**One of Proposed Algorithms” called MFSUTC**

We will present a method to speed up the generation of frequent sequential patterns, reduce the computing time of generating *p*’ of (|*p*|+1)-sequences from pattern *p*.

From the experiments, we can find that the proposed method has the less computing time than METISP, which is the available method having the least computing time to our knowledge.

The major differences between the proposed method and METISP are: (a) the proposed method will not look back to find frequent items to form type-2 patterns and METISP will; (b) a termination criterion is used by the proposed method to stop the process of generating infrequent patterns; (c) METISP checks the time index for each item to form *p*’ from *p*; while the proposed method uses each item from a transaction in the set of valid transactions to speed up the generation of *p*’.

**Using valid transactions**

Denote the last added item to *p* as *LIT*, which is an element in *tj* (the *j*th element of a data sequence *dsi* in *D*). Suppose that pattern *p* can be found *r* times from a sequence *dsi*. The time interval record (Tir*i*) of pattern *p* with respect to *dsi* is denoted as *p*- Tir*i* and is represented by *p*-Tir*i* = {*i*, Tir*i* } = {*i*, (it1, lst1, TID1, IID1), (it2, lst2, TID2, IID2),…, (it*r*, lst*r*, TID*r*, IID*r*)}, where *i* is the SID (sequence ID); it*j* and lst*j* are the initial time and last-starting time, respectively, of the *j*th repetition of *p* in *dsi*. TID*j* and IID*j* (*j =* 1 to *r*) are the transaction ID and item ID, respectively, where TID*j* and IID*j* specify the last item *LIT* adding to *p*. The time interval record of pattern *p* is denoted as *p*-Tir = {p*-*Tir*i* }. Note here that the support of *p* equals | *p*-Tir|.

Denote let*j* as the last-ending time of the *j*th repetition of *p* in *dsi*. Note here that let*j* equals , where  is occurring time of the transactionwith TID = TID*j* in *dsi*. In the case that *LIT* is the last element of  and pattern *p*satisfies one of the following conditions: (a) (– lst*j*) > *swin* and (– lst*j*) > *maxgap*, (b) (– lst*j*) > *swin* and (–) < *mingap* and (c) (– it*j*) > *dun* as shown in figure 1, then a pattern *p*’ of length (*k*+1) cannot be formed, where  is the occurring time of the transaction *tnext*, whose transaction ID is (TID*j*+1), next to . This is due to *p*’ will not satisfy (1) the maximum gap or sliding window constraint if condition (a) is satisfied, (2) the sliding window or minimum gap constraint if condition (b) is held, and (3) the duration constraint if condition (c) is satisfied. If one of the above three conditions holds, no valid transaction exists for *p*’, where a valid transaction for *p*’ is used to form sequential patterns of length (*k*+1).

Denote the support of a pattern *p* as *p*.*sup*, where *p*.*sup* = |*p*-Tir|. If there exist *k* sequences with SVT (*p*) = ∅ and (*p*.*sup* - *k*) < *min\_sup*×|*D*|, then no frequent sequential pattern *p*’ of length (|*p*|+1) can be formed, where SVT (*p*) is the set of valid transactions for a pattern *p*. This is because *p*’.*sup* ≤ (*p*.*sup* - *k*) < *min\_sup*×|*D*|. The proposed method will use the early termination condition: (there exist *k* sequences with SVT (*p*) = ∅ and (*p*.*sup* - *k*) < *min\_sup*×|*D*|), to stop the process of generating *p*’ from *p*. This early stopping criterion is not used by METISP. The early termination criterion is implemented in step (2.1) of the pattern generation algorithm presented in the following sections.

The proposed method first determines frequent items and all transactions of a data sequence are lexicographically sorted. Next, this method deletes all infrequent items from all transactions of a data sequence. Note here that the frequency of an infrequent item is less than *min\_sup*×|*D*|.

*ds*

ot*tnext*

*swin*

lst*j*

(a)

*maxgap*

*ds*

(b)

ot*tnext*

*swin*

lst*j*

ot*TIDj*

*mingap*

*ds*

(c)

ot*tnext*

it*j*

*dun*

Figure 1: The condition that *p*’ cannot be formed: (a) (– lst*j*) > *swin* and (– lst*j*) >

*maxgap*, (b) (– lst*j*) > *swin* and (– ) < *mingap*, and (c) (– it*j*) > *dun*.

Let *L*1 consist of all frequent items. For each *x*∈*L*1, set *p* = [{*x*}]. For a frequent pattern *p* of length *k*, two types of potential patterns (type-1 pattern and type-2 pattern) of length (*k*+1) will be generated. The generated pattern of length (*k*+1) is denoted as *p*’. Given a frequent pattern *p* and item *x*, *p*’is a type-1 pattern if it can be obtained by adding {*x*} after the last element of *p*. *p*’is called a type-2 pattern, if the last element of *p*’ is the union of {*x*} and the last element of *p*. Given a time interval record of pattern *p* in a data sequence *dsi*, the proposed method determines a set of valid transactions, whose frequent items are to be used for generating the type-1 or type-2 pattern *p*’ of *p*. This set of valid transactions for a pattern *p* is denoted as SVT*i* (*p*) with respect to a data sequence *dsi*. Let the time interval record of pattern *p* with respect to *dsi* be denoted as *p*-Tir*i* = {*i*, (it1, lst1, TID1, IID1), (it2, lst2, TID2, IID2),…, (it*r*, lst*r*, TID*r*, IID*r*)}. To generate type-1 pattern, a transaction *t*SVT*i* (*p*) should satisfy the following condition: (+ *mingap*)  minimum of (lst*j* + *maxgap*, it*j* + *dun*) for *j* = 1 to *r*, where is the occurring time of transaction *t* in *dsi*. If the item *LIT* indicated by IID*j* is not the last element of , then all elements in transaction  from (IID*j*+1) to |*t*| can be used form the type-2 pattern *p*’. For all items of a transaction *t*, which fulfills the following requirement can also be used to generate the type-2 pattern *p*’: < minimum of (*swin*+lst*j*, it*j*+*dun*) for *j* = 1 to *r*. Note here that the proposed method will not look back to find frequent items to form type-2 patterns and METISP will.

At this stage, we would like to present the pattern generation algorithm and the proposed algorithm, which is referred to as MFSPUTC, below.

Pattern Generation Algorithm

1. Input pattern *p* and the time interval record *p*-Tir. Set *j* = 0, S1temp = S2temp = ∅, and S’ = *p*’-Tir = ∅. Let *support* = | *p*-Tir|.
2. Fetch the next SID in *p*-Tir until no SID can be fetched:

(2.1) Generate the set of valid transactions of *p* for type-1 and type-2 patterns, which are denoted as SVT1 and SVT2, respectively. Let SVTSID (*p*) = SVT1 ∪ SVT2. If SVTSID (*p*) = ∅: (a) set *support* = *support* – 1; (b) if *support* < *min\_sup*×|*D*|, return and output S’ = ∅, otherwise go to step (2).

(2.1.1) If SVT1 ≠ ∅, for each transaction *t*∈SVT1 and each item *x*∈*t*: (a) if *x*∉S1temp: obtain *p*’ by adding *x* to *p*[|*p*|]; and put *x* and *p*’ in S1temp and S’, respectively; (b) update *p*’-TirSID.

(2.1.2) If SVT2 ≠ ∅, for each *t*∈SVT2:

(2.1.2.1) If *t*=, for each element *x* of *t* from (IID*l*+1) to |*t*|: (a) If *x*∉S2temp, set *p*’ = *p*∪{*x*}; put *x* and *p*’ in S2temp and S’, respectively; (b) update *p*’-TirSID.

(2.1.2.2) If *t*≠, for each item *x*∈*t*: (a) If *x*∉S2temp: set *p*’ = *p*∪{*x*}; and put *x* and *p*’ in S2temp and S’, respectively; (b) update *p*’-TirSID.

(3) For each pattern *p*’∈S’: if |*p*’-TirSID|< *min\_sup*×|*D*|, delete *p*’ from S’; otherwise, for each SID, put *p*’-TirSID in *p*’-Tir.

(4) Output S’. For each *p*’ in S’, output *p*’-Tir.

Proposed Algorithm

(1) Input the sequential database *D*, minimum support threshold (*min\_sup*), minimum gap (*mingap*), maximum gap (*maxgap*), sliding window (*swin*) and duration (*dun*). Set Stemp = ∅. Determine the frequencies of all items in *D* and delete the infrequent items from each transaction of all data sequences.

(2) Sort all transactions in data sequences lexicographically. For each SID in *D* and each item *x* of a transaction *t*: (a) If *x*∉Stemp: set *p* = [{*x*}] and update *p*-TirSID and Stemp = Stemp∪{*x*}. (b) If *x*∈Stemp, update *p*-TirSID.

(3) Put all *p* in *L*1. For each *p*∈*L*1 and each SID∈*D*: put *p*-TirSID in *p*-Tir. Set *k* = 1.

(4) Set *Lk*+1 = ∅. For each *p*⊂*Lk* with |*p*-Tir| ≥ *min*\_*sup*×|*D*|: (a) use the pattern generation algorithm to generate S’ and *p*’-Tir for each *p*’ in S’; (b) update *Lk*+1 = *Lk*+1∪ S’; and set *k* = *k* +1.

(5) Repeat step (4) until no frequent sequential pattern can be found.

**An illustrative example**

the mined frequent sequential patterns with *min\_sup* = 50%, *mingap* = 3, *maxgap* = 15, *swin* = 2 and *dun* = 25 using the proposed method. From table 2, we can the frequent items in this data set are “*a*,” “*b*,” “*c*,” “*d*,” and “*e*” with frequencies of 3. In table 2, “[{*a*}, {*d*}]:2” implies that there are two sequences containing the pattern [{*a*}, {*d*}]. We can find that the set of time interval records of pattern for pattern [{*a*}] is [{*a*}]-Tir = {{*a*}-Tir1, {*a*}-Tir2, {*a*}-Tir4}, where [{*a*}]-Tir1 = {1, (5, 5, 2, 1), (31, 31, 4, 1)}, [{*a*}]-Tir2 = {2, (6, 6, 1, 1), (18, 18, 4, 1)}, and [{*a*}]-Tir4 = {4, (5, 5, 1, 1)}. Note here that [{*a*}]-Tir4 = {4, (5, 5, 1, 1)} implies that TID = 1 and IID = 1. That is, the first element of the first transaction in *ds*4 is item “*a*” and the initial time and last start time of [{*a*}] are 5. The same implications hold for [{*a*}]-Tir1 and [{*a*}]-Tir4. Using the presented method, we will generate three frequent patterns [{*a*}, {*b*}], [{*a*}, {*d*}], [{*a*, *c*}] from pattern [{*a*}]. The set of time interval records of pattern for pattern [{*a*}] is [{*a*, *c*}]-Tir = {[{*a*, c}]-Tir1, [{*a*, *c*}]-Tir2}, where [{*a*, *c*}]-Tir1 = {1, (3, 3, 2, 1), (31, 31, 4, 1)} and [{*a*, *c*}]-Tir2 = {2, (6, 6, 1, 1), (18, 18, 4, 1)}. Applying the proposed method for pattern [{*a*, *c*}], we will obtain the frequent pattern [{*a*, *c*}, {*b*}] with support = 2 and [{*a*, *c*}, {*b*}]-Tir = {{1, (3, 18, 3, 10)}, {2, (6, 10, 2, 1)}}. Since SVT1 ([{*a*, *c*}, {*b*}]) = ∅, we will have (|[{*a*, *c*},{*b*}]-Tir| -1 }<*min*\_*sup*×|*D*| = 2. The proposed method will stop the mining process using [{*a*, *c*}, {*b*}] to find frequent patterns of length 4. However, METISP will continue the mining process of using [{*a*, *c*}, {*b*}] to generate patterns of length 4. This will explain the proposed method can reduce the computing time of METISP. From pattern [{*b*}], we will find the frequent patterns [{*b*}, {*a*}], [{b}, {d}], and [{*b*}, {*e*}] with support = 2. The set of time interval records for pattern [{*b*},{*e*}] is [{*b*},{*e*}]-Tir = {[{*b*},{*e*}]-Tir2, [{*b*},{*e*}]-Tir3}, where [{*b*},{*e*}]-Tir2 = {2, (10, 17, 3, 1)} and [{*b*},{*e*}]-Tir3 = {3, (1, 27, 3, 1), (20, 27, 3, 1)}. Using the proposed method, we generate frequent pattern [{*b*}, {*e*}, {*d*}] with [{*b*},{*e*},{*d*}]-Tir = {[{*b*},{*e*},{*d*}]-Tir2}. We will stop the mining process due to |[{*b*},{*e*},{*d*}]-Tir| = 1 < 2.

At this stage, we will summarize the proposed method as follows:

Step 1: Scan *D* to determine the set of frequent items *L*1 and delete the infrequent items.

Step 2: Construct the time interval records of the frequent items.

Step 3: Use the time interval record to generate SVTs.

Step 4: Using the time interval records of sequential patterns to grow all sequential patterns of longer length until no frequent sequential pattern can be found.

Table 1: An example of sequential database

|  |  |
| --- | --- |
| SID | Sequences |
| 1 | {3{*c*}, 5{*a*, *f*}, 18{*b*}, 31{*a*}, 45{*f*}} |
| 2 | {6{*a*, *c*}, 10{*b*}, 17{*e*}, 18{*a*}, 24{*c***,** *d*}} |
| 3 | {1{*b*}, 20{*b*, *g*}, 27{*e*}, 34{*d*, *g*}, 35{*g*}} |
| 4 | {5{*a*}, 10{*d*}, 21{*c*, *d*}, 26{*e*}} |

Table 2: The mined frequent sequential patterns with *min\_sup* = 50%, *mingap* = 3, *maxgap* = 15, *swin* = 2 and *dun* = 25 using the proposed methd

|  |  |  |
| --- | --- | --- |
| SID | Sequences | Frequent sequential patterns |
| 1 | {3{*c*}, 5{*a*, *f*}, 18{*b*}, 31{*a*}, 45{*f*}} | [{*a*}]:3, [{*b*}]:3, [{*c*}]:3, [{*d*}]:3, [{*e*}]:3, [{*a*}, {*b*}]:2, [{*a*}, {*d*}]:2, [{*a*, *c*}]:2, [{*b*}, {*a*}]:2, [{b}, {d}]:2, [{*b*}, [*e*}]:2, [{*c*}, {*b*}]:2, [{*c*}, {*e*}]:2, [{*c*, *d*}]:2, [{*e*}, {*d*}]:2, [{*a*, *c*}, {*b*}]:2, [{*b*}, {*e*}, {*d*}]:2 |
| 2 | {6{*a*, *c*}, 10{*b*}, 17{*e*}, 18{*a*}, 24{*c***,** *d*}} |
| 3 | {1{*b*}, 20{*b*, *g*}, 27{*e*}, 34{*d*, *g*}, 35{*g*}} |
| 4 | {5{*a*}, 10{*d*}, 21{*c*, *d*}, 26{*e*}} |