This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

TNS: Mining Top-K Non-Redundant Sequential Rules

Philippe Fournier-Viger[1] and Vincent S. Tseng[2]

[1] Department of Computer Science, University of Moncton, Moncton, Canada

[2] Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC

philippe.fournier-viger@umoncton.ca, tsengsm@mail.ncku.edu.tw

ABSTRACT

Mining sequential rules from sequence databases is an important topic in data mining with wide applications. However, depending on the choice of the thresholds, current algorithms can become very slow and generate an extremely large amount of results or generate too few results, omitting valuable information. Furthermore, it is well-known that a large proportion of sequential rules generated are redundant. In previous works, these two problems have been addressed separately. In this paper, we address both of them at the same time by proposing an algorithm named TNS for mining top-k non redundant sequential rules.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Applications – *data mining.*

General Terms

Algorithms, Performance.

Keywords

Sequential rules, top-k pattern mining, redundancy, algorithm

1. INTRODUCTION

Various methods have been proposed for mining temporal patterns in sequence databases such as mining repetitive patterns, trends and sequential patterns (see [1] for a survey). Among them, *sequential pattern mining* is probably the most popular set of techniques (e.g. [2, 3]). It consists of finding subsequences appearing frequently in a set of sequences. However, knowing that a sequence appear frequently is not sufficient for making predictions [4]. An alternative that addresses the problem of prediction is *sequential rule mining* [4-13]. A sequential rule indicates that if some item(s) occurred, some other item (s) are likely to occur afterward with a given confidence or probability. Sequential rules can be discovered in a single sequence [12, 16], across sequences [7, 8, 9] or common to several sequences [4, 5, 6, 10, 13]. In this paper, we use the definition of [4] because it has two reported real applications and because not many works have addressed the case of discovering sequential rules common to several sequences, despite that it has many potential applications [4, 13]. According to this definition, the problem of *sequential rule mining* is stated as follows. A *sequence database* is a set of sequences $S=\{s_1, s_2 \ldots s_s\}$ and a set of items $I=\{i_1, i_2, \ldots i_t\}$ occurring in these sequences. A *sequence* is defined as an ordered list of *itemsets* (sets of items) $s_x=I_1, I_2, \ldots I_n$ such that $I_1, I_2, \ldots I_n \subseteq I$, and where each sequence is assigned a unique *sid* (sequence id). As an example, Figure 1 depicts a sequence database containing four sequences with sids *s1*, *s2*, *s3* and *s4*. In this example, each single letter represents an item. Items between curly brackets represent an itemset. For instance, the sequence *s1* means that items *a* and *b* occurred at the same time, and were followed successively by *c*, *f*, *g* and *e*. A *sequential rule* X⇒Y is defined as a relationship between two itemsets X, Y ⊆ I such that X∩Y = Ø, X, Y ≠ Ø, and X and Y are unordered. The interpretation of a rule X⇒Y is that if items in X occur in a sequence, the items in Y will occur afterward in the same sequence [4]. For example, the rule {*a*, *b*, *c*}⇒{*e*, *f*, *g*} occurs in the sequence {*a*, *b*}, {*c*}, {*f*}, {*g*}, {*e*}, whereas the rule {*a*, *b*, *f*}⇒{*c*} does not because item *c* does not occur after *f*. For a given sequence database and a rule X⇒Y, the notation *sids*(X⇒Y) represents the set of sequences where the rule occurs. For an itemset X and a sequence database, the notation *sids*(X) denotes the sequences where all the items of X appears. For example, sids({*a*, *b*, *c*}) = {*s1*, *s2*}. Two interestingness measures are defined for sequential rules, which are similar to those used in association rule mining [14]. The *support* of a rule X⇒Y is defined as sup(X⇒Y) = |*sids*(X⇒Y)| / |S|. The *confidence* is defined as conf(X⇒Y) = |*sids*(X⇒Y)| / |*sids*(X)|. The *problem of mining sequential rules* is to find all rules such that their support and confidence are respectively no less than user-defined thresholds *minsup* and *minconf*. For instance, Figure 1 (right) shows some rules found in the database shown in Figure 1 (left) for *minsup* = 0.5 and *minconf* = 0.5.

ID	Sequences
s1	{*a*, *b*},{*c*},{*f*},{*g*},{*e*}
s2	{*a*, *d*},{*c*},{*b*},{*a*, *b*, *e*, *f*}
s3	{*a*},{*b*},{*f*},{*e*}
s4	{*b*},{*f*, *g*}

ID	Rule	Supp.	Conf.
r1	{*a*, *b*, *c*}⇒{*e*}	0.5	1.0
r2	{*a*}⇒{*c*, *e*, *f*}	0.5	0.66
r3	{*a*, *b*}⇒{*e*, *f*}	0.5	1.0
r4	{*b*}⇒{*e*, *f*}	0.75	0.75
r5	{*a*}⇒{*e*, *f*}	0.75	1.0
r6	{*c*}⇒{*f*}	0.5	1.0
r7	{*a*}⇒{*b*}	0.5	0.66
…	…	…	…

Fig 1. A sequence database (left) and some sequential rule found (right)

Despite that much research has been done on sequential rule mining, an important issue that has been overlooked is how users should choose the *minsup* and *minconf* thresholds to generate a desired amount of rules [4-13]. This is an important problem because in practice users have limited resources (time and storage space) for analyzing the results and thus are often only interested in discovering a certain amount of rules, and fine tuning the parameters is time-consuming. Depending on the choice of the thresholds, current algorithms can become very slow and generate an extremely large amount of results or generate none or too few results, omitting valuable information. To address this problem, it was proposed to mine top-k sequential rules, where *k* is the number of sequential rules to be found, and is set by the user [10]. However, a major problem with this approach is that top-k sequential rules often contain a large proportion of redundant rules. i.e. rules that provide information that is redundant to the user. This problem has been confirmed in our experimental study (cf. section 4). We found that up to 79.59 % of top-k sequential

This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

rules are redundant for datasets commonly used in the data mining literature. This means that the user has to analyze a large proportion of redundant rules when using such algorithms.

The problem of eliminating redundancy in sequential rule mining has been previously studied [5]. However, it remains an open challenge to combine the idea of mining a set of non-redundant rules with the idea of top-k sequential rule mining, to propose an efficient algorithm to mine top-k non-redundant sequential rules. The benefit of such an algorithm would be to present a small set of *k* non-redundant rules to the user. However, devising an algorithm to mine these rules is difficult. The reason is that eliminating redundancy cannot be performed as a post-processing step after mining the top-k sequential rules, because it would result in less than *k* rules. The process of eliminating redundancy has therefore to be integrated in the mining process.

In this paper, we undertake this challenge by proposing an algorithm for mining the top-k non redundant sequential rules that we name TNS (*T*op-k *N*on-redundant *S*equential *R*ules). It is based on a recently proposed approach for generating sequential rules that is named "rule expansions", and adds strategies to avoid generating redundant rules. An evaluation of the algorithm with datasets commonly used in the literature shows that TNS has excellent performance and scalability.

The rest of the paper is organized as follows. Section 2 reviews related works and presents the problem definition. Section 3 presents TNS. Section 4 presents an experimental evaluation. Section 5 presents the conclusion.

2. RELATED WORKS AND PROBLEM DEFINITION

2.1 Top-k Sequential Rule Mining

To the best of our knowledge, TopSeqRules [10] is the only algorithm to discover top-k sequential rules. TopSeqRules takes two parameters named *k* and *minconf* and a sequence database as input, and it returns the *k* rules with the highest support that meet the *minconf* threshold. The reason why this algorithm defines the task of mining the top-k rules on the support instead of the confidence is that *minsup* is much more difficult to set than minconf because *minsup* depends on database characteristics that are unknown to most users, whereas *minconf* represents the minimal confidence that users want in rules and is therefore generally easy to determine. TopSeqRules defines the problem of top-k sequential rule mining as follows [10].

Definition 1. The *problem of top-k sequential rule mining* is to discover a set L containing *k* rules in T such that for each rule $r \in L \mid conf(r) \geq minconf$, there does not exist a rule $s \in L \mid conf(s) \geq minconf \wedge sup(s) > sup(r)$.

2.2 Discovering Non-Redundant Rules

Few works have studied the problem of discovering non redundant sequential rules. To our knowledge only Lo et al [5] proposed algorithms for mining non redundant sequential rules. However, these algorithms are designed based ons a less general definition of sequential rules than the one used in this paper [4]. A definition that is closer to the one used in this paper is the definition of rules in association rule mining. In this domain, researchers have proposed to mine several sets of non-redundant association rules such as the Generic Basis [15], the Informative Basis [15], the Informative and Generic Basis [16], the Minimal Generic Basis [17] and Minimum Condition Maximum Consequent Rules [18]. These rule sets can be compared based on several criteria such as their compactness, the possibility of recovering redundant rules with their properties (support and confidence), and their meaningfulness to users (see [18] for a detailed comparison). In this paper, we based our work on Minimum Condition Maximum Consequent Rules (MCMR). The reason is that the most important criteria in top-k sequential rule mining is the meaningfulness of rules for users. MCMR meet this goal because it defines non-redundant rules as rules with a minimum antecedent and a maximum consequent [18]. In other words, MCMR are the rules that allow deriving the maximum amount of information based on the minimum amount of information. It is argued that these rules are the most meaningful for several tasks [18]. In the context of top-k sequential rule mining, other criteria such as the compactness and recoverability of redundant rules are not relevant because the goal of a top-k algorithm is to present a small set of *k* meaningful rules to the user. MCMR are defined based on the following definition of redundancy [18].

Definition 2. A rule r_a: $X \rightarrow Y$ is *redundant* with respect to another rule $r_b : X_1 \rightarrow Y_1$ if and only if $conf(r_a) = conf(r_b) \wedge sup(r_a) = sup(r_b) \wedge X_1 \subseteq X \wedge Y \subseteq Y_1$.

Example. Consider the sequential rules presented in the right part of Figure 1. The rule $\{a\} \rightarrow \{c, f\}$ is redundant with respect to $\{a\} \rightarrow \{c, e, f\}$. Moreover, the rule $\{a, b\} \rightarrow \{e, f\}$ is redundant with respect to $\{a\} \rightarrow \{e, f\}$.

2.3 Problem Definition

Based on the previous definition of redundancy and the definition of top-k sequential rule mining, we define the problem of top-k non redundant sequential mining as follows.

Definition 3. The *problem of mining top-k non-redundant sequential rules* is to discover a set L containing *k* sequential rules in a sequence database. For each rule $r_a \in L \mid conf(r_a) \geq minconf$, there does not exist a rule $r_b \in L \mid conf(r_b) \geq minconf \wedge sup(r_b) > sup(r_a)$, otherwise r_b is redundant with respect to r_a. Moreover, $\nexists r_c, r_d \in L$ such that r_c is redundant with respect to r_d.

3. THE TNS ALGORITHM

To address the problem of top-*k* non-redundant rule mining, we propose an algorithm named TNS. It is based on the same depth-first search procedure as TopSeqRules. The difference between TNS and TopSeqRules lies in how to avoid generating redundant rules. The next subsection briefly explains the search procedure of TopSeqRules.

3.1 The Search Procedure

To explain the search procedure, we introduce a few definitions. A rule $X \rightarrow Y$ is of size $p*q$ if $|X| = p$ and $|Y| = q$. For example, the size of $\{a\} \rightarrow \{e, f\}$ is 1*2. Moreover, we say that a rule of size p*q is larger than a rule of size $r*s$ if $p > r$ and $q \geq s$, or if $p \geq r$ and $q > s$. A sequential rule r is valid if $sup(r) \geq minsup$ and $conf(r) \geq minconf$.

This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

The search procedure takes as parameter a sequence database, an integer *k* and the *minconf* threshold. The search procedure first sets an internal *minsup* variable to 0 to ensure that all the top-k rules are found. Then, the procedure starts searching for rules. As soon as a rule is found, it is added to a list of rules L ordered by the support. The list is used to maintain the top-*k* rules found until now and all the rules that have the same support. Once *k* valid rules are found in L, the internal *minsup* variable is raised to the support of the rule with the lowest support in L. Raising the *minsup* value is used to prune the search space when searching for more rules. Thereafter, each time that a valid rule is found, the rule is inserted in L, the rules in L not respecting *minsup* anymore are removed from L, and *minsup* is raised to the support of the rule with the lowest support in L. The algorithm continues searching for more rules until no rule are found. This means that it has found the top-*k* rules in L. The top-*k* rules are the *k* rules with the highest support in L.

To search for rules, the search procedure first scans the database to identify single items that appear in at least *minsup* sequences. It uses these items to generate rules of size 1*1 (containing a single item in the antecedent and a single item in the consequent). Then, each rule is recursively grown by adding items to its antecedent or consequent (a depth-first search). To determine the items that should be added to a rule, the search procedure scans the sequences containing the rule to find single items that could expand its antecedent or consequent. The two processes for expanding rules are named left expansion and right expansion. These processes are applied recursively to explore the search space of sequential rules. Left and right expansions are formally defined as follows. A *left expansion* is the process of adding an item *i* to the left side of a rule X→Y to obtain a larger rule X∪{*i*}→Y. A *right expansion* is the process of adding an item i to the right side of a rule X→Y to obtain a larger rule X→Y∪{*i*}.

The search procedure described above is correct and complete for mining top-k sequential rules [10]. Moreover, it is very efficient because the internal *minsup* variable is raised during the search. This allows pruning large part of the search space instead of generating all sequential rules. This pruning is possible because the support is monotonic with respect to left and right expansions (see [13] for a proof).

3.2 Adapting the Search Procedure to Find Top-k Non-Redundant Rules

We now explain how we have adapted the search procedure of TopSeqRules to design a top-*k* non-redundant rule mining algorithm. These modifications are based on the following observation.

Property 1. During the search, if only non-redundant rules are added to L, then L will contain the top-k non-redundant rules when the search procedure terminates. **Rationale.** The search procedure is correct and complete for mining the top-*k* sequential rules [10]. If only non-redundant rules are added to L instead of both redundant and non-redundant rules, it follows that the result will be the top-k non-redundant rules instead of the top¬*k* sequential rules.

Based on this observation, we have aimed at modifying the search procedure to ensure that only non-redundant rules are added to *L*. This means that we need to make sure that every generated rule r_a is added to *L* only if $sup(r_a) \geq minsup$ and r_a is not redundant with respect to another rule. To determine if r_a is redundant with respect to another rule, there are two cases to consider.

The *first case* is that r_a is redundant with respect to a rule r_b that was generated before r_a. By the definition of redundancy (cf. Definition 2), if r_a is redundant with respect to r_b, then $sup(r_b) = sup(r_a)$. Because $sup(r_a) \geq minsup$, it follows that $sup(r_b) \geq minsup$, and that $r_b \in L$. Therefore, the first case can be detected by implementing the following strategy.

Strategy 1. For each rule r_a that is generated such that $\text{sup}(r_a) \geq minsup$, if $\exists\ r_b \in \text{L} \mid \text{sup}(r_b) = \text{sup}(r_a)$ and r_a is redundant with respect to r_b, then r_a is not added to L. Otherwise, r_a is added to L.

The second case is that r_a is redundant with respect to a rule r_b that has not yet been generated. There are two ways that we could try to detect this case. The first way is to postpone the decision of adding r_a to L until r_b is generated. However, this would not work because it is not known beforehand if a rule r_b will make r_a redundant and r_b could appear much later after r_a. The second way is to scan sequences to determine if item(s) could be added to r_a to generate a rule r_b such tat r_a redundant is redundant with respect to r_b. However, this approach would also be inefficient because the confidence is non-monotonic with respect to left/right expansions [4]. This means that it is possible that r_b may contain several more items than r_a. For this reason, it would be too costly to test all the possibilities of adding items to r_a to detect the second case.

Because the second case cannot be checked efficiently, our solution is to propose an approximate approach that is efficient and to propose a condition that can be verified to determine if the result is exact. The idea of this approach is the following. Each rule r_a that satisfy the requirements of Strategy 1 is added to L without checking the second case. Then, eventually, if a rule r_b is generated such that the rule r_a is still in L and that r_a is redundant with respect to r_b, then r_a is removed from L. This idea is formalized as the following strategy.

Strategy 2. For each rule r_b that is generated such that $\text{sup}(r_b) \geq minsup$, if $\exists\ r_a \in \text{L} \mid \text{sup}(r_b) = \text{sup}(r_a)$ and r_a is redundant with respect to r_b, then r_a is removed from L.

By incorporating Strategy 2, the algorithm becomes approximate. The reason is that each rule r_a that is removed by Strategy 2 previously occupied a place in the set L. By its presence in L, the rule r_a may have forced raising the internal *minsup* variable. If that happened, then the algorithm may have missed some rules that have a support lower than r_a but are non-redundant.

Given that the algorithm is approximate, it would be desirable to define a condition that would allow verifying if the result is exact. To achieve this, we propose to add a parameter that we name Δ that increase by Δ the number of rules k that is necessary to raise the internal *minsup* variable. For example, if the user sets k = 1000 and Δ = 100, it will now be required to have k + Δ = 1100 rules in L to raise the internal *minsup* variable instead of just k=1000. This means that up to 100 redundant rules can be at the same time in L and the result will still be exact. This latter observation is formalized by the following property.

This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

Property 2. If the number of redundant rules in L is never more than Δ rules, then the algorithm result is exact and the k rules in L having the highest support will be the top-k non redundant rules. **Rationale**. As previously explained, the danger is that too many redundant rules are in L such that they would force to raise *minsup* and prune part of the search space containing top-k non-redundant rules. If there is no more than Δ redundant rules at the same time in L and that k + Δ are needed to raise *minsup*, then redundant rules cannot force to raise *minsup*.

The previous property states a condition for verifying if Strategy 2 generates an exact result. Based on this property, an important question is "Would it be efficient to integrate a check for this condition in the algorithm?". The answer is that it would be too costly to verify that no groups of more than Δ redundant rules are present at the same time in L. The reason is that rules are only known to be redundant when they are removed by Strategy 2. Therefore, checking Property 2 would be very expensive to perform. For this reason, we choose to utilize the following weaker version of Property 2 in our implementation.

Property 3. If the number of redundant rules removed by Strategy 2 during the execution of the algorithm is less or equal to Δ, then the final result is exact and the first *k* rules of L will be the top-k non redundant rules. **Rationale**. It can be easily seen that if Property 3 is met, Property 2 is also met.

Property 3 can be easily incorporated in the algorithm. To implement this functionality, we have added a counter that is incremented by 1 after every rule removal from L by Strategy 2. Then, when the algorithm terminates, the counter is compared with Δ. If the counter value is lower or equal to Δ, the user is informed that the result is guaranteed to be exact. Otherwise, the user is informed that the result may not be exact. In this case, the user has the option to rerun the algorithm with a higher Δ value. In the experimental study presented in section 4 of this paper, we will assess the question of how to select Δ and if an exact result can be guaranteed for all datasets.

The previous paragraphs have presented the main idea of the TNS algorithm. Due to space limitation, we do not provide the pseudo-code of TNS. But the Java source code of our implementation can be downloaded freely from http://_____.

Note that in our implementation, we have added a few optimizations that are used in TopSeqRules [10] and that are compatible with TNS. The first optimization is to try to generate the most promising rules first when exploring the search space of sequential rules. This is because if rules with high support are found earlier, the algorithm can raise its internal *minsup* variable faster to prune the search space. To perform this, an internal variable R is added to store all the rules that can be expanded to have a chance of finding more valid rules. This set is then used to determine the rules that are the most likely to produce valid rules with a high support to raise *minsup* more quickly and prune a larger part of the search space [10]. The second optimization is to implement L and R with data structures supporting efficient insertion, deletion and finding the smallest element and largest element. In our implementation, we used a Red-black tree for L and R. A red-black tree guarantees a O(log(n)) worst-case time cost for the four operations [11] .

4. EXPERIMENTAL EVALUATION

We have carried several experiments to assess the performance of TNS and to compare its performance with TopSeqRules under different scenarios. For these experiments, we have implemented TNS in Java. For TopSeqRules, we have obtained the Java implementation from their authors. All experiments were performed on a computer with a Core i5 processor running Windows 7 and 2 GB of free RAM. To perform all memory measurements, the standard Java API was used. Experiments were carried on three real-life datasets commonly used in the data mining literature. Table 1 summarizes the characteristics of these datasets. *BMSWebview1* and *Kosarak* were downloaded from http://fimi.ua.ac.be/data/. *Snake* was obtained from the authors of [20]. For Kosarak, we only used the first 10,000 sequences as in [10] to make the experiments faster.

Table 1. Datasets characteristics

Datasets	Number of sequences	Number of distinct items	Avg. item count per sequence	Type of data
BMS	59,601	497	2.5 (σ = 4.85)	click-stream from web store
Sign	730	310	93.39 (σ = 4.59)	language utterances
Kosarak	990,000	21,244	7.97 (σ = 0.89)	protein sequence

Experiment 1: **What is the percentage of redundant rules?** The goal of the first experiment was to assess the percentage of rules returned by TopSeqRules that are redundant to determine how important the problem of redundancy is in top-k sequential rule mining. For this experiment, we ran TopSeqRules on the four datasets with *k* = 1000 and *minconf* = 0.8. We chose *minconf* = 0.8 and *k* = 1000 because these values are plausible values that a user could choose. The only exception is BMSWebview1, where we set *minconf* = 0.7. This is because if we set *minconf = 0.8* with this dataset, TopSeqRules run out of memory because of the very large search space, as noted in [10].

We then examined the rules returned by TopSeqRules to detect redundant rules. The results for BMSWebView1, Snake and Kosarak are that respectively, 66.79 %, 79.59 % and 9.37 % of the rules returned by TopSeqRules are redundant. These results indicate that eliminating redundancy in top-k sequential rule mining is a major problem.

Experiment 2: How many rules are discarded by Strategy 1 and Strategy 2? The second experiment's goal was to observe how many rules were discarded by Strategy 1 and Strategy 2 of TNS for each dataset. To perform this study, we ran TNS with *minconf* and *k* set as in Experiment 1 and Δ = 0. We then recorded the number of discarded rules. Results are shown in Table 2. From this experiment, we can see that the number of discarded rules is large for all datasets and that it is especially high for dense

This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

datasets (Snake and Kosarak) because they contain more redundant rules than sparse datasets (e.g. BMSWebView1).

Table 2. Rules discarded by each strategy for *minconf* = 0.8, *k*=2000 and Δ = 0

Dataset	# rules discarded by Strategy 1	# rules discarded by Strategy 2
BMSWebView1	152	190
Snake	6721	5923
Kosarak	1314	982

Experiment 3: Can TNS guarantee an exact result by using Δ? The next experiment consisted of using the Δ parameter to see if an exact result could be guaranteed by using Property 3. For this experiment, we ran TNS with *k* and *minconf* set as in the first experiment, and we set Δ to values slightly larger than the number of rules discarded by Strategy 2 in Experiment 2. By setting Δ to values slightly larger, the goal is to attempt to get an exact result by Property 3.For BMSWebView1, we therefore set Δ = 300. The total runtime was 18.37 s and the maximum memory usage was 1.2 GB. The number of rules eliminated by Strategy 1 and Strategy 2 was respectively 244 and 279. Because 279 < Δ, the result is exact.

For Kosarak, we set Δ = 1000. The total runtime was 21.53 s and the maximum memory usage was 1.1 GB. The number of rules eliminated by Strategy 1 and Strategy 2 was respectively 10862 and 4689. Because 4689 > Δ, the result is not guaranteed to be exact. Finally, we set Δ = 6000 for Snake. For this dataset, TNS ran out of memory because the search space becomes very large for large Δ values. It is therefore not possible to guarantee an exact result for Snake.

From this experiment, we conclude that for sparse datasets (BMSWebView1), the Δ parameter can guarantee an exact result, and for moderately dense (Kosarak) and dense datasets (Snake), it is not possible to guarantee an exact result. We performed this experiment with other values of *k* and Δ (not presented here) and got similar results. The reason why the result cannot be guaranteed exact for dense datasets is that if we raise Δ for these datasets, the number of rules removed by Strategy 2 also increases. Therefore, this latter number remains larger than Δ and Property 3 cannot be satisfied. For example, consider the Kosarak dataset. When Δ=0, Strategy 2 eliminates 982 rules. If we raise Δ to 1000, Strategy 2 eliminates 4689 rules. If we further raise Δ to 1500, Strategy 2 discards 6295 rules.

But even if the *k* rules returned by TNS cannot always be guaranteed exact for all datasets, TNS always guarantee that the *k* rules returned are non-redundant, which is an important advantage over TopSeqRules.

Experiment 4: Performance comparison with TopSeqRules. The fourth experiment consisted of comparing the performance of TNS with TopSeqRules in terms of execution time and memory requirement. The parameters *minconf* and *k* were set as in Experiment 1 and Δ was set to 0. The execution times and maximum memory usage of both algorithms are presented in Table 3. The results show that there is an extra cost for using TNS compared to TopSeqRules. This is what we expected for two reasons. First, performing the checks for Strategy 1 and Strategy 2 require additional calculation. Second, TNS has to generate more rules because some rules are discarded by the strategies. But overall, we observed that the performance of TNS is very close to TopSeqRules for sparse and moderately dense datasets (Kosarak and BMSWebView1).

Experiment 5: The influence of Δ on the execution time and memory usage of TNS. The fifth experiment's goal is to study the influence of Δ on the performance of TNS in terms of execution time and maximum memory usage. For this experiment, we set *minconf* and *k* as in the previous experiments and ran TNS on each dataset while varying Δ from 250 to 1000. Furthermore, we also ran TopSeqRules and utilized its performance as a baseline for evaluating TNS. Results of this experiment are shown in Table 4 and Table 5.

Table 3. Performance comparison for Δ= 0

Datasets	Runtime (s)		Maximum Memory Usage (GB)	
	TNS	TopSeqRules	TNS	TopSeqRules
BMSWebView1	17.4	16.5	1.17	1.13
Snake	3.5	0.9	0.19	0.05
Kosarak	11.6	11.0	0.84	0.72

Table 4. Impact of varying Δ on runtime (s)

Datasets	TNS				TopSeqRules
	Δ=250	Δ=500	Δ=750	Δ=1000	
BMS	17.48	19.93	24.56	29.37	16.5
Snake	4.53	5.09	6.26	7.45	0.9
Kosarak	13.33	16.68	19.57	20.69	11.0

Table 5. Impact of varying Δ on memory usage

Datasets	TNS				TopSeqRules
	Δ=250	Δ=500	Δ=750	Δ=1000	
BMS	1.21	1.36	1.48	1.55	1.13
Snake	0.24	0.28	303.32	0.30	0.05
Kosarak	0.87	0.93	1.11	1.20	0.72

Our first observation is that the cost of using Δ in terms of memory usage and execution time is reasonable for all datasets. Furthermore, when the resulted are plotted on a chart (not shown here), we can see that the algorithm execution time and memory usage grows more or less linearly with Δ for all datasets.

Experiment 6: Scalability of TNS. Next, we ran TNS and TopSeqRules on the three datasets while varying the number of sequences in each dataset to compare their scalability. We set *k* and *minconf* as in the first experiment. We varied the database size by using 70%, 85 % and 100 % of the sequences in each dataset. Results are shown in Figure 2. Globally we found that for all datasets, the execution time and memory usage varied more or

This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

less linearly for TNS and TopSeqRules. This shows that TNS has excellent scalability that is similar to TopSeqRules.

Discussion. Our conclusion from these experiments is that TNS has excellent performance and scalability, which is very close to the performance of TopSeqRules. But TNS also provides the benefit of eliminating redundancy. Other experiments not presented in this paper due to space limitations such as varying the confidence have also confirmed this.

5. CONCLUSION

Two important problems with classical sequential rule mining algorithm are that (1) it is usually difficult and time-consuming to select the parameters to generate a desired amount of rules and (2) there can be a large amount of redundancy in results. Previously, these two problems have been addressed separately. In this paper, we have addressed them together by proposing an efficient algorithm named TNS for mining the top-k non-redundant sequential rules. To ensure that TNS is efficient, TNS rely on an approximate approach that can guarantee exact result under certain conditions. We have compared the performance of TNS with TopSeqRules on three real datasets and found that TNS has excellent performance and scalability that are comparable to TopSeqRules. But TNS provides the benefit of eliminating redundancy. This is a key improvement because as shown in our experimental study up to 79 % of rules generated by TopSeqRules are redundant. Source code of TNS can be downloaded from http://www.philippe-fournier-viger/spmf/ .

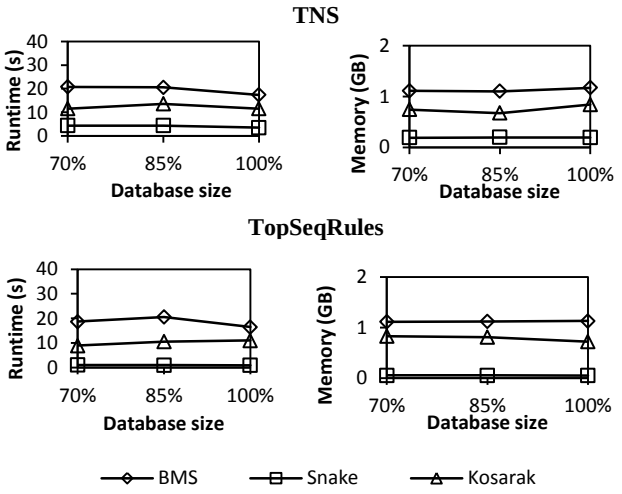

Fig. 2. Influence of the number of sequences

6. REFERENCES

[1] Laxman, S. and Sastry, P. A survey of temporal data mining. *Sadhana* 3 (2006), 173-198.

[2] Agrawal, R. and Srikant, R. Mining Sequential Patterns. In *Proc. Int. Conf. on Data Engineering* (Taipei, Taiwan, March 6-10, 1995), 3-14.

[3] Pei, J., Han, J. et al. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Trans. Knowledge and Data Engineering*, 16, 10 (2004), 1-17.

[4] Fournier-Viger, P., Nkambou, R., Tseng, V. S. RuleGrowth: Mining Sequential Rules Common to Several Sequences by Pattern-Growth. In *Proc. 27th Symposium on Applied Computing* (SAC 2011) (Taichung, Taiwan, March 21-24, 2011), ACM Press, New York, NY, 2011, 954-959

[5] Lo, D., Khoo, S.-C., Wong, L.. Non-redundant sequential rules - Theory and algorithm. *Information Systems*, 34, 4-5 (2009), 438-453.

[6] Zhao, Y., Zhang, H., Cao, L., Zhang, C. and Bohlscheid, H. Mining Both Positive and Negative Impact-Oriented Sequential Rules From Transactional Data. In *Proc. 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (PAKDD 2009), Springer, 2009, 656-663.

[7] Das., G., Lin, K.-I., Mannila, H., Renganathan, G., and Smyth, P. Rule Discovery from Time Series. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining* (New York, USA, August 27-31, 1998), 16-22.

[8] Harms, S. K., Deogun, J. and Tadesse, T. 2002. Discovering Sequential Sequential Rules with Constraints and Time Lags in Multiple Sequences. In *Proc. 13th Int. Symp. on Methodologies for Intelligent Systems* (Lyon, France, June 27-29, 2002), pp. .373-376.

[9] Hamilton, H. J. and Karimi, K. The TIMERS II Algorithm for the Discovery of Causality. In *Proc. 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (Hanoi, Vietnam, May 18-20, 2005), 744-750.

[10] Fournier-Viger, P. and Tseng, V. S. Mining Top-K Sequential Rules. In *Proc. of the 7th Intern. Conf. on Advanced Data Mining and Applications* (ADMA 2011) (Beijing, China, December 17-19, 2011). Springer, 2011, 180-194.

[11] Hsieh, Y. L., Yang, D.-L. and Wu, J. Using Data Mining to Study Upstream and Downstream Causal Relationship in Stock Market. In *Proc. 2006 Joint Conference on Information Sciences* (Kaoshiung, Taiwan, October 8-11, 2006).

[12] Mannila, H., Toivonen and H., Verkano, A.I. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1, 1 (1997), 259-289.

[13] Fournier-Viger, P., Faghihi, U., Nkambou, R. and Mephu Nguifo, E. CMRules: An Efficient Algorithm for Mining Sequential Rules Common to Several Sequences. In *Proceedings 23th Intern. Florida Artificial Intelligence Research Society Conference* (Daytona, USA, May 19-21, 2010), AAAI Press, 410-415.

[14] Agrawal, R., Imielminski, T. and Swami, A. Mining Association Rules Between Sets of Items in Large Databases. In: *Proc. 13th ACM SIGMOD Intern. Conf. on Management of Data*, (New York, USA, June 23-28), 1993, 207-216.

[15] Bastide, Y., Pasquier, N., Taouil, R., Lakhal, L. and Stumme, G. Mining minimal non-redundant association rules using frequent closed itemsets. In *Proc. 6th Intern. Conf.* on Deductive and Object-Oriented Databases (Montreux, Switzerland, December 8-12, 1997), Springer, 2000, 972-986.

This PDF is an unpublished extended 6 pages version of the 3 pages paper published at:

Fournier-Viger, P., Tseng, V. S. (2013). **TNS: Mining Top-K Non-Redundant Sequential Rules.** Proc. 28th Symposium on Applied Computing (ACM SAC 2013). ACM Press, *to appear*.

[16] Gasmi, G., BenYahia, S., Nguifo, E. M. and Slimani, Y., A new informative generic base of association rules, In *Proc. 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (PAKDD 2009), Springer, 2005, 81-90.

[17] Cherif, C. L., Bellegua, W., Ben Yahia, S. and Guesmi, G. VIE_MGB: A Visual Interactive Exploration of Minimal Generic Basis of Association Rules. In *Proc. of the 3rd Intern. Conf. on Concept Lattices and Application* (CLA 2005) (Olomouc, Czech Republic, September 7–9, 2005), CEUR, 2005, 179-196.

[18] Kryszkiewicz, M. Representative Association Rules and Minimum Condition Maximum Consequence Association Rules. In *Proc. of 2nd European Symposium on Principles of Data Mining and Knowledge Discovery, (PKDD 98)* (Nantes, France, September 23-26, 1998), Springer, 361-369.

[19] Cormen, T. H., Leiserson, C. E., Rivest, R. and Stein, C. *Introduction to Algorithms, 3rd ed.* MIT Press, Cambridge, MA, 2009.

[20] Jonassen, I., Collins, J.F. and Higgin, D.G. 1995. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4, 8 (1995), 1587-1595.