# Learning Concept Recommendation based on Sequential Pattern Mining

Loc Nguyen[1], Phung Do[2]
Loc Nguyen, Faculty of Information Technology, The University of Natural Science, Ho Chi Minh city, Vietnam, email: ng_phloc@yahoo.com
Phung Do, Faculty of Information System, The University of Information Technology, Ho Chi Minh city, Vietnam, email: dtminhphung@yahoo.com

***Abstract*—Sequential pattern mining is new trend in data mining domain with many useful applications, especially commercial application but it also results surprised effect in adaptive learning. Suppose there is an adaptive e-learning website, a student access learning material / do exercises relating domain concepts in sessions. His learning sequences which are lists of concepts accessed after total study sessions construct the learning sequence database S. S is mined to find the sequences which are expected to be learned frequently or preferred by student. Such sequences called sequential patterns are use to recommend appropriate concepts / learning objects to students in his next visits. It results in enhancing the quality of adaptive learning system. This process is sequential pattern mining. In paper, we also suppose an approach to break sequential pattern s=□c1, c2,…, cm □into association rules including left-hand and right-hand in form ci→cj. Left-hand is considered as source concept, right-hand is treated as recommended concept available to students.**

## I. INTRODUCTION

Recommendation methods are categorized into three different trends:

- Rule-based filtering system is based on manually or automatically generated decision rules that are used to recommend items to users.
- Content-based filtering system recommends items that are considered appropriate to user information in his profile.
- Collaborative filtering system is considered as social filtering when it matches the rating of a current user for items with those of similar users in order to produce recommendations for new items.

Sequential pattern mining belongs to collaborative filtering family. User does not rate explicitly items but his series of chosen items are recorded as sequences to construct the sequence database which mined to find frequently repeated patterns he can choose in future. In learning context, items can be domain concepts / learning objects which students access or learn. First, we should glance over basic concepts and what is sequential pattern mining along with its application, especially in adaptive learning.

Suppose $I = \{i_1, i_2, i_3, \ldots, i_n\}$ is a set of all items. An itemset is a subset of I, denoted $x_i=(i_u, \ldots, i_v)$, if $x_i$ has only one item $i_k$, it can be denoted $x_i=i_k$, omitting the brackets. An sequence is an ordered list of itemsets, denoted $s = \langle x_1, x_2, \ldots, x_m \rangle$, where $x_i$ is an itemset, $x_i$ is also called an element in sequence s. An item has only been once in an element of sequence but can occur multiple times in different elements in sequences. The number of instances of items occurring in sequence is the length of sequence. Sequence with length $l$ is called $l$-sequence.

Sequence $s_1=\langle x_1, x_2, \ldots, x_m \rangle$ is called sub-sequence of $s_2=\langle y_1, y_2, \ldots, y_m \rangle$ denoted $s_1 \subseteq s_2$ or $s_1 \supseteq s_2$ if there is the ordered series of indexes $k_1, k_2, \ldots, k_n$ so that $1 \le k_1 \le k_2 \le \ldots \le k_n \le m$ and $x_1 \subseteq y_{k_1}$, $x_2 \subseteq y_{k_2}, \ldots, x_n \subseteq y_{k_n}$. Assertion "$s_1$ is sub-sequence of $s_2$" is equivalent to "$s_2$ is super-sequence of $s_1$" For example, suppose that:

- A set of all items $I=\{i_1, i_2, i_3, i_4\}$
- Six itemsets $x_1=i_1$, $x_2=(i_1,i_3)$, $x_3=(i_1,i_2,i_3)$, $x_4=(i_1,i_2,i_3,i_4)$, $x_5=i_3$, $x_6=(i_1,i_2)$
- Three sequences $s_1$, $s_2$ denoted as $s_1=\langle x_1,x_4 \rangle$, $s_2=\langle x_1,x_2,x_3 \rangle$, $s_3=\langle x_5,x_6 \rangle$ but $s_1$, $s_2$ are often showed in detailed forms: $s_1=\langle i_1(i_1,i_2,i_3,i_4) \rangle$, $s_2=\langle i_1(i_1,i_3)(i_1,i_2,i_3) \rangle$, $s_3=\langle i_3(i_1,i_2) \rangle$

We have:

- $s_1$, $s_2$, $s_3$ are 5-length, 6-length, 3-length sequences, respectively. Note, item $i_1$ occurs 2 times in sequence $s_1$, so it contributes 2 to length of $s_1$
- $s_3$ is sub-sequence of $s_2$ due to $x_5 \subseteq x_2$ and $x_6 \subseteq x_3$, $s_1$ is not sub-sequence of $s_2$ because $x_1 \subseteq x_2$ but $x_4$ is not subset of any subset in $s_2$

Suppose the sequential database S has a set of records $\langle sid, s \rangle$ where $sid$ is an identifier of sequence $s$. A record $\langle sid, s \rangle$ is said to contain sequence $\alpha$ if $\alpha \subseteq s$. The support of sequence $\alpha$ is the fraction of total records

containing $\alpha$, denoted *support(α)*=| { ⟨ *sid, s* ⟩ | $\alpha \subseteq s$} |. Similar to association rules, given the number *min_sup* as support threshold, if the support of $\alpha$ is equal or greater than *min_sup*, namely *support(α)* ≥ *min_sup* then $\alpha$ is called frequent or large sequence. A sequence is maximal if it has no super-sequence. The maximal frequent sequence is called **sequential pattern**, it means that all super-sequences of sequential pattern are infrequent. In case property "maximal" is not paid attention, frequent sequence is considered as sequential pattern. If *s* is infrequent sequence but all its sub-sequences are frequent, *s* is called minimal infrequent sequence.

Totally, given a sequence database *S* and the threshold *min_sup*, sequential pattern mining is to find all complete set of sequential patterns in *S*.

*Issue of learning concepts / objects recommendation*

Suppose there are compulsory concepts (subjects) in Java course,: **d**ata type, **p**ackage, class & **O**OP, **s**election structure, **v**irtual machine, **l**oop structure, **c**ontrol structure, and inter**f**ace which in turn denoted as *d, p, o, s, v, l, c, f*. At our e-learning website, students access learning material relating such domain concepts in sessions, each session contains only one itemset and is ordered by time. The student's learning sequence is constituted of itemsets accessed in all his sessions.

Table 1 Learning sessions → learning sequences

| Student | Session | Concept accessed |
|---|---|---|
| 1 | Aug 5 10:20:01 | o |
| 1 | Aug 5 10:26:12 | f |
| 2 | Aug 6 08:20:01 | d, p |
| 2 | Aug 6 14:15:01 | o |
| 2 | Aug 6 15:00:00 | s, l, c |
| 3 | Aug 7 12:30:00 | o, v, c |
| 4 | Aug 8 07:14:20 | o |
| 4 | Aug 8 07:40:25 | s, c |
| 4 | Aug 8 10:17:20 | f |
| 5 | Aug 8 10:26:15 | f |

| ID | Learning sequences | Length |
|---|---|---|
| 1 | ⟨ *of* ⟩ | 2 |
| 2 | ⟨ (*dp*)*o*(*slc*)⟩ | 6 |
| 3 | ⟨ (*ovc*) ⟩ | 3 |
| 4 | ⟨ *o*(*sc*)*f*⟩ | 4 |
| 5 | ⟨ *f*⟩ | 1 |

Students accessed learning material in their past sessions, how system recommends appropriate domain concepts to student for next visits. It issues the application of mining sequential patterns. In e-learning context, (learning) sequential pattern is a sequence of concepts / learning materials that students prefer to study or access regularly. Our solution includes two steps as following:

- Applying techniques of mining user learning data to find learning sequential patterns.
- Breaking such patterns into concepts / learning materials which are recommended to users.

In session 2: we survey methods to mine learning sequence patterns. The approach to break patterns into concepts is proposed in session 3.

## II. APPROACHES OF LEARNING SEQUENTIAL PATTERN MINING

There are two main approaches of sequential pattern mining:

- Candidate generation-and-test approach based on algorithm Apriori is also classified into two categories: horizontal and vertical data format methods.
- Pattern-growth approach based on pattern-growth mining of frequent patterns in transaction database.

### *A. Candidate generation-and-test approach*

This approach is essentially an extension of associate rule discovering algorithm Apriori satisfying the statement "*every non-empty sub-sequence of a frequent sequence is a frequent sequence*" (and vice versa) considered as downward closure property. This approach has two trends: horizontal and vertical data format based sequential pattern with respect to three algorithms: AprioriAll [4], GSP [3] and SPADE.

**AprioriAll**

This algorithm is the most similar to algorithm Apriori but applied for sequential pattern mining and has 3 phases: large itemset phase, transformation phase, sequence phase and maximal phase. Sequence phase is the most important

1. Large itemset phase

The **support of itemset** $x_i$ is defined as the fraction of total sequences which containing $x_i$. Note that $x_i$ is contained in a sequence if any itemset in sequence contains it. In case $x_i$ occurs many times in the same sequence, its support is increased only once. Itemset $x_i$ is called *frequent itemset* or *large itemset* or ***litemset*** in brief if its support satisfies the support threshold (≥*min_sup*). If sequence s is recognized as large sequence (frequent sequence) then all its itemsets are litemsets.

Suppose all itemset in sequence database *S* are litemsets, the length of sequence is re-defined as the number of litemsets contained in it. Obviously, both supports of *1*-sequence ⟨ $l_i$ ⟩ and litemset $l_i$ are the same.

This phase finds all litemsets in *S* with support threshold *min_sup*. They are considered as atomic elements in sequences and can be mapped to integer in convenient. Let threshold *min_sup=25%*, just having litemsets *o, s, c,* (*sc*)*, h*. All litemsets are mapped to integer for convenience and each of them looks like as

single item appropriate to re-definition of sequence length.

Table 2 Large itemsets are mapped to integers

| *litemsets* | *mapped to* |
|---|---|
| o | 1 |
| s | 2 |
| c | 3 |
| (sc) | 4 |
| f | 5 |

2. Transformation phase

   In this phase, each itemset $x_i$ in sequence database $S$ are replaced by litemset $l_i$ where $x_i \subseteq l_i$ ($l_i$ contains $x_i$). If a sequence has no litemset, it will be removed from $S$ but it still contributes to the count of sequences.

Table 3 Transformed sequences

| ID | Original sequence | Transformed sequence | Mapped |
|---|---|---|---|
| 1 | ⟨ *of* ⟩ | ⟨ *of* ⟩ | ⟨ *15*⟩ |
| 2 | ⟨ (*dp*)*o*(*slc*) ⟩ | ⟨ *o*(*sc*(*sc*)) ⟩ | ⟨ *1*(*234*) ⟩ |
| 3 | ⟨ (*ovc*) ⟩ | ⟨ (*oc*) ⟩ | ⟨ (*13*) ⟩ |
| 4 | ⟨ *o*(*sc*)*f*⟩ | ⟨ *o*(*sc*(*sc*))*f*⟩ | ⟨ *1*(*234*)*5* ⟩ |
| 5 | ⟨ *f* ⟩ | ⟨ *f* ⟩ | ⟨ *5* ⟩ |

3. Sequence phase

   The purpose of sequence phase is to find all large sequences. This is iterative process that scans database many times to find the large *k*-sequences where *k* is increased until no large sequence can be found. Let $L_k$ is a set of large *k*-sequence.

   i. At the beginning of $k^{th}$ scan, all sequences in $L_{k-1}$ are joined to generate $C_k$, the set of new potential large *k*-sequences which is also called candidate sequences
   ii. After that, the supports of candidate sequences in $C_k$ are computed and assessed whether they satisfy *min_sup* so that we can determine the actually large sequences. The set of actually large *k*-sequences is $L_k$.
   iii. $L_k$ is re-used as the seed for generating $L_{k+1}$

The technique for generating candidate sequence Ck is similar to join operator in SQL-language.

INSERT INTO $C_k$
SELECT $p.litemset_1, p.litemset_2, \ldots,$
$p.litemset_{k-1}, q.litemset_{k-1}$
FROM $L_{k-1}$ *p*, $L_{k-1}$ *q*
WHERE $p.litemset_1 = q.litemset_1$ *AND*
$p.litemset_1 = q.litemset_1$ *AND… AND*
$p.litemset_{k-2} = q.litemset_{k-2}$

Because all sub-sequences of a large sub-sequence are also large and vice versa, we should weed out sequence $c \in C_k$ whose sub-sequences don't occur in $L_{k-1}$.

Table 4 Candidate generation

| Large 3-sequences | Candidate 4-sequence | Candidate 4-sequence (weeded out) |
|---|---|---|
| ⟨ *123* ⟩ | ⟨ *1234* ⟩ | ⟨ *1234* ⟩ |
| ⟨ *124* ⟩ | ⟨ *1243* ⟩ | |
| ⟨ *134* ⟩ | ⟨ *1345* ⟩ | |
| ⟨ *135* ⟩ | ⟨ *1354* ⟩ | |
| ⟨ *234* ⟩ | | |

Finally, when all larger sequences *L* were discovered, we must find maximal sequences among *L* because sequential patterns are maximal large sequences. Suppose n is the length of longest sequences in *L*, such operator is done as follow:

*for(i = n; i > 1; i--)*
*foreach i-sequence si do "remove sub-sequences of si from L"*

In this example there is only one larger sequence ⟨ *1234*⟩ , so it also is maximal.

**GSP** (**G**eneralized **S**equential **P**attern)

This algorithm is a variant of AprioriAll. It also adopts multiple passes over database, uses previous frequent sequences as seeds to generate candidate sequences in each pass and test them to find the actually frequent sequences (see Apriori) which re-used for next pass. However, there are some differences:

- There is no large itemset phase and transformation phase. The length of sequence is actually the number of its items (not litemsets) like original definition. So all frequent sequences in $L_1$ have only one item in form ⟨ *x*⟩
- The way to generate candidate set produces total possible combination of sequences. For example, $L_1$={⟨ *a* ⟩ , ⟨ *b* ⟩ , ⟨ *c* ⟩ } having three 1-sequences generate fifteen 2-sequences, $C_2$={⟨ *aa*⟩ , ⟨ *ab* ⟩ , ⟨ *ac* ⟩ , ⟨ *ba* ⟩ , ⟨ *bb* ⟩ , ⟨ *bc* ⟩ , ⟨ *ca*⟩ , ⟨ *cb* ⟩ , ⟨ *cc* ⟩ , ⟨ (*ab*) ⟩ , ⟨ (*ac*) ⟩ , ⟨ (*ba*) ⟩ , ⟨ (*bc*) ⟩ , ⟨ (*ca*) ⟩ , ⟨ (*cb*) ⟩ }. If $L_1$ have *n 1*-sequence, it generates $n^2 + n*(n-1)/2$ elements in candidate set $C_2$.

Evaluations: GSP finds more frequent sequences than AprioriAll because of generating full of candidate sets but it can raise the problem of the large number of candidate sequences. Suppose there are *300* large sequences of 1-length, algorithm will deliver $300^2 + 300*299/2 = 134{,}850$ candidate 2-sequences.

**SPADE** (**S**equential **PA**ttern **D**iscovery using **E**quivalent classes)

AprioriAll and GPS generate the candidate sequences by scanning database in horizontal data format. That scanning row by row requires accessing database many times. It takes the consequence of downward performance. Whereas SPADE mines frequent sequences by considering sequence database as vertical format data set. This algorithm has below steps:

- Cracking the sequence database into a vertical format data set in which each single itemset associates with the sequence identifier (*sid*) and its individual event identifier (*eid*) to form the new record. For example, a sequence record $\{1, \langle\ itemset_1, itemset_2, itemset_3\ \rangle\ \}$ is broken into three records: $\{1, 1, \langle\ itemset_1\ \rangle\ \}$, $\{1, 2, \langle\ itemset_2\ \rangle\ \}$, $\{1, 3, \langle\ itemset_1\ \rangle\ \}$. Note in simple case, itemset is replaced by item if we consider that each itemset has only one item.

Table 5 Cracking sequence database (table 1) into vertical format data sets

| sid | eid | itemsets (item) |
|---|---|---|
| 1 | 1 | of |
| 2 | 1 | dp |
| 2 | 2 | o |
| 2 | 3 | slc |
| 3 | 1 | ovc |
| o 4 | 1 | o |
| 4 | 2 | sc |
| 4 | 3 | f |
| 5 | 1 | f |

- The frequent *1*-sequences are found by exploring frequent itemsets. Then, each candidate *k-sequence* is formed by joining two frequent (*k-1*)-*sequences* if they share the same *sid* and their event identifiers (*eid)* follow the sequential order. The length of frequent sequences is growth until reaching maximal length or it is not able to find any more frequent itemsets.

Table 6 (1). frequent itemsets (*o*) and (*f*) are associated with pairs (*sid, eid*)

(2). *2*-sequence $\langle\ of\ \rangle$ are found by joining itemsets (*o*), (*f*) in (1)

| *o* | | *f* | |
|---|---|---|---|
| sid | eid | sid | eid |
| 1 | 1 | 1 | 1 |
| 2 | 2 | 4 | 3 |
| 3 | 1 | 5 | 1 |
| 4 | 1 | | |

| *of* | | |
|---|---|---|
| sid | eid(o) | eid(f) |
| 1 | 1 | 1 |
| 4 | 1 | 3 |
| | | |

Evaluation: SPADE reduce frequency of global database access because candidate sequence generation is based on local vertical format data set in which itemsets, sub-sequence are represented by their pair (*sid, eid*). However, nature of SPADE is similar to AprioriAll and GPS, the weakness of exploring breadth-first search and generating large candidate sequences still exists in SPADE.

*B. Pattern-growth approach*

Huge candidate sets generated in AprioriAll, GSP, SPADE are caused by the large number of elements in seed set. Mining separately frequent sequences with disjointed database for the purpose of reduce the number of elements is idea of FreeSpan algorithm.

**FreeSpan**

(**Fre**quent pattern-projected **S**equential **pa**ttern**n**)

Let a sequence be $s=\langle\ s_1, s_2, \ldots, s_k\ \rangle$, the itemset $\bigcup_{i=1}^{k} s_i$ is defined as *projected itemset* of *s*. The algorithm's methodology obeys the property "if an itemset *X* is infrequent, sequences whose projected itemset contains *X* is infrequent too". The algorithm has two following steps:

1. Determine a descending ordered set of all frequent item $f\_list = \{x_1, x_2, \ldots, x_n\}$. Note that $support(x_1) > support(x_2) > \ldots > support(x_n)$. According to *f_list*, all frequent sequences can be divided into *n* disjointed subset $FS_i$ following condition: $FS_i$ contains items $x_i$s but no item after $x_i$.
2. Each subset $FS_i$ is found separately by mining particularly $x_i$-*projected database* by other algorithms such as GSP, AprioriAll… where $x_i$-*projected database* is constructed by removing all items (after $x_i$) from each sequence of original database. For example, if sequence *s* in original database contains $x_i$, it is replaced by sub-sequence *s'* which is derived by removing $x_i$ from *s*.

For example, from table 1, *f_list* is determined along with the support of each item in form $\{x_1{:}support(x_1); x_2{:}support(x_2); \ldots; x_n{:}support(x_n)\}$ as following:

*f_list* = {*o:4, c:3, f:3, s:2, d:1, p:1, v:1, l:1*}

Let *min_sup* be *25%*, applying FreeSpan for *o,c* - projected databases, we have results as below:

Table 7 Frequent sequences mined from projected databases

| | Sequences | Frequent sequences (Mined by GPS…) |
|---|---|---|
| *o*-projected database | $\langle\ o\rangle$ ,$\langle\ o\rangle$ ,$\langle\ o\rangle$ ,$\langle\ o\rangle$ | $\langle\ o\rangle$ :4 |
| *c*-projected database | $\langle\ o\rangle$ , $\langle\ oc\rangle$ , $\langle\ (oc)\ \rangle$ , $\langle\ oc\ \rangle$ | $\langle\ c\rangle$ :3, $\langle\ oc\rangle$ :2 |

Evaluation: FreeSpan is more efficient than Apriori-like algorithms since it projects recursively a large database into smaller databases according to current frequent sequences. Generation of longer sequences is done on such small databases leading to a smaller set of candidate. However, FreeSpan can create redundant projected databases, which affects performance.

## III. A PROPOSAL OF BREAKING LEARNING SEQUENTIAL PATTERN

Suppose we discovered the sequential pattern ⟨ *osc*(*sc*) ⟩ which means:

*"class & **O**OP" → "**s**election structure" → "**c**ontrol structure" → "**s**election structure, **c**ontrol structure"*

and some other patterns as results in algorithm AprioriAll. We accept that these patterns like the learning "routes" that student preferred or learned often in past but in the next time if a student chooses the concept "***c**ontrol structure*", the adaptive learning system should recommend which next concepts in above patterns? So the patterns should be broken into association rules with their confidence. For example, breaking above pattern ⟨ *osc*(*sc*) ⟩ follows steps:

1. Breaking entire ⟨ *osc*(*sc*) ⟩ into litemsets such as *o, s, c,* (*sc*) and determining all possible large 2-sequences which are 2-arrangment of all litemsets following the criterion: order of 2-sequences must comply with the order of sequential pattern. There are six large 2-sequences: ⟨ *os* ⟩ , ⟨ *oc* ⟩ , ⟨ *o*(*sc*) ⟩ , ⟨ *sc* ⟩ , ⟨ *s*(*sc*)⟩ , ⟨ *c*(*sc*) ⟩ . Thus, we have six rules derived from these large *2*-sequences in form: "*left-hand litemset → right-hand litemset"*, for example, rule "*s→c*" derived from *2*-sequence ⟨ *sc* ⟩
2. Computing the confidences of such rules and sorting them according to these measures. The confidence of a rule is the ratio of the support of *2*-sequences and the support of left-hand litemset, *confidence(x → y) = support*( ⟨ *xy* ⟩ ) / *support*((*x*)). The rules whose confidence is less than threshold *min_conf* is removed in order to ensure that remains are strong rules. We called these rules as **sequential rules** and these confidences as sequential confidences

Table 8 Sequential rules

| sequential rules | sequential confidences |
|---|---|
| o→s | 40% |
| o→c | 40% |
| o→sc | 40% |
| s→c | 0% |
| s→sc | 0% |
| c→sc | 0% |

Now, if student choose the concept (itemset) *x*, system will find whole rules broken from all sequential patterns and the *left-hand* litemset of each rule must contain *x*. Then, these rules are sorted by their confidences in descending order. Final outcome is an ordered list of *right-hand* litemsets (concepts) of rules, which are recommended to students. The top concept (litemset) in such list is referred as the most necessary concept. For example, as results in table 8, if users choose concept "*class & **O**OP*", we recommend them other concepts as below:

| Concepts | Rates |
|---|---|
| *"selection structure"* | 40% |
| *"control structure"* | 40% |

Evaluations: This methodology is similar to mining association rules but it achieves high performance and precise prediction since it derived from result of sequential pattern mining process. The sequential rules stick close on user's learning "route" because they are mined in accordance with sequential pattern and pay attention to the sequence order.

## IV. CONCLUSION

Although sequential pattern mining is applied in e-commercial for customer purchase but the extracted patterns can be used to predict user's learning "route", which is fundamental of learning object recommendation. There are two sequential pattern mining approaches: candidate generation-and-test, pattern-growth. The first which is essentially a refinement of the Apriori-like is easy to implement but causes a problem of large candidate set and so, leads to performance downfall and requirement of both complex computation and huge storage. Especially, in e-learning environment, there are thousands of students; speed and performance are very important factors. The second which is a divide-and-conquer solution intends to reduce the number of candidate sequences. So, it is more efficient.

Last, we propose the technique to break sequential patterns into rules containing recommendable concepts / learning materials. The ideology of this approach is derived from association rule mining but it is more efficient than association rules.

## V. REFERENCES


[1] M. Zaki, "An efficient algorithm for mining frequent sequences", Machine Learning, 40:31-60, 2001.

[2] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation", In Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD00), pp. 112, Dallas, TX, May 2000

[3] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements", In Proc. 5th Int.

Conf. Extending Database Technology (EDBT96), pp. 317, Avignon, France, Mar. 1996

[4] R. Agrawal and R. Srikant, "Mining sequential patterns", In Proc. 1995 Int. Conf. Data Engineering (ICDE95), pp.314, Taipei, Taiwan, Mar.1995.

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", In Proc. 1994 Int. Conf. Very Large DataBases (VLDB94), pp. 487-499, Santiago, Chile, Sept. 1994.