The Journal of Systems and Software 86 (2013) 1603–1612

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

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

Closed inter-sequence pattern mining

Chun-Sheng Wang[a], Ying-Ho Liu[b], Kuo-Chung Chu[c,*]

[a] Department of Information Management, Jinwen University of Science and Technology, No. 99, An-Chung Road, Hsin-Tien Dist., New Taipei City, Taiwan, ROC
[b] Department of Information Management, National Dong Hwa University, No. 1, Section 2, Da-Hsueh Road, Hualien 97401, Taiwan, ROC
[c] Department of Information Management, National Taipei University of Nursing and Health Sciences, No. 365, Min-Te Road, Taipei, Taiwan, ROC

ARTICLE INFO

Article history:
Received 9 May 2012
Received in revised form 17 October 2012
Accepted 5 February 2013
Available online 24 February 2013

Keywords:
Closed patterns
Data mining
Inter-sequence pattern

ABSTRACT

Inter-sequence pattern mining can find associations across several sequences in a sequence database, which can discover both a sequential pattern within a transaction and sequential patterns across several different transactions. However, inter-sequence pattern mining algorithms usually generate a large number of recurrent frequent patterns. We have observed mining closed inter-sequence patterns instead of frequent ones can lead to a more compact yet complete result set. Therefore, in this paper, we propose a model of closed inter-sequence pattern mining and an efficient algorithm called CISP-Miner for mining such patterns, which enumerates closed inter-sequence patterns recursively along a search tree in a depth-first search manner. In addition, several effective pruning strategies and closure checking schemes are designed to reduce the search space and thus accelerate the algorithm. Our experiment results demonstrate that the proposed CISP-Miner algorithm is very efficient and outperforms a compared EISP-Miner algorithm in most cases.

1. Introduction

Mining frequent patterns in various types of database has been an important issue in the area of data mining research (Agarwal et al., 2001; Lee et al., 2006, 2007). The sequential pattern mining is an essential task for the research, so many algorithms (Lee and Wang, 2003; Pei et al., 2004; Zaki, 2001) have been proposed for finding such patterns in sequence databases, in which every transaction contains a sequence. However, the traditional sequential pattern mining treats sequences independently, without considering the relationships between sequences. We call this as intra-sequence pattern mining, because all patterns are bounded in a single sequence.

In addition to intra-sequence pattern mining, several works are done for mining inter-transactional patterns (Feng et al., 2001, 2002; Lee and Wang, 2007; Lee et al., 2008; Li et al., 2005; Tung et al., 2003) recently. In Hu and Panda (2010), the authors proposed an inter-transaction data dependency mining for database intrusion detection, which uses traditional intra-transaction and intra-sequence pattern mining algorithms for mining the dependency. Besides, in Chiu et al. (2009) and Yang et al. (2009), the authors utilize the inter-transaction pattern mining model for mining repeating patterns with gap constraint and inter-transactional time series association rules, respectively. Although the model can find itemsets across several transactions, it does not consider the ordered relationships between items within a transaction, because the items are treated as unordered sets. If we want to find relationships between sequences in a database containing a set of sequences, all the above intra-transaction, intra-sequence, and inter-transaction pattern mining algorithms cannot perform the task.

Because the limitations described above, a new model of inter-sequence pattern mining was introduced in a research entitled "Mining inter-sequence patterns" (Wang and Lee, 2009). It extends the scope of sequential pattern mining from traditional intra-sequence patterns to inter-sequence patterns, which describe associations across several sequences. Inter-sequence patterns can discover both a sequential pattern within a transaction and sequential patterns across several different transactions. For example, in financial markets, a frequent inter-sequence pattern might be: "if the steel market's price index increases more than the real estate market's price index in the first month, then the real estate market's price index increases more than that of the gold market in the third month". This is a useful inter-sequence pattern that could help investors manage their portfolios, since they could invest more in real estate than in gold two months ahead. Likewise, a pattern like "if the demand for beer is greater than the demand for orange juice in week one, then the demand for soda is greater than that for cola two weeks later" could help retailers plan future beverage purchases. In another example, the following

* Corresponding author. Tel.: +886 937026166; fax: +886 937026166.
E-mail addresses: seanwang@just.edu.tw (C.-S. Wang), daxliu@mail.ndhu.edu.tw (Y.-H. Liu), d91725001@ntu.edu.tw, kcchu@ntunhs.edu.tw (K.-C. Chu).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.02.010

pattern may help weather forecasters: "if more tornados occurred in Texas than in Kansas last year, it is likely that there will be more tornados in Colorado than in Utah this year". The authors have proposed an algorithm called EISP-Miner (enhanced inter-sequence pattern Miner) for mining a complete set of frequent inter-sequence patterns. It has been shown that the EISP-Miner algorithm outperforms an Apriori-like algorithm (Wang and Lee, 2009).

Since frequent inter-sequence mining breaks the boundaries of sequences, the number of potential patterns will increase substantially and the mining process could be extremely time-consuming. The huge number of patterns will make an inter-sequence mining algorithm inefficient. In recent years, many studies have presented convincing arguments that for mining frequent patterns, one should not mine all frequent patterns but mine all the closed ones because the latter leads to not only a more compact yet complete result set but also to better efficiency (Lee et al., 2008; Liu et al., 2009; Wang et al., 2007; Wu and Lee, 2010; Yan et al., 2003; Zaki and Hsiao, 2005). A frequent pattern p is closed if the database does not contain a super-pattern of p with equal support. For example, an inter-sequence pattern: "if the steel market's price index increases in the first month, then the real estate market's price index increases more than that of the gold market in the third month" is a sub-pattern of the previous cited example. If both patterns have equal supports, this sub-pattern is not closed. By this example, we can observe that under the same strength (support), the sub-pattern contains less information then its super-pattern, thus the sub-pattern is redundant and can be pruned. Generally speaking, the number of closed patterns in a database is often much smaller than the number of frequent patterns. Since the closed patterns may be used to generate a complete set of frequent patterns, mining closed patterns would be more efficient than mining all frequent patterns.

From the above observation, therefore, in this paper, we propose an approach called the CISP-Miner algorithm for efficient mining of closed inter-sequence patterns in a large sequence database. The proposed algorithm consists of two phase. First, it scans the sequence database to construct all *Plist* of frequent 1-patterns, which stores a frequent pattern and a list of location information for that pattern. Next, the CISP-Miner constructs a CIS-tree to generate the closed inter-sequence patterns recursively in a depth-first search (DFS) manner. By using the CIS-tree and Plists, the effective pruning strategies and closure checking schemes can be embedded to avoid costly candidate generation and repeated support counting. Therefore, the CISP-Miner algorithm can efficiently mine the closed inter-sequence patterns. Up to present, since EISP-Miner is the most efficient algorithm for mining inter-sequence patterns (Wang and Lee, 2009), we conduct experiments by comparing EISP-Miner and CISP-Miner algorithms it to show the efficiency of our proposed algorithm.

The contribution of this paper is threefold. First, we introduce the innovative concept of closed inter-sequence patterns. Second, we propose an efficient algorithm for mining closed inter-sequence patterns, which uses several pruning strategies and closure checking schemes to reduce the search space and thus speed up the algorithm. Third, we demonstrate via experiments that the proposed algorithm is efficient and scalable, and outperforms the EISP-Miner algorithm in most cases.

The remainder of this paper is organized as follows. Section 2 introduces the closed inter-sequence pattern mining problem and defines some notations. In Section 3, we present the CISP-Miner algorithm. Section 4 describes the experiments and their performance results. Finally, we present our conclusions and indicate some future research directions in Section 5.

2. The problem

In traditional sequential pattern mining models, each transaction in database contains a sequence of itemsets. Although transactions may occur in different contexts, such as time and location, this contextual information is ignored in traditional sequential pattern mining because the patterns are intra-transactional in nature. However, if we are interested in inter-sequence patterns across multiple transactions, the context in which a transaction occurs is important. An enhanced sequence database for inter-sequence pattern mining is defined as follows. An itemset $I=(i_1 i_2 \ldots i_n)$ is a set of distinct items. When there is only one item in an itemset, the parentheses can be omitted; that is, (i) can be written as i. Items in an itemset are listed in alphabetical order. A sequence $S=\langle t_1 t_2 \ldots t_m\rangle$ is an ordered list of itemsets, where t_j is an itemset for $1\le j\le m$. A sequence database $D=\{s_1, s_2, \ldots, s_{|D|}\}$, where $|D|$ is the number of transactions in D and s_i $(1\le i\le |D|)$ is a transaction of the form (*Dat*, *Sequence*). *Dat* is the domain attribute of s_i that describes the contextual information, such as the time stamp or space location associated with s_i. To demonstrate the closed inter-sequence pattern mining algorithms, Table 1 shows a sequence database containing four transactions.

An inter-sequence context can be defined through the domain attribute of a sequence database. Let dat_1 and dat_2 be domain attributes for sequences S_1 and S_2, respectively. If we take dat_1 as the reference point, the span between S_1 and S_2 is defined as $dat_2 - dat_1$. If $S_2=\langle t_1 t_2 \ldots t_n\rangle$, where t_j is an itemset $(1\le j\le n)$, the sequence S_2 at domain attribute dat_2 with respect to dat_1 is called an extended sequence (e-seq for short) and denoted as $S_2^{dat2-dat1}=\left\langle t_1^{dat2-dat1} t_2^{dat2-dat1} \ldots t_n^{dat2-dat1}\right\rangle$. For example, in Table 1, the e-seq of the second transaction with respect to the first one is $\langle ca(ab)\rangle^{2-1}=\langle c^1 a^1 (ab)^1\rangle$; the e-seq of the third transaction with respect to the first one is $\langle b\rangle^{3-1}=\langle b^2\rangle$; and the e-seq of the fourth transaction with respect to the second one is $\langle ad\rangle^{4-2}=\langle a^2 d^2\rangle$. Suppose we have an e-seq $s^i=\left\langle t_1^i t_2^i \ldots t_n^i\right\rangle$, where t_j is an itemset $(1\le j\le n)$ and i is the span of s. We define t_j associated with i as an extended itemset (e-iset for short) denoted by t_j^i. Also, if $t_j=(u_1 u_2 \ldots u_m)$, where u_k is an item $(1\le k\le m)$, we define u_k associated with i as an extended item (e-item for short) denoted by u_k^i. For example, the extended sequence $\langle c^1 a^1 (ab)^1\rangle$contains three e-isets: c^1, a^1, and $(ab)^1$, or four e-items: c^1, a^1, a^1, and b^1.

Given a list of k consecutive transactions $z_1=(t_1, s_1)$, $z_2=(t_2, s_2)$,..., $z_k=(t_k, s_k)$ in a sequence database, $w=s_1^0 \cup s_2^{t2-t1} \cup \ldots \cup s_k^{tk-t1}$ is called a megasequence, where $k\ge 1$. In a megasequence the span of the last transaction must be less than or equal to *maxspan* (i.e., $t_k - t_1 \le maxspan$ in w), where *maxspan* is a user-specified threshold. If we set *maxspan* = 1, the sequence database in Table 1 contains four megasequences: $\langle a^0(abc)^0 c^1 a^1 (ab)^1\rangle$, $\langle c^0 a^0 (ab)^0 b^1\rangle$, $\langle b^0 a^1 d^1\rangle$, and $\langle a^0 d^0\rangle$.

Let $\alpha=s_1^{w1}, s_2^{w2}, \ldots, s_m^{wm}$ be a list of e-seqs, which can be normalized a as a pattern $\beta=s_1^0, s_2^{w2-w1}, \ldots, s_m^{wm-w1}$. The number of e-items in a pattern is called the length of the pattern, and a pattern of length k is called a k-pattern. Given two sequences, $S=\langle s_1 s_2 \ldots s_n\rangle$ and $S'=\left\langle s'_1 s'_2 \ldots s'_m\right\rangle$, where $n\le m$, we say that S

Table 1
A sequence database D and its megasequences with *maxspan* = 1.

Dat	Sequence	Magasequence (*maxspan* = 1)			
1	$\langle a(abc)\rangle$	$\langle a^0(ab)^0\rangle$			
2	$\langle ca(ab)\rangle$	$\langle c^1 a^1 (ab)^1\rangle$	$\langle c^0 a^0 (ab)^0\rangle$		
3	$\langle b\rangle$		$\langle b^1\rangle$	$\langle b^0\rangle$	
4	$\langle ad\rangle$			$\langle a^1 d^1\rangle$	$\langle a^0 d^0\rangle$

Table 2
The patterns mined from Table 1 with *maxspan* = 1 and *minsup* = 2.

Length	Frequent inter-sequence patterns	Closed inter-sequence patterns
1	$\langle a^0\rangle$:3, $\langle b^0\rangle$:3, $\langle c^0\rangle$:2,	$\langle a^0\rangle$:3, $\langle b^0\rangle$:3
2	$\langle (ab)^0\rangle$:2, $\langle a^0a^0\rangle$:2, $\langle a^0b^0\rangle$:2, $\langle a^0b^1\rangle$:2, $\langle b^0a^1\rangle$:2, $\langle b^0b^1\rangle$:2, $\langle c^0b^1\rangle$:2	$\langle b^0a^1\rangle$:2, $\langle c^0b^1\rangle$:2
3	$\langle a^0(ab)^0\rangle$:2, $\langle (ab)^0b^1\rangle$:2, $\langle a^0a^0b^1\rangle$:2, $\langle a^0b^0b^1\rangle$:2	
4	$\langle a^0(ab)^0b^1\rangle$:2	$\langle a^0(ab)^0b^1\rangle$:2

is a subsequence of S' if we can find $s_1 \subseteq s'_{j1}$, $s_2 \subseteq s'_{j2}$,..., $s_n \subseteq s'_{jn}$, such that $1 \le j_1 < j_2 < \ldots < j_n \le m$. For example, $\langle a(ac)dc\rangle$ is a subsequence of $\langle a(abc)(ac)d(cf)\rangle$, but it is not a subsequence of $\langle (abc)(ac)d(f)\rangle$. Assume there are two patterns, $P = s_1^{w1}, s_2^{w2}, \ldots, s_n^{wn}$ and $P' = s'^{v1}_1, s'^{v2}_2, \ldots, s'^{vm}_m$, where $n \le m$. Then, P is a subpattern of P' if we find $S_1^{w1} \subseteq S'^{vj1}_{j1}$, $S_2^{w2} \subseteq S'^{vj2}_{j2}$, ..., $S_n^{wn} \subseteq S'^{vjn}_{jn}$, such that $1 \le j_1 < j_2 < \ldots < j_n \le m$ and $w_1 = v_{j1}, w_2 = v_{j2}, \ldots, w_n = v_{jn}$. We can also say that $P \subseteq P'$ or P' contains P. For example, both $\langle d^0a^3c^3\rangle$ and $\langle (ad)^0(ad)^1a^1\rangle$ are subpatterns of $\langle c^0(ad)^0b^1(ad)^1a^1a^3c^3d^3a^3\rangle$.

In a sequence database D, let P be a pattern, and T_P be a set of megasequences in D, where each megasequence in T_P contains P. The support of P, $sup(P)$, is defined as $|T_P|$. If $sup(P)$ is not less than the user-specified minimum support threshold *minsup*, P is called a frequent pattern. An *inter-sequence association rule* is written in the form of $P \to Q$, where both P and $P \cup Q$ are frequent patterns; $P \cap Q = \varnothing$, $conf(P \to Q)$ is not less than the user-specified minimum confidence; and the confidence of the rule $conf(P \to Q)$ is defined as $sup(P \cup Q)/sup(P)$. Furthermore, let X and Y be two patterns, we say that Y subsumes X if Y contains X and both have equal support. For example, assume that $X = \langle d^0a^2a^2\rangle$and $Y = \langle (ad)^0(ad)^2a^2\rangle$. Since we have $X \subseteq Y$, if X and Y have equal support, Y subsumes X. A frequent pattern X is closed if there does not exist a pattern Y such that (1) $X \subseteq Y$, and (2) $sup(X) = sup(Y)$, i.e., X cannot be subsumed by another pattern. According to previous studies (Lee et al., 2008; Liu et al., 2009; Wang et al., 2007; Wu and Lee, 2010; Zaki and Hsiao, 2005), all frequent patterns in a database can be generated from the closed patterns. Once the frequent patterns are identified, the association rules can be derived in a straightforward manner. Therefore the problem of mining closed inter-sequence patterns involves finding all frequent and closed inter-sequence patterns in a sequence database with respect to the user-specified *minsup* and *maxspan* thresholds.

Let us consider the example shown in Table 1. Assume that *maxspan* = 1, *minsup* = 2 and $X = \langle a^0b^1\rangle$. Since the megasequence formed by the first and second transactions, $\langle a^0(abc)^0c^1a^1(ab)^1\rangle$, contains X, sup(X) is set to 1. Also, the megasequence formed by the second and third transactions, $\langle c^0a^0(ab)^0b^1\rangle$, contains X. Thus, $sup(X)$ is incremented by 1. That is, $sup(X) = 2$. Consequently, X is a frequent pattern. Let another pattern $Y = \langle a^0(ab)^0b^1\rangle$, we can find $sup(Y) = 2$ so Y is a frequent pattern. Since $X \subseteq Y$ and $sup(X) = sup(Y)$, X is subsumed by Y and is not closed. Similarly, since Y has no super-pattern with equal support, it is a closed pattern.

Table 2 shows the complete set of frequent inter-sequence patterns and closed inter-sequence patterns with their supports listed afterwards. In this table, patterns are mined from the sequence database in Table 1 with *maxspan* = 1 and *minsup* = 2, where the number of frequent inter-sequence patterns is 15 and the number of closed ones is 5. From this example, we can observe that the closed patterns are more compact then the frequent ones.

3. The proposed method

We now introduce the CISP-Miner algorithm. In Section 3.1 we present the data structure Plist, and the search tree CIS-tree. The closure checking schemes, pruning strategies, and examples are explained in Section 3.2. Finally, we describe the algorithm in detail in Section 3.3.

3.1. Plist and CIS-tree

Since the proposed algorithm uses a Plist to record information about an inter-sequence pattern in a CIS-tree, we begin by defining Plist and CIS-tree.

Definition 1 *((Plist)).* A Plist is defined as *X-list* = *X*-{(t_1.p_1), (t_2.p_2),..., (t_n.p_n)}, where X is a pattern and *list* is a list of two-dimensional points; t_i is the *dat* (*t*-value); and p_i is the position (*p*-value) that X's last e-item contained in the sequence database. We also define *count*(*X-list*) as the number of points and *sup*(*X-list*) as the number of distinct *t*-values contained in *X-list*. If X is a k-pattern and $sup(X\text{-}list) \ge minsup$, we say that *X-list* is a frequent k-Plist.

For example, let *X-list* = $\langle a^0\rangle$ – {(1.1),(1.2),(2.2),(2.3),(4.1)} in the sequence database shown in Table 1. *X-list* is a Plist with five points, (1.1), (1.2), (2.2), (2.3), and (4.1), where the point (1.2) means $\langle a^0\rangle$ contained in the second itemset (*p*-value) of the sequences with *dat* 1 (*t*-value). If *minsup* = 2, we have *count*(*X-list*) = 5, *sup*(*X-list*) = 3, and *X-list* is a frequent 1-Plist. For another example, let *Y-list* = $\langle b^0a^1\rangle$ – {(2.2),(2.3),(4.1)}, we can observe that its last e-item $\langle a^1\rangle$ is contained in the second and third itemsets of the sequences with *dat* 2 and in the first itemset of the sequence with *dat* 4. Therefore, we have *count*(*Y-list*) = 3, *sup*(*Y-list*) = 2, and *Y-list* is a frequent 2-Plist.

Definition 2 *((CIS-tree)).* A CIS-tree is a search tree denoted as T in which every node contains a Plist. In T, the root node represents an empty Plist and all Plists of frequent k-patterns are recorded at level k of the tree, such that $k \ge 1$. Moreover, in T, if X_i-*list* is a child node of *X-list*, X_i is obtained by adding an e-item to X and using an edge to connect both nodes.

Let us consider the database shown in Table 1, and assume that *maxspan* = 1 and *minsup* = 2. As shown in Fig. 1, all frequent inter-sequence patterns can be enumerated in the CIS-tree. Note that joining the Plists in the CISP-tree is the key operation to generating frequent patterns and the CISP-tree can be condensed to a smaller one in the proposed algorithm by using some closure checking schemes and pruning strategies. Based on the concepts introduced in Wang and Lee (2009), the problem of mining frequent inter-sequence patterns has the following properties.

Property 1 ((Joining of 1-patterns)). Let $X = \langle x_1^0\rangle$ and $Y = \langle y_1^0\rangle$ be two frequent 1-patterns. X is joinable to Y in any instance, which yields three types of join operations: (1) itemset-join: $X\ \theta_i\ Y = \langle (x_1y_1)^0\rangle$, where $x_1 < y_1$; (2) sequence-join: $X\ \theta_s\ Y = \langle x_1^0y_1^0\rangle$; (3) inter-join: $X\ \theta_t^d\ Y = \langle x_1^0y_1^d\rangle$, where $0 < d \le maxspan$.

For example, if *maxspan* = 1, $\langle a^0\rangle\ \theta_i\ \langle b^0\rangle = \langle (ab)^0\rangle$, $\langle a^0\rangle\ \theta_s\ \langle b^0\rangle = \langle a^0b^0\rangle$, and $\langle a^0\rangle\ \theta_t^1\ \langle b^0\rangle = \langle a^0b^1\rangle$.

Property 2 ((Joining of k-patterns)). Let $X = \langle x_1^{i1}x_2^{i2}\ldots x_k^{ik}\rangle$ and $Y = \langle y_1^{j1}y_2^{j2}\ldots y_k^{jk}\rangle$ be two frequent k-patterns, where $k > 1$. X is joinable to Y if the first $k-1$ e-items of X are equal to those of Y, which yields three types of join operations: (1) itemset-join: $X\ \theta_i\ Y = \langle x_1^{i1}x_2^{i2}\ldots(x_ky_k)^{jk}\rangle$, where $x_k < y_k$ and $i_k = j_k$; (2) sequence-join: $X\ \theta_s\ Y = \langle x_1^{i1}x_2^{i2}\ldots x_ky_k^{jk}\rangle$, where $i_k = j_k$; (3) inter-join: $X\ \theta_t^d\ X = \langle x_1^{i1}x_2^{i2}\ldots x_k^{ik}y_k^{jk}\rangle$, where $i_k < j_k$ and $j_k - i_k = d$.

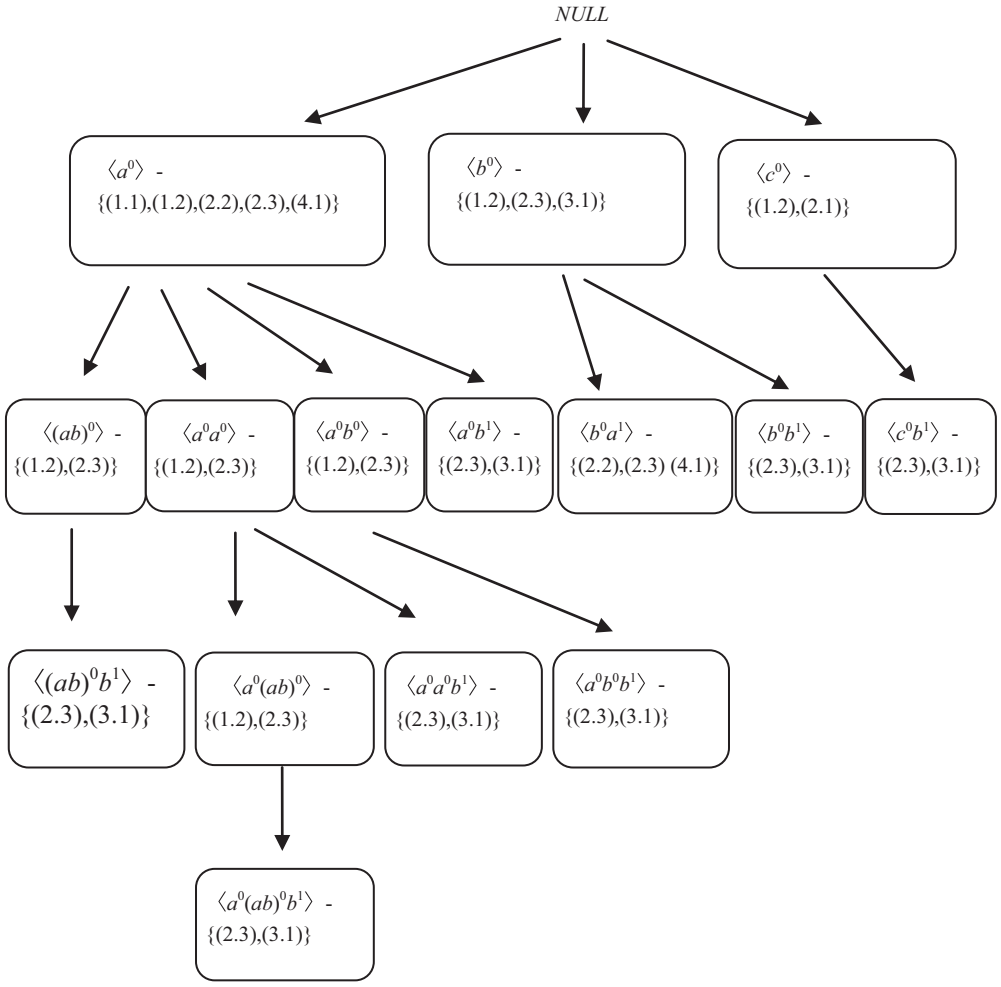

Fig. 1. The CIS-tree of frequent patterns.

For example, $\langle a^0a^1\rangle\ \theta_i\ \langle a^0b^1\rangle = \langle a^0(ab)^1\rangle$, $\langle a^0a^1\rangle\ \theta_s\ \langle a^0b^1\rangle = \langle a^0a^1b^1\rangle$, and $\langle a^0a^0\rangle\ \theta_t{}^1\ \langle a^0b^1\rangle = \langle a^0a^0b^1\rangle$.

Property 3 ((Joining of Plists)). Let $X\text{-}list = X\text{-}\{(tx_1.px_1), (tx_2.px_2), \ldots, (tx_n.px_n)\}$ and $Y\text{-}list = Y\text{-}\{(ty_1.py_1), (ty_2.py_2), \ldots, (ty_m.py_m)\}$ be two frequent k-Plists, where $k \geq 1$. *X-list* is joinable to *Y-list* if X is joinable to Y, which yields three types of join operations: (1) itemset-join: $X\text{-}list\ \theta_i\ Y\text{-}list = X\ \theta_i\ Y\text{-}\{(ty_{i1}.py_{i1}), (ty_{i2}.py_{i2}), \ldots, (ty_{ik}.py_{ik})\}$, where $ty_\alpha = tx_\beta$ and $py_\alpha = px_\beta$; (2) sequence-join: $X\text{-}list\ \theta_s\ Y\text{-}list = X\ \theta_s\ Y\text{-}\{(ty_{i1}.py_{i1}), (ty_{i2}.py_{i2}), \ldots, (ty_{ik}.py_{ik})\}$, where $ty_\alpha = tx_\beta$ and $py_\alpha > px_\beta$; (3) inter-join: $X\text{-}list\ \theta_t^d\ Y\text{-}list = X\ \theta_t^d\ Y\text{-}\{(ty_{i1}.py_{i1}), (ty_{i2}.py_{i2}), \ldots, (ty_{ik}.py_{ik})\}$, where $ty_\alpha - tx_\beta = d$.

For example, $\langle a^0\rangle\text{-}\{(1.1),(1.2),(2.2),(2.3),(4.1)\}\ \theta_i\ \langle b^0\rangle\text{-}\{(1.2),(2.3),(3.1)\} = \langle (ab)^0\rangle\text{-}\{(1.2),(2.3)\}$, $\langle c^0\rangle\text{-}\{(1.2),(2.1)\}\ \theta_s\ \langle a^0\rangle\text{-}\{(1.1),(1.2),(2.2),(2.3),(4.1)\} = \langle c^0a^0\rangle\text{-}\{(2.2),(2.3)\}$, and $\langle (ab)^0\rangle\text{-}\{(1.2),(2.3)\}\ \theta_t{}^1\ \langle a^0b^1\rangle\text{-}\{(2.3),(3.1)\} = \langle (ab)^0b^1\rangle\text{-}\{(2.3),(3.1)\}$.

Property 4 ((CIS-tree property)). For a Plist *x-list* in a CISP-tree T, we define (1) *joinable group* of *x-list* as $J(x\text{-}list) = \{c_1\text{-}list, c_2\text{-}list, \ldots, c_n\text{-}list\}$, where *x-list* is joinable to c_i-*list* ($1 \leq i \leq n$); and (2) *extended group* of *x-list* as $E(x\text{-}list) = \{f_1\text{-}list, f_2\text{-}list, \ldots, f_m\text{-}list\}$, where each f_j-*list* is a child of *x-list* in T ($1 \leq j \leq m$). Let *X-list* and *Y-list* be two Plists in T, where *Y-list* is a child of *X-list* in T, then we have: (1) $Y\text{-}list \in E(X\text{-}list)$; (2) $J(Y\text{-}list) \subseteq E(X\text{-}list)$; (2) $E(Y\text{-}list) = \{Y\text{-}list\ \theta\ c_i\text{-}list\}$, where $c_i\text{-}list \in J(Y\text{-}list)$ and $\theta \in \{\theta_i, \theta_s, \theta_t^d\}$.

For example, let $X\text{-}list = \langle a^0\rangle\text{-}\{(1.1),(1.2),(2.2),(2.3),(4.1)\}$ and $Y\text{-}list = \langle (ab)^0\rangle\text{-}\{(1.2),(2.3)\}$ in Fig. 1, we can observe that $J(X\text{-}list) = \{\langle a^0\rangle\text{-list}, \langle b^0\rangle\text{-list}, \langle c^0\rangle\text{-list}\}$; $E(X\text{-}list) = \{\langle (ab)^0\rangle\text{-list}, \langle a^0a^0\rangle\text{-list}, \langle a^0b^0\rangle\text{-list}, \langle a^0b^1\rangle\text{-list}\}$; $J(Y\text{-}list) = \{\langle a^0a^0\rangle\text{-list}, \langle a^0b^0\rangle\text{-list}, \langle a^0b^1\rangle\text{-list}\}$; and $E(Y\text{-}list) = \{\langle (ab)^0b^1\rangle\text{-list}\}$. From the figure, we can know that $E(X\text{-}list)$ can be obtained by joining *X-list* with each Plist in $J(X\text{-}list)$, while $E(Y\text{-}list)$ can be obtained by joining *Y-list* with each Plist in $J(Y\text{-}list)$. Therefore, based on this property, the inter-sequence pattern mining problem can be partitioned recursively (Wang and Lee, 2009).

3.2. Closure checking and pruning strategies

The properties described in Section 3.1 can only be used to mine a complete set of frequent inter-sequence patterns instead of closed ones. When getting a new frequent inter-sequence pattern, we need to do pattern closure checking to determine whether it is closed or not, that is, it cannot be subsumed by one of its super-pattern with the same support. Therefore, we device a hash table H to filter out non-closed inter-sequence patterns efficiently. In the mining process, when a frequent k-Plist $X\text{-}list = \left\langle x_1^{d1}x_2^{d2}\ldots x_k^{dk}\right\rangle - \{(tx_1.px_1), (tx_2.px_2), \ldots (tx_n.px_n)\}$ is discovered, X's hash value is computed by $hv(X) = \sum(tx_i - dk) \bmod hashsize$, where each tx_i is distinct and $1 \leq i \leq n$. If $hv(X)$ and $sup(X)$ are equal to those of a pattern Z in H, then (1) if $X \subseteq Z$, X is not closed, thus X is discarded; or (2) if $Z \subseteq X$,

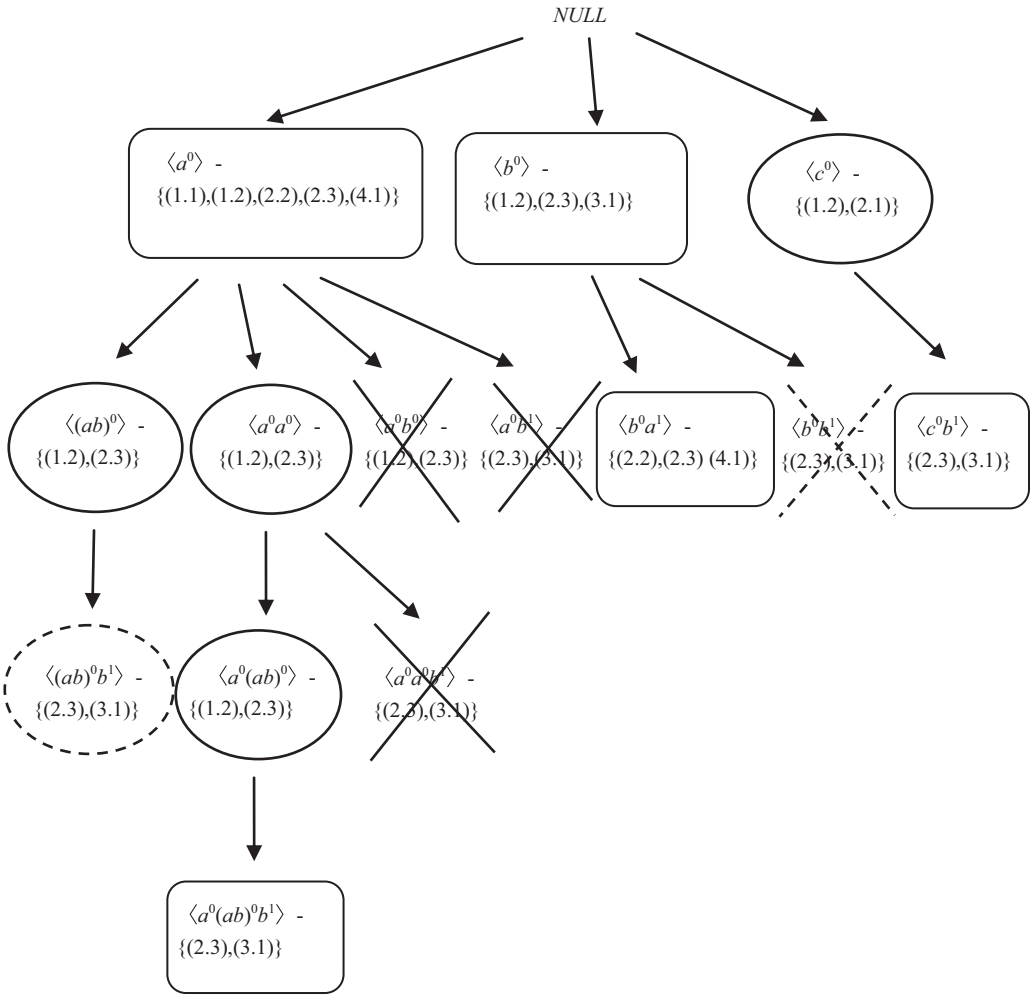

Fig. 2. The CIS-tree of closed patterns.

Z is not closed, thus Z is discarded from H and X is added to H. For example, assume *hashsize* = 9, let us consider $\langle a^0\rangle$-{(1.1),(1.2),(2.2),(2.3),(4.1)} and $\langle b^0a^1\rangle$-{(2.2),(2.3),(4.1)} in Fig. 2, we can compute $hv(\langle a^0\rangle) = ((1-0)+(2-0)+(4-0)) \bmod 9 = 7$, $hv(\langle b^0a^1\rangle) = ((2-1)+(4-1)) \bmod 9 = 4$. Since $H[7]$ and $H[4]$ are empty, $\langle a^0\rangle$ and $\langle b^0a^1\rangle$ are added to $H[7]$ and $H[4]$ in Table 3, respectively.

In the following, we propose two closure checking schemes and two pruning strategies to efficiently mine closed inter-sequence patterns.

Lemma 1 ((Join checking)). A pattern X is not closed if it meets any of the following conditions: (1) there exists a Plist *Y-list* so that *sup*(*X-list*) = *sup*(*X-list* θ *Y-list*), where $\theta \in \{\theta_i, \theta_s, \theta_t^d\}$; or (2) there exists a Plist *Y-list* so that *sup*(*X-list*) = *sup*(*Y-list* θ *X-list*), where $\theta \in \{\theta_i, \theta_s, \theta_t^d\}$.

Table 3
The content of H.

Hash value	Patterns	Support
3	~~$\langle (ab)^0b^1\rangle$~~	2
	$\langle a^0(ab)^0b^1\rangle$	2
	$\langle c^0b^1\rangle$	2
4	$\langle b^0a^1\rangle$	2
6	$\langle b^0\rangle$	3
7	$\langle a^0\rangle$	3

Proof. If $X = \left\langle x_1^{dx1}x_2^{dx2}\ldots x_k^{dxk}\right\rangle$ and $Y = \left\langle y_1^{dy1}y_2^{dy2}\ldots y_k^{dyk}\right\rangle$, the first condition means that in each megasequence that contains X in the database, at least one y_k^{dyk} appears after X. Since the super-pattern formed by X and y_k^{dyk} is $\left\langle x_1^{dx1}x_2^{dx2}\ldots x_k^{dxk}y_k^{dyk}\right\rangle$ that has the same support as X, X is not closed. Similarly, the second condition means that in each megasequence that contains X in the database, at least one y_k^{dyk} appears before x_k^{dxk}. Since the super-pattern formed by X and y_k^{dyk} is $\left\langle x_1^{dx1}x_2^{dx2}\ldots x_k^{dxk}y_k^{dyk}\right\rangle$ that has the same support as X, X is not closed.

For example in Fig. 2, since *sup*($\langle c^0\rangle$-list) = *sup*($\langle c^0\rangle$-list θ_t^1 $\langle b^0\rangle$-list) = *sup*($\langle c^0b^1\rangle$-list) = 2, the pattern $\langle c^0\rangle$ is not closed.

Lemma 2 ((Hash table checking)). In a hash table H, a pattern X is not closed if it cannot meet Lemma 1, but there exists a pattern Z in H so that $X \subseteq Z$ and *sup*(X) = *sup*(Z).

Proof. Since $X \subseteq Z$ and both X and Z has equal support, X is not closed by the definition of closed pattern.

For example in Table 3, since $\langle (ab)^0b^1\rangle \subseteq \langle a^0(ab)^0b^1\rangle$ and *sup*($\langle (ab)^0b^1\rangle$) = *sup*($\langle a^0(ab)^0b^1\rangle$) = 2, the pattern $\langle (ab)^0b^1\rangle$ is not closed.

Lemma 3 ((Join pruning)). A k-Plists *Y-list* does not need to be extended, i.e., we set $J(Y\text{-}list) = NULL$ and $E(Y\text{-}list) = NULL$, if $k > 1$ and there exists a *X-list* so that $count(Y\text{-}list) = count(X\text{-}list\ \theta\ Y\text{-}list)$.

Proof. If $X = \left\langle x_1^{dx1} x_2^{dx2} \ldots x_k^{dxk} \right\rangle$ and $Y = \left\langle y_1^{dy1} y_2^{dy2} \ldots y_k^{dyk} \right\rangle$, the condition means that in each point $(t_y.p_y)$ that y_k^{dyk} appears in the database, there exists a point $(t_x.p_x)$ in *X-list* that located before $(t_y.p_y)$, in other words, at least one x_k^{dxk} appears before y_k^{dyk}. Since the super-pattern formed by X joining Y is Z, which has the same count of points as Y, we have $count(Z) = count(Y)$, $sup(Z) = sup(Y)$, and $E(Z) = E(X)$. It means each frequent pattern extended from Y is a subpattern of a frequent pattern extended from Z and both have equal support. Therefore, *Y-list* does not need to be extended.

For example in Fig. 2, since $count(\langle a^0 b^0\rangle\text{-list}) = count(\langle a^0 a^0\rangle\text{-list}\ \theta_i\ \langle a^0 b^0\rangle\text{-list}) = count(\langle a^0 (ab)^0\rangle\text{-list}) = 2$, the $\langle a^0 b^0\rangle$-list does not need to be extended.

Lemma 4 ((Hash table pruning)). A k-Plists *X-list* does not need to be extended, i.e., we set $J(X\text{-}list) = NULL$ and $E(X\text{-}list) = NULL$, if $k > 1$, *X-list* cannot meet Lemma 1, and there exists a pattern Z which has already stored in H so that $X \subseteq Z$ and $sup(X) = sup(Z)$.

Proof. If $X = \left\langle x_1^{dx1} x_2^{dx2} \ldots x_{k-1}^{dxk-1} x_k^{dxk} \right\rangle$ and $Z = \left\langle x_1^{dx1} x_2^{dx2} \ldots z_i^{dzi} x_{k-1}^{dxk-1} x_k^{dxk} \right\rangle$, the condition $X \subseteq Z$ means that there exists an e-item z_i^{dzi} inserted in X to form Z. In other words, there must exist at least one point of z_i^{dzi} that appears either before or after each point in *X-list*, which can be partially checked by lemma 1. Since *X-list* does not meet the first condition of Lemma 1, it is not possible to insert z_i^{dzi} after x_k^{dxk} in X to form Z. Similarly, since *X-list* does not meet the second condition of Lemma 1, it is not possible to insert z_i^{dzi} between x_{k-1}^{dxk-1} and x_k^{dxk} in X either. From the above analysis, we can conclude that z_i^{dzi} should be inserted before x_{k-1}^{dxk-1} in X to form Z. Since Z has already stored in H and has the same support as X, it means that (1) all patterns extended from Z have already generated, (2) $E(Z) = E(X)$, and (3) each frequent pattern extended from X is a subpattern of a pattern extended from Z and has equal support as X. Therefore, *X-list* does not need to be extended.

For example in Fig. 2, since $\langle b^0 b^1\rangle$-list cannot meet Lemma 1, and there exists a pattern $\langle a^0 (ab)^0 b^1\rangle$ in Table 3 so that $\langle b^0 b^1\rangle \subseteq \langle a^0 (ab)^0 b^1\rangle$ and $sup(\langle b^0 b^1\rangle) = sup(\langle a^0 (ab)^0 b^1\rangle) = 2$, the $\langle b^0 b^1\rangle$-list does not need to be extended. Fig. 2 and Table 3 illustrate how the closure checking schemes and pruning strategies are applied for the example shown in Fig. 1. The symbols used in Fig. 2 are described as follows: (1) a solid circle indicates that the pattern is not closed by the join checking; (2) a dotted circle indicates that the pattern is not closed by the hash table checking; (3) a solid cross symbol indicates that the Plist is pruned by the join pruning; (4) a dotted cross symbol indicates that the Plist is pruned by the hash table pruning; and (5) a rectangle indicates that the pattern is closed. In Table 3, a horizontal line indicates that the pattern is not closed by the hash table checking.

3.3. The CISP-Miner algorithm

Fig. 3 shows our proposed algorithm called CISP-Miner to find the complete set of closed inter-sequence patterns. In step 1 of the figure, we scan the sequence database to generate a set of all frequent 1-Plists as the extended group of the root node of a CISP-tree. Then in step 2, we construct an empty hash table that stores closed inter-sequence patterns and prunes unnecessary nodes in the CISP-tree. Finally in steps 3 and 4, we recursively enumerate all closed inter-sequence patterns in a depth first search manner by calling the Join procedure, which is described in Fig. 4.

Algorithm: CISP-Miner(D, *minsup*, *maxspan*): Closed Inter-Sequence Patterns Miner.
Input: A sequence database D, a minimum support *minsup*, and a minimum span *maxspan*.
Output: The complete set of closed inter-sequence patterns H.
1. Scan D to generate a set of all frequent 1-Plists, $E(NULL)$, as the extended group of the root node of an CISP-tree T;
2. Construct an empty hash table H;
3. Call Join($E(NULL)$, *minsup*, *maxspan*, H);
4. Output H;

Fig. 3. The CISP-Miner algorithm.

For the Join procedure in Fig. 4, steps 1–11 are used for joining of 1-patterns, while steps 12–23 are used for joining of k-patterns, where $k > 1$. In steps 2–9, for each frequent 1-Plist *X-list*, we join *X-list* with each frequent 1-Plist *Y-list* in $J(X\text{-}list)$ to get $E(X\text{-}list)$, where $J(X\text{-}list)$ is the set equal to $E(NULL)$ and $E(X\text{-}list)$ is the set of frequent 2-Plists extended by *X-list*. For a Plist *W-list* that is not *NULL* in step 12, we use the Join and Hash table checking schemes (Lemmas 1 and 2) to filter out non-closed patterns in step 13, and then use the Hash table pruning strategy (Lemma 4) to check if *W-list* can be pruned in step 14. In steps 15–21, for each frequent k-Plist *X-list* ($k > 1$) in $E(W\text{-}list)$, we join *X-list* with each frequent k-Plist *Y-list* in $J(X\text{-}list)$ to get $E(X\text{-}list)$, where $J(X\text{-}list)$ is a subset of $E(W\text{-}list)$ containing all frequent k-Plists that *X-list* can join and $E(X\text{-}list)$ is the set of frequent $(k+1)$-Plists extended by *X-list*. In each join operation, we use the Join pruning strategy (Lemma 3) to check if *Y-list* can be pruned in step 20. After all $E(X\text{-}list)$ have been collected in steps 10 and 22, we apply the Join procedure to find longer patterns recursively in a depth-first search manner until no further frequent Plists can be generated.

Lemma 5. Every inter-sequence pattern found by the CISP-Miner algorithm is frequent and closed.

Proof. In step 1 of the CISP-Miner algorithm, we scan the sequence database once to find all frequent 1-patterns, thus every 1-pattern must be frequent. In the Join procedure, at each level of extension, we use the Properties 1, 2, 3 and 4 to join the frequent k-Plists in the previous level to form frequent $(k+1)$-patterns, $k \geq 1$. Moreover, at each level of extension, we use the Join and Hash table checking schemes in lemmas 1 and 2 to eliminate non-closed patterns.

Procedure: Join($E(W\text{-}list)$, *minsup*, *maxspan*, H)
Input: $E(W\text{-}list)$: the extended group, *minsup*: the minimum support, *maxspan*: the maximum span, and H: the set of closed inter-sequence patterns.
Output: The hash table that stores current set of closed inter-sequence patterns H.
1. **If** $W\text{-}list = NULL$, **then** //join of 1-patterns
2. **For** each $X\text{-}list = \langle x_1^{\ 0}\rangle$ *-list* in $E(W\text{-}list)$, **do**
3. **For** each $Y\text{-}list = \langle y_1^{\ 0}\rangle$ *-list* in $E(W\text{-}list)$, **do**
4. **For** $d = 0$ to *maxspan*, **do**
5. **If** $d=0$, $x_1 < y_1$, and $sup(Z\text{-}list = X\text{-}list\ \theta_i\ Y\text{-}list) \geq minsup$, **then** add $Z\text{-}list$ to $E(X\text{-}list)$;
6. **If** $d=0$ and $sup(Z\text{-}list = X\text{-}list\ \theta_s\ Y\text{-}list) \geq minsup$, **then** add $Z\text{-}list$ to $E(X\text{-}list)$;
7. **If** $d>0$ and $sup(Z\text{-}list = X\text{-}list\ \theta_i^d\ Y\text{-}list) \geq minsup$, **then** add $Z\text{-}list$ to $E(X\text{-}list)$;
8. **End for**
9. **End for**
10. Call Join($E(X\text{-}list)$, *minsup*, *maxspan*, H);
11. **End for**
12. **Else** //join of k-patterns for $k > 1$
13. **If** W can not meet Lemma 1 and Lemma 2, **then** add W to H; //use closure checking
14. **If** W meets Lemma 4, **then** set $E(W\text{-}list) = NULL$; //use Hash table pruning
15. **For** each $X\text{-}list = \langle x_1^{\ i1} x_2^{\ i2} \ldots x_k^{\ ik}\rangle$ *-list* in $E(W\text{-}list)$, **do**
16. **For** each $Y\text{-}list = \langle y_1^{\ j1} y_2^{\ j2} \ldots y_k^{\ jk}\rangle$ *-list* in $J(X\text{-}list)$, **do**
17. **If** $x_k < y_k$, $i_k = j_k$, and $sup(Z\text{-}list = X\text{-}list\ \theta_i\ Y\text{-}list) \geq minsup$, **then** add $Z\text{-}list$ to $E(X\text{-}list)$;
18. **If** $i_k = j_k$ and $sup(Z\text{-}list = X\text{-}list\ \theta_s\ Y\text{-}list) \geq minsup$, **then** add $Z\text{-}list$ to $E(X\text{-}list)$;
19. **If** $i_k < j_k$, $j_k - i_k = d$ and $sup(Z\text{-}list = X\text{-}list\ \theta_i^d\ Y\text{-}list) \geq minsup$, **then** add $Z\text{-}list$ to $E(X\text{-}list)$;
20. **If** Y meets Lemma 3, **then** set $J(Y\text{-}list) = NULL$ and $E(Y\text{-}list) = NULL$;
21. **End for**
22. Call Join($E(X\text{-}list)$, *minsup*, *maxspan*, H);
23. **End for**

Fig. 4. The Join procedure.

Algorithm: EISP-Miner(*D*, *minsup*, *maxspan*): Frequent Inter-Sequence Patterns Miner.
Input: A sequence database *D*, a minimum support *minsup*, and a minimum span *maxspan*.
Output: The complete set of frequent inter-sequence patterns *FP*.
1. Scan *D* to generate a set of all frequent 1-Plists, *E*(*NULL*), as the extended group of the root node of an ISP-tree *T*;
2. Call ISP-Join(*E*(*NULL*), *minsup*, *maxspan*, *FP*);
3. Output *FP*;

Fig. 5. The EISP-Miner algorithm.

Therefore, every inter-sequence pattern found by the CISP-Miner algorithm is frequent and closed.

Lemma 6. The CISP-Miner algorithm can find every closed inter-sequence pattern.

Proof. In step 1 of the CISP-Miner algorithm, we scan the sequence database once to find all frequent 1-patterns, thus the algorithm can find every frequent 1-pattern in the database. In the Join procedure, we use Properties 1, 2, 3 and 4 to join all joinable frequent *k*-Plists to form all frequent super $(k+1)$-patterns of *p*, $k \geq 1$. Once a frequent pattern is found, we use the Join and Hash table checking schemes in Lemmas 1 and 2 to eliminate non-closed patterns. Since the CISP-Miner algorithm can find every frequent pattern where non-closed ones are either filtered out by the Join and Hash table checking schemes in Lemmas 1 and 2 or pruned out by the Join and Hash table pruning schemes in Lemmas 3 and 4, it can find every closed inter-sequence pattern.

Theorem 1. The CISP-Miner algorithm finds all closed inter-sequence patterns in the sequence database.

Proof. By Lemma 5, every inter-sequence pattern found by the CISP-Miner algorithm is frequent and closed, and by Lemma 6, the algorithm can find every closed inter-sequence pattern. Therefore, we conclude that the CISP-Miner algorithm is sound and complete, so that it can find all closed inter-sequence patterns in the sequence database.

4. Performance evaluation

To evaluate the performance of the CISP-Miner algorithm, we compared it with the EISP-Miner algorithm (Wang and Lee, 2009) on synthetic and real datasets. All experiments were conducted on an IBM Compatible PC with an Intel® Core™ 2 Duo CPU 2.4 GHz, 2 GB main memory running on Microsoft Windows Vista Home Premium. Each algorithm was implemented using Microsoft Visual C++ 6.0.

4.1. The EISP-Miner algorithm

The compared EISP-Miner algorithm and the associated ISP-Join procedure are showed in Figs. 5 and 6, respectively. In this paper, we propose the join checking, hash table checking, join pruning, and hash table pruning in the CISP-Miner algorithm. Therefore, we can observe in Figs. 3–6 that the main difference between CISP-Miner and EISP-Miner algorithms is Lemmas 1–4 are not used in the latter algorithm.

4.2. Synthetic data

To generate the synthetic datasets, we use three phases in the generation of synthetic data. The method used by this study to generate synthetic transactions is similar to the one used in Agrawal and Srikant (1994), where the data generator tools can be downloaded in Zaki (2012). We first generate all potentially frequent itemsets, T_{item}. Let *L* be the number of potentially frequent itemsets, and *N* be the total number of distinct items. The length of each itemset

Procedure: ISP-Join(*E*(*W-list*), *minsup*, *maxspan*, *FP*)
Input: *E*(*W-list*): the extended group, *minsup*: the minimum support, *maxspan*: the maximum span, and *FP*: the set of frequent inter-sequence patterns.
Output: The current set of frequent inter-sequence patterns *FP*.
1. **If** *W-list* = *NULL*, **then** //join of 1-patterns
2. **For** each *X-list* = $\langle x_1^{\ 0} \rangle$ *-list* in *E*(*W-list*), **do**
3. **For** each *Y-list* = $\langle y_1^{\ 0} \rangle$ *-list* in *E*(*W-list*), **do**
4. **For** $d = 0$ to *maxspan*, **do**
5. **If** $d=0$, $x_1<y_1$, and *sup*(*Z-list*=*X-list* θ_i *Y-list*)≥*minsup*, **then** add *Z-list* to *E*(*X-list*);
6. **If** $d=0$ and *sup*(*Z-list*=*X-list* θ_s *Y-list*)≥*minsup*, **then** add *Z-list* to *E*(*X-list*);
7. **If** $d>0$ and *sup*(*Z-list*=*X-list* θ_i^d *Y-list*)≥*minsup*, **then** add *Z-list* to *E*(*X-list*);
8. **End for**
9. **End for**
10. Call ISP-Join(*E*(*X-list*), *minsup*, *maxspan*, *FP*);
11. **End for**
12. **Else** //join of *k*-patterns for $k > 1$
13. Add *W* to *FP*;
14. **For** each *X-list* = $\langle x_1^{\ i1} x_2^{\ i2} \ldots x_k^{\ ik} \rangle$ *-list* in *E*(*W-list*), **do**
15. **For** each *Y-list* = $\langle y_1^{\ j1} y_2^{\ j2} \ldots y_k^{\ jk} \rangle$ *-list* in *J*(*X-list*), **do**
16. **If** $x_k < y_k$, $i_k = j_k$, and *sup*(*Z-list*=*X-list* θ_i *Y-list*)≥*minsup*, **then** add *Z-list* to *E*(*X-list*);
17. **If** $i_k = j_k$ and *sup*(*Z-list*=*X-list* θ_s *Y-list*)≥*minsup*, **then** add *Z-list* to *E*(*X-list*);
18. **If** $i_k < j_k$, $j_k - i_k = d$ and *sup*(*Z-list*=*X-list* θ_i^d *Y-list*)≥*minsup*, **then** add *Z-list* to *E*(*X-list*);
19. **End for**
20. Call ISP-Join(*E*(*X-list*), *minsup*, *maxspan*, *FP*);
21. **End for**

Fig. 6. The ISP-Join procedure.

is a Poisson distribution with mean equal to *I*. Next, we generate *F* potentially frequent patterns, T_{e_seq}. When we generate each pattern, we need to determine the total number of itemsets in it. The number of itemsets is a Poisson distribution whose mean is equal to *S*. All the itemsets in the first potentially frequent pattern are selected from T_{item}., and every itemset is associated with a relative time in the range 0 to *W* (*maxspan*). Finally, from T_{e_seq}, we generate $|D|$ transactions, each of which contains a series of frequent patterns. The number of itemsets and the average length of the itemsets are Poisson distributions whose means are equal to *C* and *T*, respectively. We conduct several experiments with various parameter settings. In each experiment, we vary one parameter and set the others to their default values. We set $L = 25{,}000$, $|D| = 10{,}000$, $T = 2$, $C = 4$, $F = 5000$, $S = 6$, $I = 1.5$, $N = 1000$, and $W = 3$ as default values. Note that, in the experiments, the support of a pattern is defined as the fraction of megasequences in the database that contain the pattern.

4.3. Experiments on synthetic data

Fig. 7 shows the effect of minimum support on the execution time, where the support varies between 0.2% and 0.8%. The results

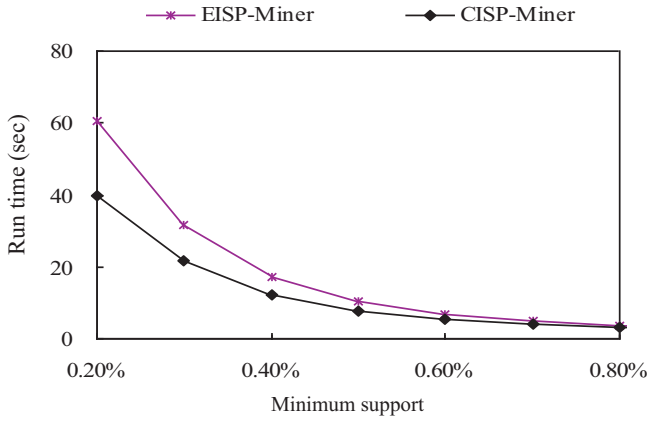

Fig. 7. The effect of minimum support on the execution time.

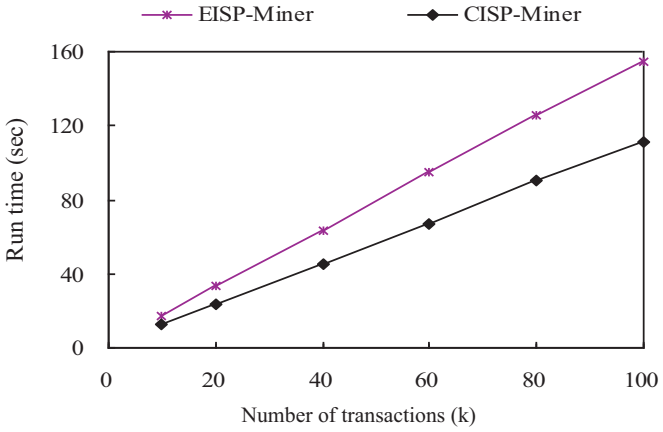

Fig. 8. The effect of the number of transactions on the execution time.

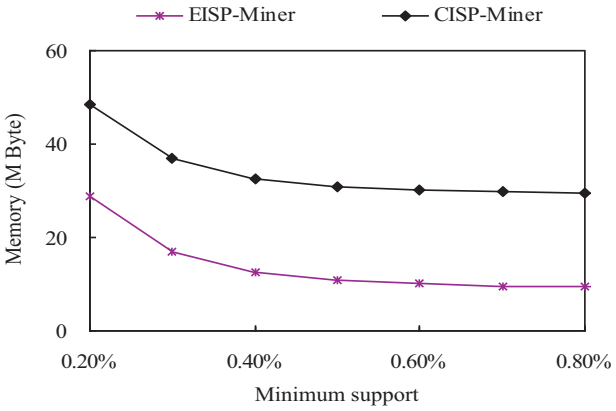

Fig. 10. The effect of minimum support on memory usage.

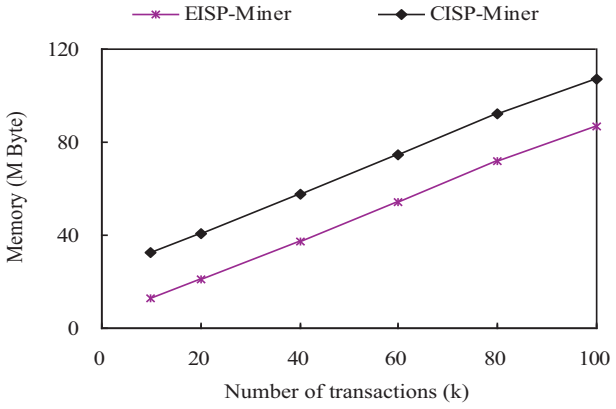

Fig. 11. The effect of the number of transactions on memory usage.

show that, as the minimum support threshold decreases, the execution time of the two algorithms increases; however, CISP-Miner runs faster than the EISP-Miner algorithms. This is due to the benefit derived from using the join operations so that CISP-Miner can localize a pattern's extension in a joinable group. Moreover, by using two pruning strategies, CISP-Miner can remove many unpromising Plists. As a result, the CISP-Miner algorithm runs 1–2 times faster than the EISP-Miner algorithm.

Fig. 8 shows the effect of the number of transactions on the execution time when the number of transactions varies between 1 K and 100 K. Although the execution times of the two algorithms increase almost linearly as the number of transactions increases, the CISP-Miner algorithm runs faster than the EISP-Miner algorithms in all cases.

In Fig. 9, we show the impact of *maxspan* on the performance of the algorithms. When *maxspan* increases, more candidates and patterns are generated, so the execution times of the two algorithms increase. However, the CISP-Miner is more efficient than the other algorithm. This is because it incorporates the concept of closed patterns and associated pruning strategies, so the time required for pattern discovery is reduced significantly.

Fig. 10 shows the memory usage as the minimum support increases from 0.2% to 0.8%; and Fig. 11 shows the memory usage as the number of transactions increases from 1 K to 100 K. Generally speaking, the CISP-Miner algorithm requires more memory than the EISP-Miner algorithm because it uses a hash table *H*, which occupies extra memory to store closed inter-sequence patterns.

4.4. Experiments on real data

We use a real dataset called Gazelle to evaluate the performance of the algorithms. The dataset was originally provided by Blue Martini which contains 29,369 customers' Web click-stream data. Therefore, this dataset contains 29,369 sequences and 1423 distinct items (Kohavi et al., 2000). We performed run time versus minimum support and run time versus *maxspan* experiments on the Gazelle dataset. The results, shown in Figs. 12 and 13, respectively, demonstrate that the CISP-Miner algorithm outperformed the EISP-Miner algorithms in all cases. Mining this dataset we discover many interesting inter-sequence patterns. For example, we find four frequent inter-sequence patterns with equal supports:

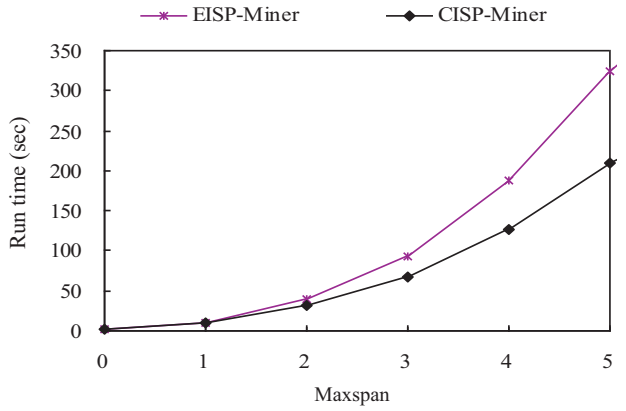

Fig. 9. The effect of *maxspan* on the execution time.

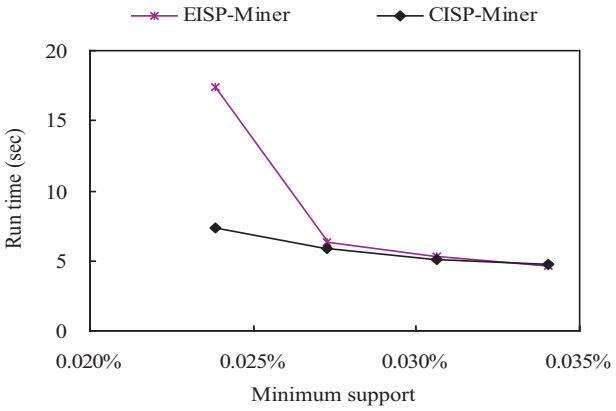

Fig. 12. Gazelle: the effect of minimum support on the execution time.

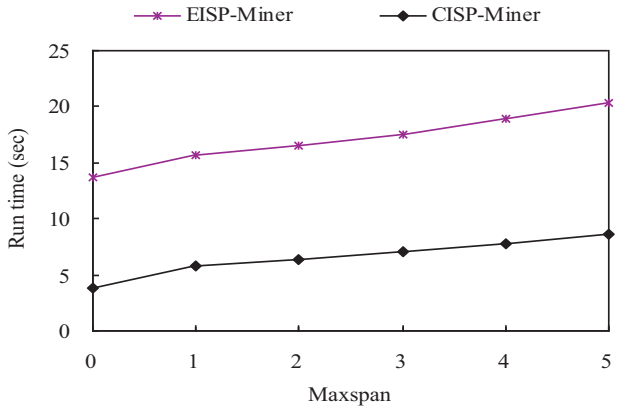

Fig. 13. Gazelle: the effect of *maxspan* on the execution time.

$\langle k^0e^0e^3\rangle$:5329, $\langle k^0f^0(ef)^3\rangle$:5329, $\langle(ef)^0(ef)^3\rangle$:5329, and $\langle k^0(ef)^0(ef)^3\rangle$:5329. Among them,$\langle k^0(ef)^0(ef)^3\rangle$is the only closed inter-sequence pattern and can be interpreted as: "if the items e and f followed by item k are bought in one day, then the items e and f are likely be bought again three days later". By this example, we illustrate the closed inter-sequence pattern mining can find meaningful and compact patterns in a sequence database.

In summary, since the CISP-Miner algorithm uses effective pruning strategies and closure checking schemes, it is more efficient than the EISP-Miner algorithms in most cases. The experiment results demonstrate that the incorporating of the idea of closed pattern is a feasible and effective solution for the inter-sequence pattern mining problem.

5. Conclusion

The proposed CISP-Miner algorithm mines all closed inter-sequence patterns so that a pattern can be used to describe associations across many different sequences. The algorithm searches for frequent inter-sequence patterns recursively in a depth-first search manner. Since CISP-Miner localizes pattern extensions in joinable group and uses several effective pruning strategies and closure checking schemes to reduce the search space and thus accelerate the algorithm, it is more efficient and scalable than the EISP-Miner algorithm. The experiment results demonstrate that the CISP-Miner algorithm outperforms the compared algorithm in most cases.

Although we have shown that the CISP-Miner algorithm can mine closed inter-sequence patterns efficiently, there are still some issues to be addressed in future research. First, we could extend the algorithm to mine generalized inter-sequence patterns (Li et al., 2005), which expand rule contexts from point-wise (e.g. two days later) to scope-wise contexts (e.g. within three days). We could also extend the algorithm from mining a sequence database with one domain attribute to mining multiple domain attributes. Finally, the CISP-Miner algorithm uses a hash table to store in memory all closed patterns that have been mined so far. If the number of inter-sequence patterns is too large, the hash table will consume considerable memory space and the performance of CISP-Miner might be degraded. Therefore, in the future, we will device a new mechanism to mine closed inter-sequence patterns without using the hash table to avoid such limitation.

References

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of 1994 Int. Conf. on Very Large Data Bases (VLDB'94), pp. 487–499.

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

Chiu, S.Y., Chiu, S.C., Huang, J.L., 2009. On mining repeating pattern with gap constraint. In: Proceedings of 11th IEEE International Conference on High Performance Computing and Communications, pp. 557–562.

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

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

Hu, Y., Panda, B., 2010. Mining inter-transaction data dependencies for database intrusion detection. Innovations and Advances in Computer Sciences and Engineering, 67–72.

Kohavi, R., Brodley, C., Frasca, B., Mason, L., Zheng, Z., 2000. KDD-Cup 2000 organizers' report: peeling the onion. SIGKDD Explorations 2, 86–98, Web site: http://www.kdd.org/kdd-cup-2000-online-retailer-website-clickstream-analysis (last accessed October 2012).

Lee, A.J.T., Hong, R.W., Ko, W.M., Tsao, W.K., Lin, H.H., 2007. Mining spatial association rules in image databases. Information Sciences 177 (17), 1593–1608.

Lee, A.J.T., Lin, W.C., Wang, C.S., 2006. Mining association rules with multi-dimensional constraints. The Journal of Systems and Software 79 (1), 79–92.

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

Lee, A.J.T., Wang, C.S., Weng, W.Y., Chen, Y.A., Wu, H.W., 2008. An efficient algorithm for mining closed inter-transaction itemsets. Data and Knowledge Engineering 66 (1), 68–91.

Lee, A.J.T., Wang, Y.T., 2003. Efficient data mining for calling path patterns in GSM networks. Information Systems 28 (8), 929–948.

Li, Q., Feng, L., Wong, A., 2005. From intra-transaction to generalized inter-transaction: landscaping multidimensional contexts in association rule mining. Information Sciences 172 (3-4), 361–395.

Liu, H., Wang, X., He, J., Han, J., Xin, D., Shao, Z., 2009. Top-down mining of frequent closed patterns from very high dimensional data. Information Sciences 179 (7), 899–924.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C., 2004. Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Transactions on Knowledge and Data Engineering 16 (10), 1424–1440.

Tung, A.K.H., Lu, H., Han, J., Feng, L., 2003. Efficient mining of intertransaction association rules. IEEE Transactions on Knowledge and Data Engineering 15 (1), 43–56.

Wang, C.S., Lee, A.J.T., 2009. Mining inter-sequence patterns. Expert Systems with Applications 36 (4), 8649–8658.

Wang, J., Han, J., Li, C., 2007. Frequent closed sequence mining without candidate maintenance. IEEE Transactions on Knowledge and Data Engineering 19 (8), 1042–1056.

Wu, H.W., Lee, A.J.T., 2010. Mining closed flexible patterns in time-series databases. Expert Systems with Applications 37 (3), 2098–2107.

Yan, X., Han, J., Afshar, R., 2003. CloSpan: mining closed sequential patterns in large datasets. In: Proceedings of SIAM Conference on Data Mining, pp. 166–177.

Yang, W., Dong, C., Cheng, J., Fang, F., 2009. The research into an improved algorithm of telecommunication inter-transactional association rules based on time series of all confidence. In: Proceedings of IEEE Symposium on Industrial Electronics and Applications, pp. 192–196.

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

Zaki, M.J., 2012. Software: Real and Synthetic Datasets, Web site: http://www.cs.rpi.edu/~zaki/ (last accessed October 2012).

Zaki, M.J., Hsiao, C.J., 2005. Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Transactions on Knowledge and Data Engineering 17 (4), 462–478.

Chun-Sheng Wang received an MBA and a PhD degree in information management from National Taiwan University, Taiwan, ROC, in 1996 and 2007, respectively. He is currently an assistant professor of information management at the Jinwen University of Science and Technology, Taiwan, ROC. His papers have appeared in *Journal of Systems and Software*, *Information Sciences*, *Data and Knowledge Engineering*, *Expert Systems with Applications*, *Journal of Network and Systems Management*, etc. His current research interests include data mining, information systems, and network management.

Ying-Ho Liu received the BS, MS, and PhD degrees in information management from National Taiwan University, Taiwan, in 2000, 2003, and 2009, respectively. In August 2009, he joined the department of information management at National Dong Hwa University and he is now an assistant professor. His papers have appeared in *Data and Knowledge Engineering*, *Journal of Systems and Software*, and *Pattern Recognition*. His research interest includes multimedia content analysis and indexing, data mining, machine learning, and pattern recognition.

Kuo-Chung Chu received his PhD degree in information management from the National Taiwan University in 2005. In 1992, he joined the computer center, Academia Sinica, Taiwan, where he was responsible for network systems management; followed by the Faculty of Information Management, Jinwen University of Science and Technology, Taipei, and as a department chair from 2006 to 2007. Since 2007, he has been with the Department of Information Management, National Taipei College of Nursing (NTCN), Taiwan. He is currently an associate professor in that department, and as a director of computer center at NTCN.

He received the Dragon Thesis Award, Gold Medal, by the Acer Foundation in 2005, and the Excellent Practical Dissertation Award, by the Industrial Development Bureau/MOEA and Chinese Society of Information Management of Taiwan in 2006, and the Excellent Paper Award, by the Operations Research Society of Taiwan in 2006. His papers have appeared in *Annals of Telecommunications*, *Computers and Electrical Engineering*, *International Journal of Information and Management Sciences*, *Journal of Network and Systems Management, Telecommunication Systems*, etc. His research interests include data mining, decision modeling, optimization approach, simulation, and their applications on healthcare and network management.