**METISP: Memory Time-Indexing Sequential Pattern mining**

We have developed a new algorithm, called METISP, for time-constrained sequential pattern mining. METISP algorithm is the first algorithm which can solve all time constraints including minimum/maximum/exact gap, sliding window and duration constraints at the same time. The novel algorithm utilizes the technique of memory indexing to mark the timestamps. Hence, the inherent properties of time constraints can be applied to reduce the search space and improve the efficiency of mining time-constrained sequential patterns greatly. For extra-large database, the algorithm applies the idea of partition-and-verification for mining sequential patterns with time constraints.

The problem is formulated in Section 3.1. Section 3.2 introduces the terminology used in this chapter. The details of METISP algorithm is described in Section 3.3. An example of mining using METISP is illustrated in Section 3.4. For convenience, we refer to the sequence database as *DB* and a data sequence as *ds* in the following context.

**3.1** **Problem Statement**

Let *Ψ*= {*α*1, *α*2, …, *αr*} be a set of literals, called *items*. An *itemset* *I*= (*β*1, *β*2, …, *βq*) is a nonempty set of *q* *items* such that *I* ⊆ *Ψ*. A *sequence* *s*, denoted by <*e*1*e*2…*ew*>, is an ordered list of *w elements* where each element *ei* is an *itemset*. Without loss of generality, we assume the items in an element are in lexicographic order. The *length* of a *sequence s*, written as |*s*|, is the total number of *items* in all the *elements* in *s*. Sequence *s* is a *k*-*sequence* if |s| = *k*. The sequence database *DB* contains |DB| data sequences. A *data sequence* *ds* has a unique identifier *sid* and is represented by <*t*1*e*1’ *t*2*e*2’ …*tn* *en*’>, where element *ei*’ occurred at time *ti* , *t*1< *t*2< ...< *tn*.

A user gives a parameter *minsup* (minimum support) and four time-constraints *maxgap* (maximum gap), *mingap*(minimum gap), *swin*(sliding window) and *duration* to discover the set of all time-constrained sequential patterns. A sequence *s* is a *time-constrained sequential pattern* if *s.sup* ≥ *minsup*, where *s.sup* is the *support* of the sequence *s* and *minsup* is the user specified minimum support threshold. The *support* of *s* is the number of data sequences *containing s* divided by |*DB*|. A data

sequence *ds*= <*t*1*e*1’*t*2*e*2’…*tnen*’> *contains* a sequence *s* = <*e*1*e*2…*ew*> if there exist integers *l*1, *u*1, *l*2,*u*2, …,*lw*, *uw* and

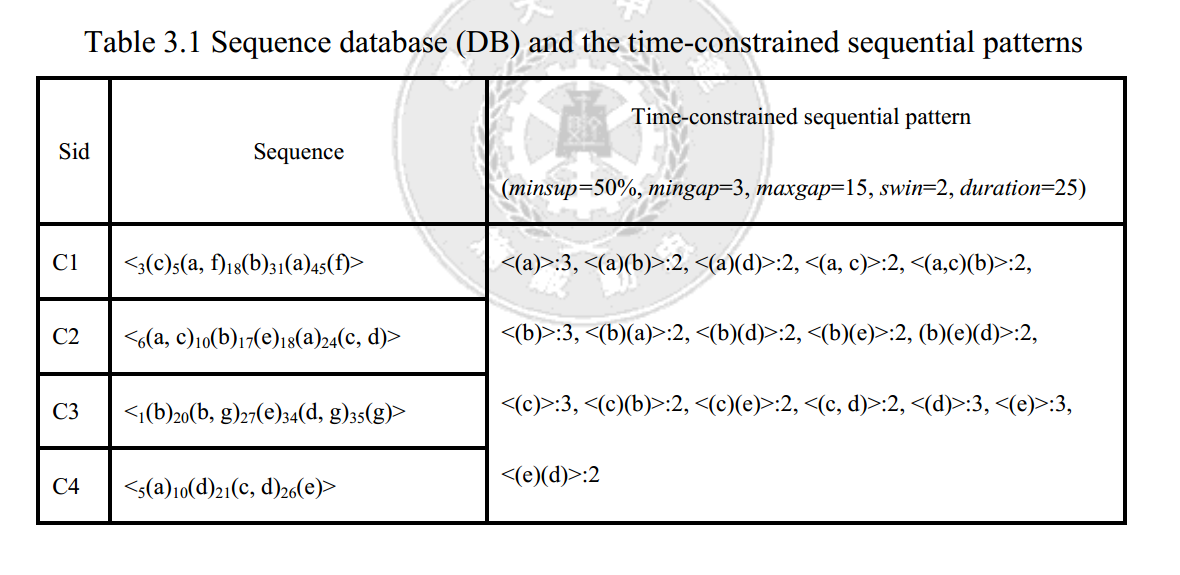
1 ≤ *l*1 ≤ *u*1< *l*2 ≤ *u*2< …< *lw* ≤ *uw* ≤ *n* such that the five conditions hold: (1) ei⊆(*eli’a*..*eui’*), 1 ≤ i ≤ *w*(2) *tui*− *tli*≤ *swin*,

1 ≤ *i* ≤ *w*(3)*tui*−*tli*-1≤ maxgap, 2 ≤ *i* ≤ *w* (4)*tli*−*tui*-1≥ mingap, 2 ≤ *i* ≤ *w* (5) *tuw*–*tl*1 ≤ duration.

Assume that *tj, mingap, maxgap, swin*, and *duration* are all positive integers, *maxgap* ≥ *mingap* ≥1. Common sequential pattern mining without time constraints is a special case by setting *mingap*= 1, *maxgap*= ∞, *swin*= 0 and *duration*= ∞. When *mingap* is the same as *maxgap*, the time gap is called *exact gap*.

For example, given a sequence *DB* of four data sequences with their *sids* in Table 3.1, the rightmost column shows the time-constrained sequential patterns with constraints *mingap*=3, *maxgap*=15, *swin*=2 and *duration*=25. In the *DB*, data sequence C4 <5(a)10(d)21(c, d)26(e)> has four itemsets occurring at time 5, 10, 21 and 26, respectively. The third itemset of C4 has two items *c* and *d*. Sequences <(a, c)(b)> and <(b)(e)(d)> are both 3-sequences. The sequence <(a, c)(b)> is contained in data sequences C1 <3(c)5(a, f)18(b)31(a)45(f)> because element (a, c) can be contained in the transaction combining 3(c) and 5(a, f) for 5−3>=2 (swin). Meanwhile, the other constraints can be satisfied for 18−5>=3 (mingap), 18−3<=15(maxgap) and total time span 18−3<=25 (duration). Similarly, <(a, c)(b)> is contained in C2. The support of <(a, c)(b)> is 2/4 and it is a time-constrained sequential pattern for *minsup*= 50%. Considering sequence <(a)(b)(a)>, it can be contained in C1 for the other constraints but it fails the *duration* constraint (31−5>=25). Given *mingap* = *maxgap* = 7, that is, exact gap = 7, C3 <1(b)20(b, g)27(e)34(d, g)35(g)> contains pattern<(b)(e)(d)> but it does not contain pattern <(b)(d)> because 34−20≠7 (exact gap).

Table 3.1 Sequence database (DB) and the time-constrained sequential patterns



**3.2 Terminology Used in the Proposed Algorithm**

**Definition 1 (frequent item)** An item *x* is called a *frequent item* in *DB* if <(*x*)>*.sup* ≥ *minsup*.

**Definition 2: (type-1 pattern, type-2 pattern, stem, prefix)** Given a frequent pattern *P* and a frequent item *x* in *DB*, *P ’*is a *type-1 pattern* if it can be formed by adding an itemset of the single item *x* after the last element of *P*, and a *type-2 pattern* if formed by appending *x* to the last element of *P*. The item *x* is called the *stem* of the new frequent pattern *P’*. The *prefix pattern*(abbreviated as *prefix*) of *P ’*is *P*. For example, <(a)>, <(a)(b)>, <(a)(d)> and <(a, c)> in Table 3.1 are the discovered time-constrained sequential patterns. <(a)(b)> and <(a)(d)> are type-1 patterns by adding (b) and (d) after (a) respectively and <(a, c)> is a type-2 pattern by appending (c) to (a). Here, (b), (d) and (c) are the stems and <(a)> is their prefix.

**Definition 3: (initial time, last-start time, last-end time, time index)** Let the first element of a frequent pattern *P* be *FE* and last element be *LE*. If *ds* contains *P* by having *FE* ⊆ *e*δ∪ *eδ*+1∪…∪*eε* and *LE* ⊆ *e**γ* ∪*eγ*+1∪…∪*eω*, where *eδ*,…, *eω* are element in *ds*, the occurring time *tδ* , *tγ* , and *tω* for itemsets *eδ,* *e*γ and *eω*, are named *initial time*(abbreviated as it), *last-start time*(abbreviated as lst) and *last-end time*(abbreviated as let) of *P* in *ds*. The it:lst:let is marked as *eδ* :*e*γ:*e*ω. Every occurrence

of timestamp it:lst:let is collected as [it1:lst1:let1, it2:lst2:let2,…, itk:lstk:letk], iti≤lsti≤leti for 1≤i≤k. Such a timestamp lists is called the *time index* of *P* in *ds*.

For example, given a sequence database as shown in Table 3.1, minimum support=50%, mingap=5, maxgap=16, swin=0, and duration=30. <(a)(d)> is a sequential pattern contained by C2 and C4. For C2 <6(a, c)10(b)17(e)18(a)24(c, d)>, the time stamp is marked as 18:24:24. The initial time represents (a) of <(a)(d)> occurs at time 18. The last-start time and last-end time represents (d) of <(a)(d)> occurs at time 24. Similarly, for C4 <5(a)10(d)21(c, d)26(e)>, <(a)(d)> appears two times and the time index of <(a)(d)> is collected as [5:10:10, 5:21:21].

Definition 4: (valid time periods) Given a time index of *P* in *ds*, the time periods of the itemsets in *ds* to be used for finding a potential stem *x* of pattern *P’*, where *P* is the prefix of *P’*, are called *valid time periods* (abbreviated as VTPs).

**Lemma 1**: Given a time index of *P* in *ds* [it1:lst1:let1, it2:lst2:let2,…, itk:lstk:letk], the valid time period to form a type-1 pattern must satisfy either one of the following conditions, as illustrated in Figure 3.1:

leti+mingap ≤ VTP ≤ min{lsti+maxgap, iti+duration}, ∃ *i* , 1<=*i* <=*k*

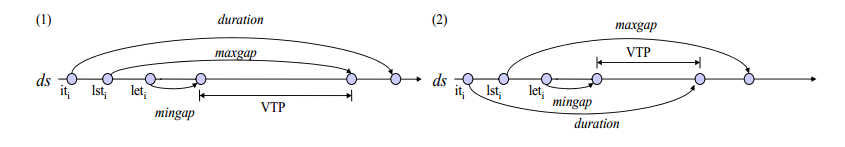


Figure 3.1 Valid time periods of a potential type-1 pattern (Type-1 VTPs)

For example, in the rightmost column of Table 3.1, the time-constrained sequential pattern <(b)(e)> is contained in C2 and C3. For C2 <6(a, c)10(b)17(e)18(a)24(c, d)>, the time index is marked as [10:17:17]. For the type-1 patterns, the VTP can be obtained as 17+3 (mingap) to 17+15 (maxgap) because the duration limitation of 10+25 (duration) is larger than the maxgap limitation of 17 +15(maxgap). Similarly, for C3, a stem (d) can be found within those time periods and <(b)(e)(d)> is a newly discovered pattern.

**Lemma 2**: Given a time index of *P* in *ds* [it1:lst1:let1, it2:lst2:let2,…, itk:lstk:letk], the valid time period to form a type-2 pattern must satisfy either one of the following conditions, as illustrated in Figure 3.2:

leti−swin ≤ VTP ≤min{lsti+swin, iti+duration}, ∃ *i* , 1<=*i* <=*k*

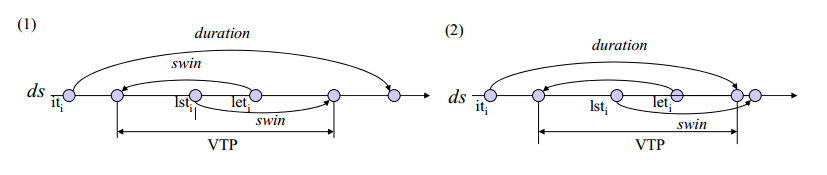


Figure 3.2 Valid time periods of a potential type-2 pattern (Type-2 VTPs)

Figure 3.2 shows the two conditions in Lemma 2. Note that the potential stem found using Lemma 2 can be pruned if it is lexicographically smaller than the items of the last element in *P*. For any type-2 pattern, the valid time periods are checked on not violating the minimum/maximum gap constraints between the last two elements. In case *P* has only one element and the VTP to be used is earlier than sti, then *duration* constraint must be checked additionally.

For instance, the VTP of the type-2 pattern <(b)(e)> can be obtained as 17−2(swin) to 17+2(swin) for C2 <6(a, c)10(b)17(e)18(a)24(c, d)>. The (a) appears at time 18, the support count therefore cannot be increased since the lexicographical order of (a) is small than (e).

**3.3 METISP Algorithm: A Detailed Description**

Figure 3.3 outlines the METISP Algorithm. METISP mines patters within the pattern-growth framework similar to the pseudo projection version of PrefixSpan algorithm [25], while it can handle constraints minimum/maximum gaps, duration and sliding time-window. Assume that the DB can fit into the main memory, METISP first loads DB into memory (as MDB) and scans MDB once to find all frequent items. With respect to each frequent item, METISP then constructs a time index-set for the 1-sequence item and recursively forms time-constrained sequential patterns of longer length. The time index-set is a set of (data-sequence pointer, time index) pairs. Only those data sequences containing that item would be included. The time index indicates the list of it:lst:let triplets as described in Definition 3.

In Figure 3.4 Mine (P, P-TIdx) mines type-1 patterns and type-2 patterns having prefix P by effectively locating VTPs using Lemma 1 and Lemma 2, respectively. The P-TIdx is the time index-set for P. METISP thus never search the data sequences

irrelevant to P. Moreover, the VTPs ensure that METISP locates and counts the effective stems which can form valid patterns, rather than the whole set of items in the data sequence. Likewise, for a newly formed type-1 pattern or type-2 pattern P’, its

time index-set P’-TIdx is constructed and Mine(P’, P’-TIdx) is invoked. By pushing time attributes deeply into the mining process, METISP efficiently discovers the desired patterns.

Algorithm: METISP

Input: *DB*(a sequence database),*minsup* (minimum support), *mingap*(minimum gap),

*maxgap* (maximum gap), *swin* (sliding time-window), *duration*(duration)

Output: the set of all time-constrained sequential patterns

1. load *DB* into memory (as MDB) and scan MDB once to find all frequent items.

2. for each frequent item *x*,

(1) form the sequential pattern P = <(*x*)> and output P.

(2) scan MDB once to construct P-Tidx, time index set of *x*.

(3) call Mine(P, P-Tidx)

Figure 3.3 Algorithm METISP

**Theorem 1**: All the sequential patterns can be divided into type-1 patterns and type-2 patterns. With the time constraints, a stem and a prefix must satisfy all time restrictions in data sequences to form a new type-1 or type-2 pattern. METISP mines all patterns by type-1 and type-2 pattern growth recursively. For each prefix, METISP counts the supports of items in all VTPs which have satisfied all time constraints as that all stems are found in the VTPs of type-1and type-2 patterns to form type-1 and type-2 patterns, respectively. Hence, METISP mines completely time-constrained sequential patterns finally.

Subroutine: Mine (P, P-TIdx)

Parameter: P = prefix pattern, P-Tidx = time index set

1. for each data sequence *ds* in the P-DB, // P-DB: sequences indicated in P-TIdx

(1) use the corresponding time index to collect the type-1 VTPs satisfying:

leti+mingap ≤ VTP ≤ min{lsti+maxgap, iti+duration},∃ *i* , 1≤*i* ≤*k*

(2) for each item in the VTPs of type-1 pattern, add one to its support count.

(3) use the corresponding time index to collect the type-2 VTPs satisfying:

leti−swin ≤ VTP ≤min{lsti+swin, iti+duration},∃ i , 1≤i ≤k

(4) for each item in the VTPs of type-2 pattern, add one to its support count

2. for each item x’ found in the VTPs of type-1 pattern and its support is greater than or equal to *minsup*,

(1) form the type-1 pattern *P’* by extending stem *x’* and output *P’*.

(2) scan the VTPs of each *ds* in P-DB to construct P’-Tidx, time index set of *x’*.

// in this, the VTPs must satisfy the formulas indicated in Lemma 1

(3) call Mine(P’, P’-TIdx);

3. for each item *x’* found in the VTPs of type-2 pattern and its support is greater than or equal to *minsup*,

(1) form the type-2 pattern *P’* by appending stem *x’* and output *P’*.

(2) scan the VTPs of each *ds* in P-DB to construct P’-Tidx, time index set of *x’*.

// in this, the VTPs must satisfy the formulas indicated in Lemma 2

(3) call Mine(P’, P’-TIdx);

Figure 3.4 Subroutine Mine() of METISP algorithm

**3.4 An Illustrating Example ofMining Patterns Using METISP**

Example: Given a DB in Table 3.1, minsup = 50%,mingap = 3, maxgap=15, swin=2 and duration=25, METISP mines sequential patterns by the following steps.

*Step 1: Load DB into memoryand find all frequent items.*

METISP first reads the DB into memory and scans the in-memory DB (abbreviated as MDB) once to find frequent items. <(a)>:3, <(b)>:3, <(c)>:3, <(d)>:3,<(e)>:3 are found, <(a)>:3 shows its support count of 3. For each frequent item, METISP uses step 2 to construct the Time Index Set.

*Step 2: Construct the Time Index Set of frequent items*

For each frequent item, METISP scans MDB once to construct the time index set which contains the time indexes and the sequence pointers where the item appears. Take <(a)> for example, METISP scans MDB and constructs the time index set

<(a)>-Tidx, as shown in Figure 3.5(a). Three pointers appear in <(a)>-Tidx since <(a)> appears in C1, C2, and C4. For C1 <3(c)5(a, f)18(b)31(a)45(f)>, element (a) occurs at time 5 and 31 so the time index is marked as [5:5:5, 31:31:31]. The indexes for C2 and C4 are processed accordingly. The <(a)>-Tidx is used in subroutine Mine, which searches potential type-1and type-2 stems.

*Step 3: Find stems in VTPs from the time index set and grow(discover) patterns (type-1 and type-2 stems of <(a)>)*

For each sequence containing the prefix P, METISP uses the time index to find out the valid time periods of P for type-1 patterns and type-2 patterns and count the supports of all items within the valid time periods respectively. METISP finds all the stems quickly and growths type-1 patterns and type-2 patterns. Subroutine Mine computes the supports of potential stems here. With respect to prefix <(a)>, the type-1 VTPs for C1 is [8:20, 34:46] from [lst1+mingap:let1+maxgap, lst2+mingap:let2+maxgap], where lst1 = let1 = 5 and lst2 = let2 = 31. The VTPs of C2 [9:21, 21:33] and C4 [8:20] are obtained similarly. Likewise, the type-2 VTPs for C1 is [3:7, 29:33] from [lst1+swin:let1-swin, lst2+swin:let2-swin], for C2 is [4:8, 16:20], and for C4 is [3:7]. Now, the supports of items *b* and *d* pass the threshold to be new type-1 stems and items *c* has enough support to form a new type-2 stem. Thus, <(a)(b)> (and later <(a)(d)>) is outputted to form two new type-1 patterns and <(a, c)> is outputted to form a new type-2 pattern. METISP further uses step 4 to construct the time index set of <(a)(b)> and grow pattern in subroutine Mine. Then <(a)(b)>, <(a)(d)> and <(a, c)> are processed in turn using the same steps.

*Step 4: Construct the time index set of the sequential pattern and discover all sequential patterns recursively.*

For each sequential pattern P’ with prefix P and stem *x*, METISP scans each data sequence containing P and records all the initial time, last-start time and last-end time of P’ within the valid time periods of P. METISP also labels sequence pointers *s\_ptrs* containing P’ simultaneously. Thus, < (a) (b)>-Tidx is constructed, and Mine is called recursively. The <(a)(b)>-Tidx is shown in Figure 3.5(b). Considering type-1 VTPs of <(a)(b)>, which are [21:33] for C1 and [13:25] for C2, no more stems can be found. No pattern of prefix <(a)(b)> can be formed. Similarly, type-2 VTPs of <(a)(b)> is computed to find potential type-2 stems.

With respect to prefix < (a)(b)>, the type-2 VTPs for C1 is [16:20, 8:12] from [let1−swin:lst1+swin, let2-swin:lst2+swin], where lst1 = let1 = 18 and lst2 = let2 = 10. No pattern of prefix < (a)(b)> can be formed, the process returns to work on <(a)(d)>. Unfortunately, no stems can be found within both type-1 and type-2 VTPs of < (a)(d)>. The process returns to work on the type-1 and type-2 stems of < (a, c)>.

With respect to prefix < (a, c)>, < (a, c)>-Tidx is constructed. Note that in Figure 3.5(d), the initial time reflects the *swin* effect so that the time index for C1 is [3:3:5]. Subroutine Mine is then called recursively. Referring to < (a, c)>-Tidx, the type-1 VTPs for C1 is [8:18] from [5+mingap: 3+maxgap], for C2 is [6+mingap: 6+maxgap].

Subroutine Mine finds item *b*, outputs < (a, c) (b)>. For the type-2 VTPs of < (a, c)>, Mine cannot form any type-2 pattern and only < (a, c) (b)>-Tidx is constructed finally. However, no more patterns are formed and the mining of prefix < (a)> now stops. METISP then applies steps 2 to 4 on < (b)>, <(c)>, < (d)>, and < (e)>. The complete set of time-constrained patterns is listed in the rightmost column in Table 3.1.

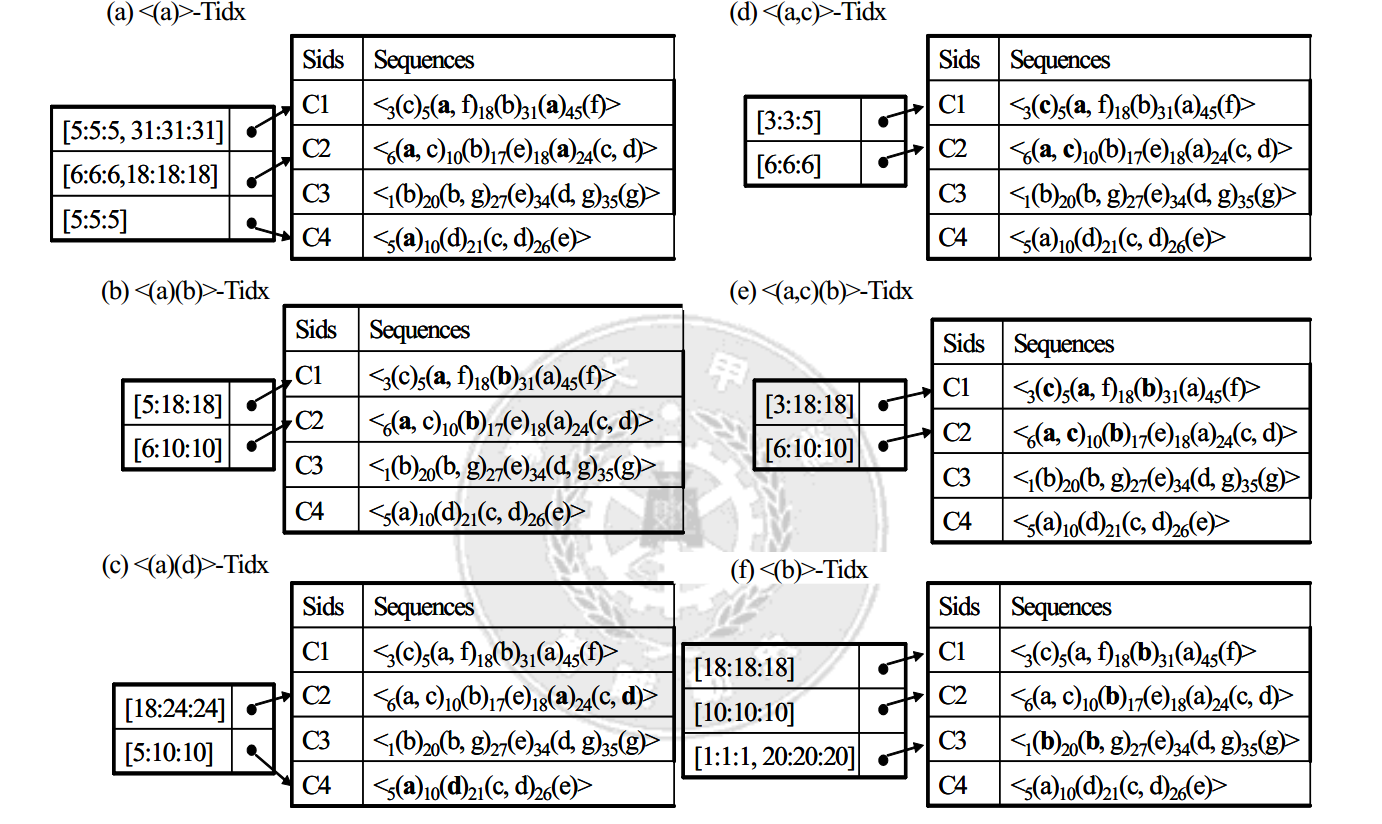


Figure 3.5 Some time index sets of sequential patterns