative approach to mining frequent itemsets from high speed transactional data streams, Information Sciences, Vol. 176, No. 14, 2006, pp. 1986-2015.

[27] Lee, A.J.T., Wang, C.S., An efficient algorithm for mining frequent inter-transaction patterns, Information Sciences, Vol. 177, No. 17, pp. 3453-3476, 2007, NSC 95-2416-H-002-053. (SCI, impact factor= 1.003)