**METISP: Khai phá chuỗi dữ liệu với ràng buộc thời gian bằng cách đánh chỉ số thời gian bộ nhớ.**

METISP là thuật toán đầu tiên, gồm ít tính toán nhất (được biết) thực hiện việc khai phá dữ liệu chuỗi với ràng buộc về thời gian. Các ràng buộc thời gian đó là : minimum/maximum/exact gap, sliding window and duration . Chúng được dùng để giảm bớt không gian tìm kiếm và nâng cao hiệu quả của việc khai phá dữ liệu chuỗi. Thuật toán sử dụng kĩ thuật đánh chỉ số cho bộ nhớ (memory indexing) để dánh dấu nhãn thời gian( timestamps)

Kí hiệu cơ sở dữ liệu chuỗi (sequence database) là *DB* và một chuỗi trong DB là *ds* .

**3.1** **Kiến** thức nền tảng

Cho *Ψ*= {*α*1, *α*2, …, *αr*} là tập hợp toàn bộ các *item*  được sắp xếp theo thứ tự bảng chữ cái.

Một tập hợp các item gọi là  *itemset* *I*= (*β*1, *β*2, …, *βq*) bao gồm *q* item với *I* ⊆ *Ψ*.

Một chuỗi ( *sequence* ) *s*, biễu diễn là s = <*e*1*e*2…*ew*>, là một danh sách gồm *w* phần tử*( element)* trong đó mỗi phần tử *ei* là 1 *itemset*. Độ dài của chuỗi *s*, kí hiệu |*s*|, là số lượng *items* nằm trong *s*. Ví dụ chuỗi *k*-*sequence* có |s| = *k* hay gồm *k* item.

Một cơ sở dữ liệu chuỗi (sequence database) *DB* chứa |DB| chuỗi dữ liệu. Một chuỗi dữ liệu *ds* có định danh *sid* và được biểu diễn bởi <*t*1*e*1’ *t*2*e*2’ …*tn* *en*’>, với mỗi phần tử *ei*’ xảy ra tại thời gian *ti* , *t*1< *t*2< ...< *tn*.

Bài toán cần giải quyết:

Người dùng cho trước ngưỡng hỗ trợ (mức hỗ trợ nhỏ nhất) *minsup* (minimum support) và 4 ràng buộc thời gian *maxgap* (maximum gap), *mingap*(minimum gap), *swin*(sliding window) and *duration* để khai phá dữ liệu chuỗi.

1 chuỗi *s* gọi là 1 *time-constrained sequential pattern(pattern chuỗi* ràng buộc thời gian) nếu nó có độ hỗ trợ không nhỏ hơn minsup *hay s.sup* ≥ *minsup*, với *s.sup* là độ hỗ trợ của s và *minsup* là ngưỡng hỗ trợ nhỏ nhất cho trước.

Độ hỗ trợ của *s* tính bằng số lượng chuỗi dữ liệu *ds* chứa *s* chia cho tổng số chuỗi dữ liệu |*DB*| trong cơ sở dữ liệu.

Một chuỗi dữ liệu (data sequence) *ds*= <*t*1*e*1’*t*2*e*2’…*tnen*’> được cho là chứa chuỗi (sequence) *s* = <*e*1*e*2…*ew*> nếu tồn tại các số nguyên *l*1, *u*1, *l*2,*u*2, …,*lw*, *uw* với 1 ≤ *l*1 ≤ *u*1< *l*2 ≤ *u*2< …< *lw* ≤ *uw* ≤ *n* và thỏa mãn 5 điều kiện sau:

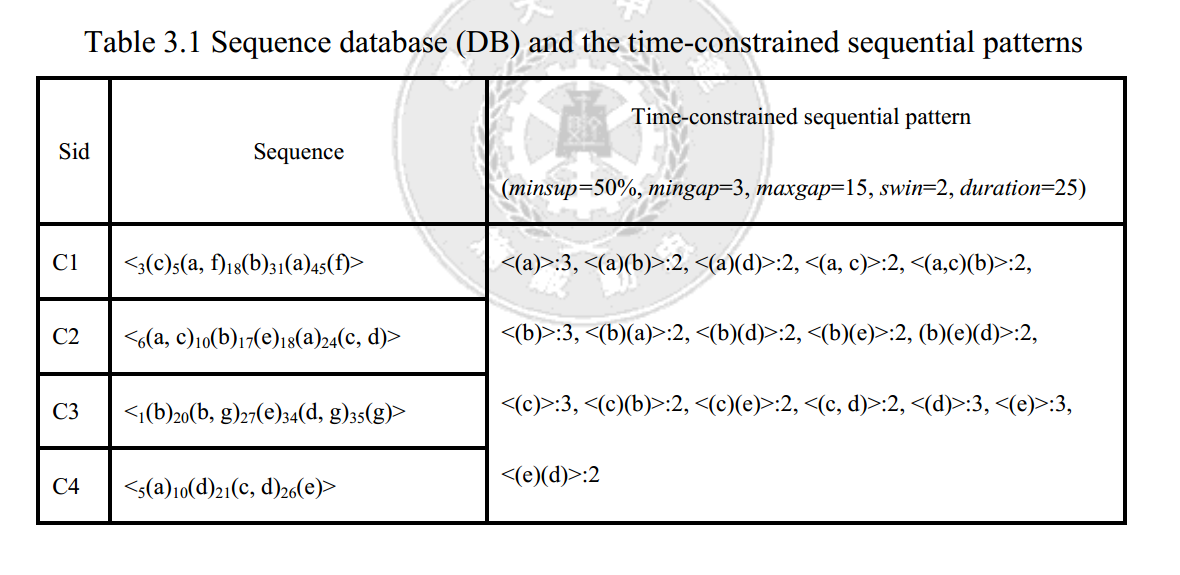
(1) ei⊆(*eli’a*..*a eui’*), 1 ≤ i ≤ *w*(2) *tui*− *tli*≤ *swin*, 1 ≤ *i* ≤ *w*(3)*tui*−*tli*-1≤ maxgap, 2 ≤ *i* ≤ *w* (4)*tli*−*tui*-1≥ mingap, 2 ≤ *i* ≤ *w* (5) *tuw* – *tl*1 ≤ duration.

Giả sử *tj, mingap, maxgap, swin*, and *duration* là các số nguyên dương, *maxgap* ≥ *mingap* ≥1.

Trường hợp khai phá không có ràng buộc thời gian cũng có nghĩa là ta gán *mingap*= 1, *maxgap* = ∞, *swin* = 0, *duration* = ∞.

Nếu *mingap* = *maxgap*, thì ta gọi là *exact gap*.(khoảng thời gian chính xác)

Xét ví dụ :



Đề bài cho *minsup*=50%, *mingap*=3, *maxgap*=15, *swin*=2 and *duration*=25.

Trong *DB*, data sequence C4 <5(a)10(d)21(c, d)26(e)> gồm 4 itemsets xảy ra ở thời gian 5, 10, 21 and 26; itemset thứ 3 of C4 có 2 items *c* và *d*. Các chuỗi < (a, c) (b)> and <(b)(e)(d)> là 3-sequences(chuỗi có 3 phần tử). Chuỗi <(a, c)(b)> nằm trong

C1 <3(c)5(a, f)18(b)31(a)45(f)>vì element (a, c) nằm trong transaction (giao dịch) kết hợp giữa 3(c) và 5(a, f) do 5−3>=2 (swin). Ta cũng có 18−5>=3 (mingap), 18−3<=15(maxgap) and total time span 18−3<=25 (duration). Tương tự, <(a, c)(b)> nằm trong C2. Độ hỗ trợ <(a, c)(b)> là 2/4 do đó nó là 1 time-constrained sequential pattern vì >= *minsup*(50%). Tương tự với sequence <(a)(b)(a)>, có thể nằm trong C1 với các ràng buộc khác nhưng ko thỏa mãn ràng buộc với *duration* (31−5>=25). Cho *mingap* = *maxgap* = 7, that is, exact gap = 7, C3 <1(b)20(b, g)27(e)34(d, g)35(g)> chứa<(b)(e)(d)> nhưng ko chứa <(b)(d)> vì 34−20≠7 (exact gap).

**3.2 Một số thuật ngữ**

**Definition 1 (frequent item- mục phổ biến)** item *x* (mục x) gọi là mục phổ biến *frequent item* trong cơ sở dữ liệu *DB* nếu ngưỡng hỗ trợ của nó >= minsup kí hiệu: <(*x*)>*.sup* ≥ *minsup*.

**Definition 2: (type-1 pattern, type-2 pattern, stem, prefix)** Cho một pattern phổ biến(chuỗi con phổ biến) frequent pattern *P* và frequent item *x* (một mục phổ biến) trong *DB*, *P ’* *là type-1 pattern* (pattern loại 1) nếu nó được tạo ra bằng cách chèn một single item *x* (mục x ) sau phần tử cuối cùng của P v*à type-2 pattern* (pattern loại 2) nếu nó được tạo ra bằng cách chèn một single item *x* (mục x ) vào trong phần tử cuối cùng của P . Mục *x* được gọi là *stem* của pattern mới *P’*. The *prefix pattern*(viết *tắt là prefix*) of *P ’* là *P*. For example, <(a)>, <(a)(b)>, <(a)(d)> and <(a, c)> in Table 3.1 <(a)(b)> and <(a)(d)> are type-1 patterns by adding (b) and (d) after (a) 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)** Coi phần tử đầu tiên của pattern *P* là *FE* và phần tử cuối cùng là *LE* . Nếu *ds* gồm các phần tử *eδ*, *e*γ,…, *eω* chứa *P* vì *FE* ⊆ *e*δ∪ *eδ*+1∪…∪*eε* và *LE* ⊆ *e**γ* ∪*eγ*+1∪…∪*eω*, thời gian xảy ra(occurring time) *tδ* , *tγ* , và *tω* của các itemsets *eδ,e*γ*,* và *eω*, đặt *là initial time*(kí hiệu là it), *last-start time*(abbreviated as lst) và *last-end time*(abbreviated as let) của pattern *P* trong *ds*. Chuỗi kí hiệu it:lst:let được đánh dấu như là *eδ* :*e*γ:*e*ω. Mỗi nhãn thời gian cho một lần xảy ra it:lst:let được tập hợp lại thành danh sách [it1:lst1:let1, it2:lst2:let2,…, itk:lstk:letk], iti≤lsti≤leti với 1≤i≤k. Một danh sách nhãn thời gian như vậy gọi là *time index* of *P* trong *ds*.

Với ví dụ 3.1 vừa rồi, minimum support=50%, mingap=5, maxgap=16, swin=0, and duration=30. <(a)(d)> là một pattern chuỗi nằm trong C2 và C4. For C2 <6(a, c)10(b)17(e)18(a)24(c, d)>, the time stamp đánh dấu là 18:24:24. The initial time của (a) trong <(a)(d)> xảy ra lúc time= 18. The last-start time và last-end time của (d) trong <(a)(d)> xảy ra ở thời gian 24. Tương tự với C4 <5(a)10(d)21(c, d)26(e)>, <(a)(d)> xuất hiện 2 lần và time index of <(a)(d)> được tập hợp lại thành [5:10:10, 5:21:21].

Definition 4: (valid time periods- khoảng thời gian hợp lệ) viết tắt VTPs. Cho trước time index of *P* trong *ds*, VTPs là khoảng thời gian (time periods ) của itemsets in *ds* được dùng để tìm ra các stem *x tiềm* năng của pattern *P’*, trong đó *P* is the prefix of *P’.*

**Lemma 1**: Cho trước time index of *P* in *ds* [it1:lst1:let1, it2:lst2:let2,…, itk:lstk:letk], khoảng thời gian hợp lệ để tạo ra pattern loại 1(type-1 pattern) phải thỏa mãn 1 trong những điều kiện sau đây:

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

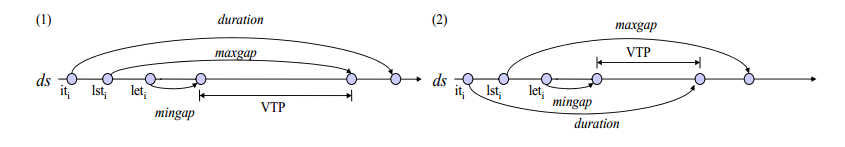


Figure 3.1 Valid time periods of a potential type-1 pattern (Type-1 VTPs)

Ví dụ trong bảng 3.1, <(b)(e)> nằm trong C2 and C3. Với C2 <6(a, c)10(b)17(e)18(a)24(c, d)>, time index được đánh dấu là [10:17:17]. Với pattern loại 1, VTP nằm trong khoảng từ 17+3 (mingap) to 17+15 (maxgap) vì 10+25 (duration) lớn hơn 17 +15(maxgap). Tương tự, với C3, a stem (d) cũng được tìm thấy và <(b)(e)(d)> là pattern mới được khai phá ra.

**Lemma 2**: Cho time index of *P* in *ds* [it1:lst1:let1, it2:lst2:let2,…, itk:lstk:letk], khoảng thời gian hợp lệ để tạo ra type-2 pattern phải thỏa mãn một trong những điều kiện sau đây:

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

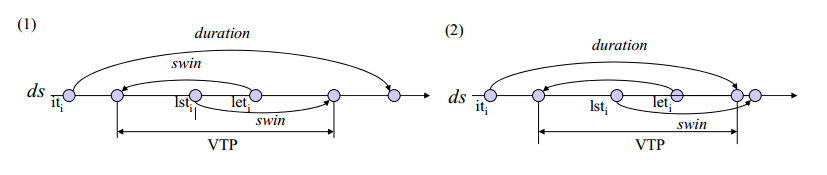


Figure 3.2 Valid time periods of a potential type-2 pattern (Type-2 VTPs)

Chú ý rằng các stem tiềm năng được tìm thấy bằng cách sử dụng Lemma 2 vẫn có thể bị loại bỏ nếu thứ tự bảng chữ cái của nó nhỏ hơn mục cuối cùng của phần tử cuối cùng trong *P*. Với loại 2, khoảng thời gian hợp lệ được kiểm tra xem có vi phạm các ràng buộc min/max gap giữa 2 phần tử cuối cùng . Trong trường hợp P chỉ có 1 phần tử và VTP được dùng sớm hơn sti, thì *duration* cần được kiểm tra thêm.

Ví dụ, VTP của type-2 pattern <(b)(e)> nằm trong khoảng từ 17−2(swin) to 17+2(swin) với C2 <6(a, c)10(b)17(e)18(a)24(c, d)>. (a) xuất hiện ở thời gian 18,vì thứ tự của (a) nhỏ hơn (e) trong bảng chữ cái nên ngưỡng hỗ trợ không tăng.

**3.3 METISP Algorithm: MÔ TẢ CỤ THỂ THUẬT TOÁN METISP**

Giả sử DB có thể load vừa bộ nhớ máy tính, METISP đầu tiên load DB vào bộ nhớ máy tính(kí hiệu là MDB) và quét bộ nhớ máy tính MDB một lần để tìm ra tất cả các items phổ biến . Với mỗi item phổ biến vừa tìm được, METISP sẽ xây dựng 1 time index-set cho 1-sequence (chuỗi một item) và thực hiện đệ quy để xây dựng nên time-constrained sequential patterns có độ dài lớn hơn từ các pattern có 1 item. Time index-set là một cặp gồm(data-sequence pointer, time index). Chỉ những chuỗi dữ liệu data sequences này mới có thể có những item có tiềm năng tạo ra các stem xây dựng nên patterns mới có độ dài lớn hơn. 1 nhãn thời gian gồm 3 phần tử it:lst:let . Thủ tục Mine (P, P-TIdx) khai phá type-1 patterns and type-2 patterns có prefix P bằng cách tìm ra VTPs dùng Lemma 1 and Lemma 2. P-TIdx là time index-set của P. METISP không bao giờ tìm kiếm chuỗi dữ liệu không liên quan tới P . Ngoài ra, VTPs đảm bảo rằng METISP định và đếm các stems mà có thể tạo ra các patterns hợp lệ hơn là toàn bộ items trong chuỗi dữ liệu. Tiếp tục, các pattern loại 1 hoặc 2, P’ mới được tạo, time index-set P’-TIdx cũng được tạo ra và gọi lại hàm Mine(P’, P’-TIdx). Càng vào sâu quá trình xử lý thuộc tính time càng hữu ích, METISP càng hiệu quả trong việc tìm ra pattern mong muốn.

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)

**Theorem 1**: Tất cả các patterns chuỗi( sequential patterns) đều thuộc dạng patterns loại 1và 2 . stem và prefix cần phải thỏa mãn tất cả các ràng buộc thời gian trong chuỗi dữ liệu để tạo ra được pattern mới loại 1 và loại 2. METISP khai phá các pattern này theo phương pháp đệ quy kiểu “pattern growth” tức là cái mới được phát triển từ cái cũ có độ dài ngắn hơn . Với mỗi prefix, METISP đếm mức hỗ trợ của các item trong tất cả VTPs để tạo ra pattern type-1 and type-2. Cứ tiếp tục như vậy, nó tìm ra được các patterns mong muốn.

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);

**3.4 Ví dụ cụ thể 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 memory and 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.

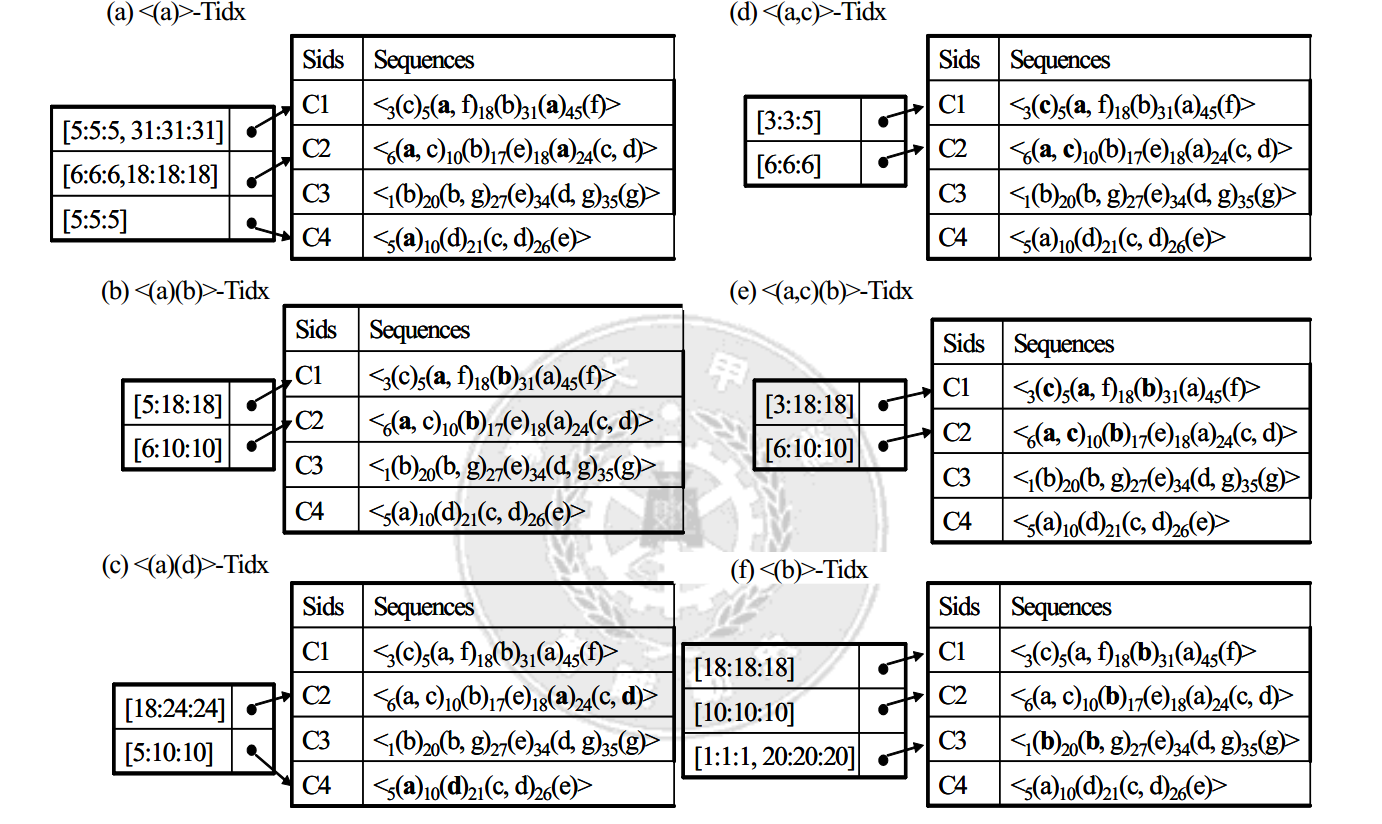


Figure 3.5 Some time index sets of sequential patterns