applied sciences

MDPI

Article

Performance Evaluation of Sequential Rule Mining Algorithms

Amira Abdelwahab [1,2,*] and Nesma Youssef [3]

[1] Information Systems Department, College of Computer Sciences and Information Technology (CCSIT), King Faisal University, P.O. Box 400, Al-Ahsa 31982, Saudi Arabia
[2] Information Systems Department, Faculty of Computers and Information, Menoufia University, Shibin Al Kawm 32511, Egypt
[3] Department of Information System, Sadat Academy for Management Science, Cairo 12411, Egypt; nesmayousef1811@gmail.com
* Correspondence: a.ahmed@kfu.edu.sa

Abstract: Data mining techniques are useful in discovering hidden knowledge from large databases. One of its common techniques is sequential rule mining. A sequential rule (SR) helps in finding all sequential rules that achieved support and confidence threshold for help in prediction. It is an alternative to sequential pattern mining in that it takes the probability of the following patterns into account. In this paper, we address the preferable utilization of sequential rule mining algorithms by applying them to databases with different features for improving the efficiency in different fields of application. The three compared algorithms are the TRuleGrowth algorithm, which is an extension sequential rule algorithm of RuleGrowth; the top-k non-redundant sequential rules algorithm (TNS); and a non-redundant dynamic bit vector (NRD-DBV). The analysis compares the three algorithms regarding the run time, the number of produced rules, and the used memory to nominate which of them is best suited in prediction. Additionally, it explores the most suitable applications for each algorithm to improve the efficiency. The experimental results proved that the performance of the algorithms appears related to the dataset characteristics. It has been demonstrated that altering the window size constraint, determining the number of created rules, or changing the value of the minSup threshold can reduce execution time and control the number of valid rules generated.

Keywords: sequential rule mining; non redundant sequential rules; TRuleGrowth; top-k non redundant rules; closed sequential patterns

Citation: Abdelwahab, A.; Youssef, N. Performance Evaluation of Sequential Rule Mining Algorithms. *Appl. Sci.* **2022**, *12*, 5230. https://doi.org/10.3390/app12105230

Academic Editors: Stefanos Ougiaroglou and Dionisis Margaris

Received: 18 April 2022
Accepted: 17 May 2022
Published: 21 May 2022

1. Introduction

There is a fundamental issue in extracting useful information from the temporal relations in large sequence datasets. It helps a user to acquire useful knowledge for making a prediction. Many methods are emerging for finding the temporal relations in datasets. One of the foremost popular methods is mining sequential patterns to discover, as often as possible, patterns in sequence datasets [1,2]. Mining sequential patterns rely on one measure called support measure. The support measure means the number of presence items in a dataset. It can be misleading and inadequate for making a prediction. Mining sequential rules is an extension of sequential patterns mining that addresses the previous problem by considering additional measures [3,4].

Mining sequential rules take another measure besides the support into consideration, called the confidence measure, which means that the probability of executing the next pattern is calculated. There are many challenges for mining sequential rules, such as classifying similar rules differently. Additionally, some rules have been discovered that lose their importance when appearing separately. Therefore, particular rules are losing their use in predicting. All previous reasons impact the generation of a large number of redundant rules that affect the efficiency of the mining process. Numerous analysts have proposed upgraded methods of sequential rule mining (SRM) to decrease redundant rules and

progress the proficiency of these algorithms. SRM is divided into two categories: partially ordered and standard sequential rules. The proposed enhanced algorithms concentrate on two directions:

The first type is a common type for mining the sequential rules named standard. It helps to enhance efficiency through the process of mining. The mining process consists of two procedures; the first is mining sequential patterns that appear frequently. The second is producing sequential rules that rely on the first procedure. Therefore, numerous researchers give consideration to the first procedure to enhance the performance through disposing of grouping that does not affect the ultimate result. Mining closed sequential patterns is an example of the previous idea. It helps to produce rules based on more compact data. In this study, we make a reference to this type with the non-redundant dynamic bit-vector algorithm.

The second type is the newest type for mining the sequential rules named partially ordered rules. It helps to enhance efficiency by extending the mining of sequential rules algorithms by adding constraints or by finding a specified number of the most visit rules in a database. In the partially ordered mining, there is no arrangement between items in the prior and posterior sides. The pattern growth approach is applied to discover all rules incrementally. There are improved algorithms that acknowledge an extra constraint to enhance the overall performance. The TRuleGrowth algorithm adds a window size parameter. It makes a difference to decrease the number of rules produced, as it diminishes the runtime, and diminished disk space is a prerequisite to store generating rules, so that users are able to analyze the rules in an easy manner. Additionally, we have another approach to produce non-redundant rules. We find the most frequent rules (the top-k) that achieved the minConf threshold. Therefore, we have only two parameters, K and minConf. Like the TNS algorithm, we did not take the minSup into consideration due to the difficulty of determining the valid values that suit each dataset's features.

In this paper, we present a broad consider of two sorts of SRM: partially ordered and standard sequential rules. We study three algorithms named TRuleGrowth, TNS, and non-redundant with a dynamic bit-vector algorithm. All these algorithms generate non-redundant rules and we compare the results of their implementation to decide the most reasonable areas for each of them. We assess the final results according to these criteria: runtime, number of produced rules, and memory utilization.

2. Literature Review

Numerous research has been suggested to enhance the mining process of sequential patterns. The primary obstacle in mining sequential patterns is producing unessential sequential patterns when setting the support measure with a very low value. Mining the sequential rules is an alternative to sequential patterns that assist the users in having knowledge of sequence items for making a prediction. It has been found in numerous zones like electronic learning [5], manufacturing simulation [6], the analysis of customer behavior [7], and decision systems [8].

The primary algorithm proposed for mining the sequential rules by Mannila and Verkano is studying the sequence behavior for the prediction process. It helps in finding all items that occurred as often as possible in a sequence dataset [9]. Then, most research has been proposed to produce sequential patterns that appear frequently and remove repetitive rules within the following stage of the mining process. They had to check the database numerous times to find the support of each itemset that induces minimum complexity and extra cost like RuleGen algorithm [10]. After that, the researchers have found rules with no consideration of their arrangement; refer to partially ordered sequential rules (POSR). It stands up on those items in the predecessor, and forerunner sides are unarranged. Two primary algorithms for the POSR are named CMRule and CMDeo [11,12]. The CMRule is the baseline algorithm that expels the temporal information and generates rules that accomplish the support threshold. It relies on the produced number of sequential rules that cause ineffectual performance [13]. The second baseline algorithm is CMDeo, which

acts as more proficient than the CMRule. It discovers all substantial rules size $1 * 1$ by expanding both sides of the rules. To address the previous obstacles, RuleGrowth has been proposed. It can extend rules amid guarantees as it was necessary for rules in sequence datasets [14–16].

Various algorithms have been developed based on the prefix tree to realize superior efficiency. The CNR, IMSR, and MNSR algorithms rely on enhancing the rules by eliminating non-redundant rules [17–19]. They sort the sequences according to the support value ascendancy before producing rules. In this way, it diminishes the scan's number and minimizes complexity to $o(n^2)$.

TRuleGrowth is an extension of RuleGrowth. It has been developed to control the maximum consecutive items by applying an additional constraint called window size. This constraint helps to produce much fewer number of rules. So, it enhances the performance due to diminishing the request space used to store the generated rules [20–23].

The researchers proposed the Top-k sequential rule mining to overcome the problem of the difficulty of determining the valid minSup value that suits each database features [24,25]. It depends on two parameters: k is the number of rules that users want to generate, and the minConf that is easy to determine due to user satisfaction. The TopSeqRules is the first algorithm that addresses the Top-k sequential rule mining. It depends on the same strategy as the RuleGrowth algorithm. It integrates the two procedures to expand the left and the right sides of the rule with the general procedure for mining the top-k pattern. Researchers enhanced many algorithms to reduce disk space and generate interesting sequential rules. The Top-k non-redundant sequential rule (TNS) utilizes the TopSeqRules for mining the top-k to eliminate the redundancy on the generated rules [26,27].

A proficient algorithm named non-redundant with dynamic bit-vector has been suggested. It is used to remove unnecessary rules early through utilizing dynamic bit-vector with pruning techniques, so that it improves the performance by reducing the execution time and memory utilization [28–30].

3. Methods of Applying Comparative Algorithms

There are two categories for mining the sequential rules. The first category is called standard sequential rules that discover relations between two patterns that act sequentially. It depends on two measures, support and confidence, which are selected by the user. It exists in many algorithms like RuleGen, IMSR (Improved Mining Sequential Rule), and NRD-DBV (Non-Redundant-Dynamic Bit Vector) algorithm. It produced the result only in integer's format, and it proves its efficiency in creating an imperative choice or expectation. The second category is called POSR, which is the newest type of SRM that discovers relations between two unarranged item-sets. It does not care about the relations between the predecessor and forerunner of rules. The POSR is based on the pattern growth approach, which starts with two items and expands the rules one element at a time recursively. It can expand the mining process by including extra constraints to improve the performance of the general rules like the TRuleGrowth algorithm. Therefore, it can control the number of generated rules, which helps in saving more space and being more specific. The POSR can also satisfy the user's desire by producing a predefined number of the rules like the Top-k algorithm. Each type has its benefits to generate the non-redundant rules. In this paper, we present the most frequent recent algorithms that help in discovering the non-redundant rules by applying them to different dataset's features to reach the best efficiency.

3.1. The Fundamental Operations of the TRuleGrowth Algorithm

The TRuleGrowth algorithm is a type of POSR that does not need to arrange its items. It is an extension of the RuleGrowth algorithm that proves its efficiency compared to the last one. It is similar to RuleGrowth based on an approach called pattern growth. It performs incrementally, begins with two elements, and grows one item at a time by expanding the right and the left side of the rule. The TRuleGrowth utilizes an extra constraint called window size to control the number of rules generated. It helps in producing more accurate

results and saving more space by generating much fewer sequential rules. The value of a window size constraint can be an obstacle when its value increases. The best solution, in this case, is using the RuleGrowth algorithm instead of TRuleGrowth to eliminate the high computations due to emerging large number of rules.

The construction of the TRuleGrowth algorithm is formed by three stages. The primary stage is transforming the database to the sequential list, and then setting the support value. The moment stage is creating a rule size $1 * 1$ and executing two procedures to extend the rule on the right and the left sides. The final stage is determining the window size and the confidence value, and after that, testing the legitimacy of the rules to produce the non-redundant sequential rules, as shown in Figure 1.

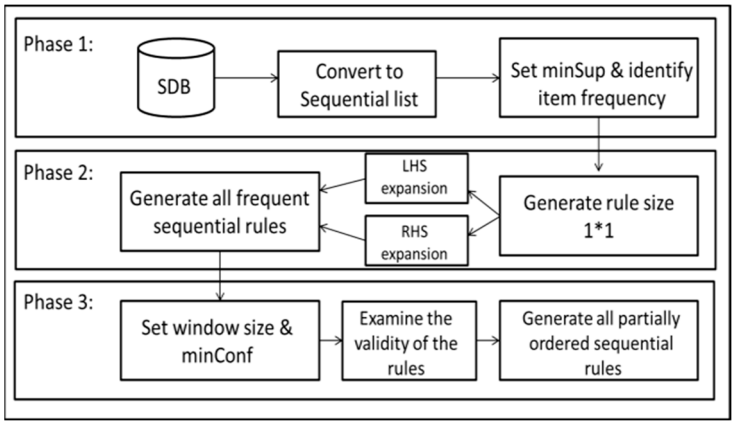

Figure 1. The RuleGrowth algorithm framework based on [31].

The TRuleGrowth Application Algorithm

This algorithm begins with scanning the dataset once to discover each item on each side. At that point, it distinguishes every item achieving the support value threshold to produce all accurate rules with size $1 * 1$. To computing sides ($x \rightarrow z$) and sides ($z \rightarrow x$), it filters the dataset to generate the first sequence and calculate each duo of items individually. Then we compute the support value by partitioning $|$sides $(x \rightarrow z)| / |s|$. If the support values of the item are not lower than the minSup, then the two strategies have been executed to grow each side of the rule [14]. We have guaranteed that the value of the sliding window has applied for all the rules. It saves all events of each item and shows their position by including that item. For the events of c in the sequence {a,c}, {b,c}, {c}, and {a,b,e} are 1, 2, and 3. At that point, utilize the hash table for searching each item found before item z and items found before x in a sequence dataset. It can be done inside a window size parameter, as appeared in Appendix A Algorithm A1 [22].

The operation of the two procedures is accomplished in the following steps: begin by establishing a hash table and assign the null value. Then, check the item-set to dispose of all items that do not accomplish the sliding window predetermined value. If the measure of 'hash x' = size 'x', then include each item $z \in y \cap n$ in the hash table. In case 'hash table' < 'x' at the point, contain each item with the position of n as DB ϵ x $\cap$ n. At last, if 'hash y' = size 'y' and size 'hash x' = size 'x', include side to accessible side ($y \cup \{z\} \rightarrow x$) for each item $z \notin y \cup x$ that contains ID, to begin with an item of 'x' inside window size.

An extra parameter called sliding window can be included which has the following characteristics:

1. Controlling the number of produced rules, so less space and time is required to search the desired sequential rules.
2. Reducing the amount of memory space required to store the generated rules by developing fewer rules. Therefore, it gives the facility to analyze the output to the user.

3. Favoring its significance in practical for temporal transactions like analyses data for shares market.

3.2. The Operation of the Top-k Non-Redundant Sequential Rule (TNS)

The TNS algorithm adopts TopSeqRules to mine the Top-k rules after eliminating redundancy of sequential rule. It depends on the same search procedure of the RuleGrowth algorithm. The TopSeqRules has two parameters: the number of rules that is determined by the user needs (K), and the minConf threshold that also satisfies the user requirements. It also utilizes three main variables; the first is minSup that is set to initial value = 0 and raises with regard to the number of rules generated. The second variable is L, which helps to perform Top-k rules by keeping each item that achieved the threshold value. The third variable is R having the rule with the maximum support values that helps in choosing the most candidate rules.

The TNS Algorithm Implementation

The TNS algorithm firstly scans the dataset only once to identify each item, as seen in Figure 2. It considers an integer parameter K, minConf value, and initializes a value of minSup with zero. Then it generates a rule recursively by growing the valid rule size $1 * 1$. After that, it performes two procedures to extend the right and the left side of the rule to generate the Top-k sequential rules. It is followed by removing all redundant rules which means that if r_a is equal to r_b with the same support, we will absorb r_a. Then it verifies the result by setting parameter delta Δ with a value higher than the all removed rules, so the Top-k non-redundant rules will be generated as shown in Algorithm A2 [24].

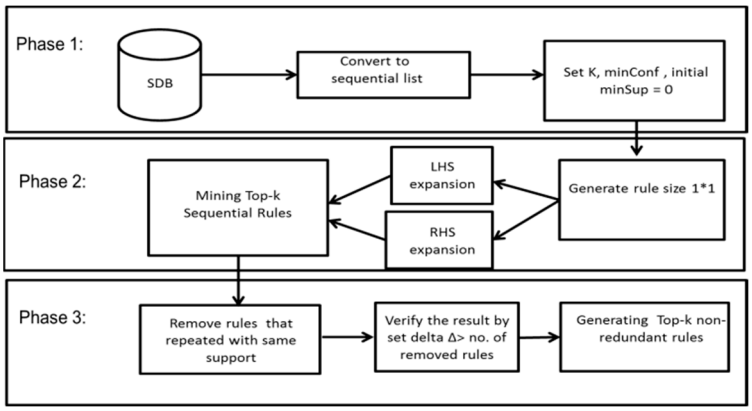

Figure 2. Framework of TNS algorithm.

Applying the TNS algorithm solves the trouble of generating either a large number of sequential rules or fewer numbers which may lose valuable information. It also addresses the challenge of generating redundant sequential rules. It has more advantages as:

1. It saves more time by depending on another parameter to generate the wanted number of sequential rules. Therefore, it does not rely on minSup value that is difficult to be determined regarding database features.
2. Removes the redundant rules and verifies the result by the delta parameter that was set higher than the removed sequential rules.

3.3. The Operation of the Non-Redundant with Dynamic Bit Vector Algorithm

The main idea of the NRD-DBV algorithm is to mine more compact data without losing any information or distorting the final result. It is based on mining closed patterns that depend on a vertical format. It performs efficiently in large datasets since it generates a smaller number of rules. Additionally, it utilizes pruning techniques to enhance overall performance.

The NRD-DBV Application Algorithm

The implementation of the NRD-DBV is performed in the following steps, as shown in Figure 3:

1. Transforming the dataset to a bit vector by removing zeroes from the front and the end of the sequence. It is followed by using a dynamic bit vector structure to determine the position of each item.
2. Setting the support threshold to find all items that achieve the threshold value and storing them in a prefix tree structure as the parent node of the tree.
3. Performing the check downward closer checking means removing all prefixes that do not expand the rule. For instance, item c ordinarily happens after item b with a similar support value of b; at that point c ought to be ingested such as A(BC) = 60% and A(BC)DE = 60%. By applying this approach, we will remove sequence A (BC).
4. Setting all found frequent sequences as a sub-node and analyzing it, whether prefix generator or closed patterns.
5. Applying the sequence extension technique for every sub-node by applying two methods. The first is called item extension. It expands the patterns by including new items at the final item-set by regarding that item as larger than the last element of the item-set. The second one is called a sequence extension item-set expansion by inserting the item as a new item-set after the final existing, as displayed in Algorithms A3 and A4.
6. Producing the NRD-DVB by applying a condition to halt producing rules that do not achieve the confidence threshold. This is indicated as, in the case $((\frac{\sup(Sequence\ of\ Sn)}{\sup(Sequence\ of\ pre)}) \geq minconf)$, the point nr-SeqRule, even with nr-SeqRule union with R. Additionally, cease to produce the rules for the sub-nodes. Conjointly, cease producing the rules when support (n1) < support (n) and if $\frac{sup(n1)}{sup(n2)} <$ minConf, then $\frac{sup(n2)}{sup(n)} <$ minConf.

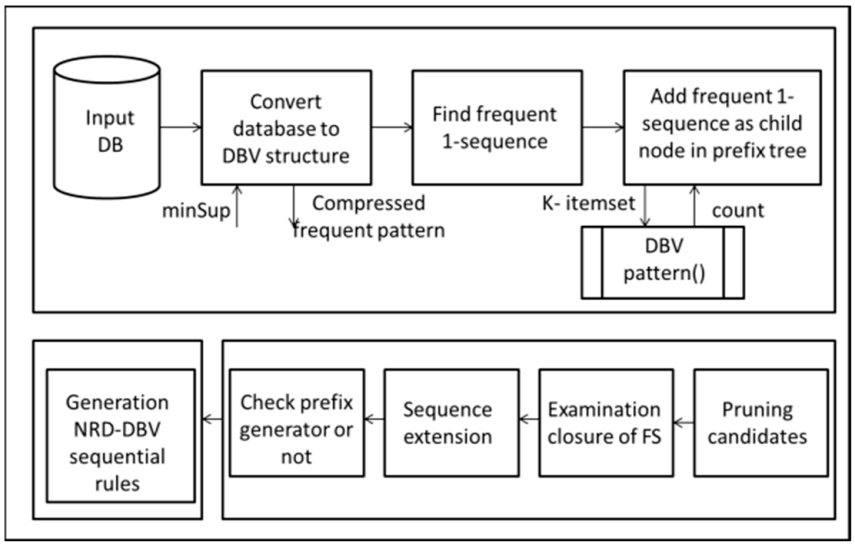

Figure 3. The NRD-DBV algorithm framework based on [31].

The NRD-DVB algorithm helps to compact the dataset by eliminating any super sequence with a similar support value to the root. Therefore, it is more proficient, as:

1. It diminishes the memory utilization, and the runtime demanded to mine large sequence datasets.
2. Furthermore, it embraces the prefix tree to store all sequences that produce more proficient non-redundant rules.

4. Experiments and Results

Experiments were processed to assess the runtime, memory usage for each algorithm, and the number of generated rules. Additionally, the influence of the minSup on both NRD-DBV and TRuleGrowth algorithms was measured and the TNS algorithm was compared with the other two algorithms after producing the same number of rules for each of them. Three algorithms were implemented on a PC with an Intel Core i5 2.3 GHz and free RAM running with 6.58 GB. We utilized Python language encoded on Jet Brains PyCharm.

Four real databases with diverse characteristics were executed to assess the results obtained from SPMF. The first dataset is named BMSwebview1 (Gazelle) [32] which consists of 59,601 clickstream data sequences from an e-commerce website. There are 497 separate items and the foremost vital issue that distinguishes their items reiterated in a seldom manner. The second database is a database of a sign dialect articulation consisting of 800 sequences transcript from recordings [33]. It includes 267 particular sets of items. The Korsarak is the third database; it is regarded as one of the biggest sequential databases. It has 990,000 sequences of click-stream information from a Hungarian news entrance. It incorporates 41,270 items. Because of the trouble connected on TRuleGrowth that caused the overhead limitation to be surpassed, we used a subset of the global database of Korsarak that contained only 25,000 items, which include non-reputation of position on the news. The BMSWebView2 (Gazelle) is the fourth database; it utilizes the information set within the KDD CUP 2000. It includes click-stream data of e-commerce in range 77,512 with 3340 particular items with an average length of 4.62. All experiments have been analyzed regarding to runtime, generated rules, and memory usage. The impact of the minSup on the previously stated criteria is studied.

The utilized datasets have different features useful for our comparison. In the first dataset (BMSwebview1), due to the variance of sequences, we selected a lower minSup value, which means that items are not repeated frequently in the dataset. We did not have any rules during the mining process while setting minSup like other attempts. In all studies, the minConf threshold was set to 0.5 for all states. The parameters' values were determined after several preliminary experiments to achieve the most preferable results.

4.1. Compared the TRuleGrowth with the NRD-DBV Algorithm

From the first dataset (BMS web view1), it is clear that the run time increased with decreasing minSup. It is a reversed relationship between the minSup and the runtime. The clarification of the relationship appears with raising values of window size and in the NRD-DBV algorithm. When the window size constraint esteem diminishes, we take note that it takes less time and has fewer number of rules. That is due to eliminating the complex computations of produced rules. It is similar to the outcome for the generated rules. Additionally, there is moreover a reversed relationship between the minSup and the generated rules' number. With the high esteem of window size constraint in the TRuleGrowth algorithm, there is an additional increment in the number of rules produced. As the window size esteem diminishes and increments minSup esteems simultaneously, the generated rules of the NRD-DBV are adjacent to the number of generated rules from the TRuleGrowth algorithm. Figures 4 and 5 show the runtime, no. of rules with minConf = 0.5% and minSup set from (0.09 to 0.06%), and Figure 6 shows the memory usage.

Additionally, it is found that when minimizing the value of window size, the time and number of sequence rules required are decreased. This is because producing sequential rules does not need very much computing.

On the other hand, in the sign dataset, we unrestricted with a low value for minSup. As before, we observed that the runtime increased with decreasing minSup value. The gap between the two parameters increased, especially for the TRuleGrowth algorithm when assigning lower values of window size constraint. This is a result of generating a large number of rules at smaller minSup values and higher window size value. It is an opposite relation between minSup and the generated rules. Figures 7 and 8 show the runtime,

number of rules with minConf = 0.5% and minSup set from (0.2% to 0.8%), and Figure 9 presents the memory usage.

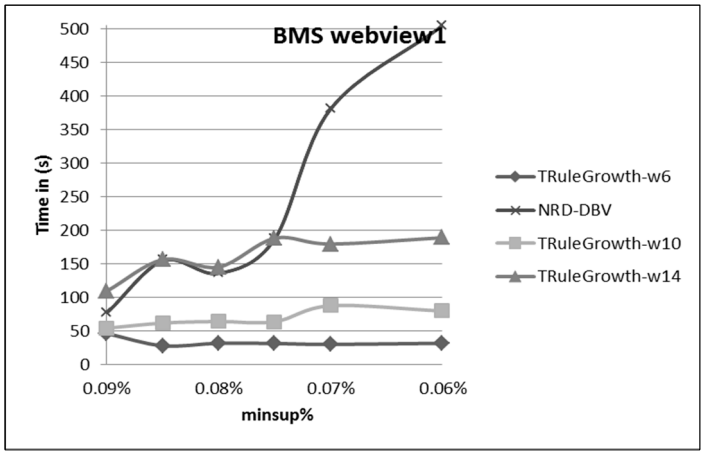

Figure 4. Comparison of the runtime of the TRuleGrowth with the NRD-DBV for (BMS webview1) dataset.

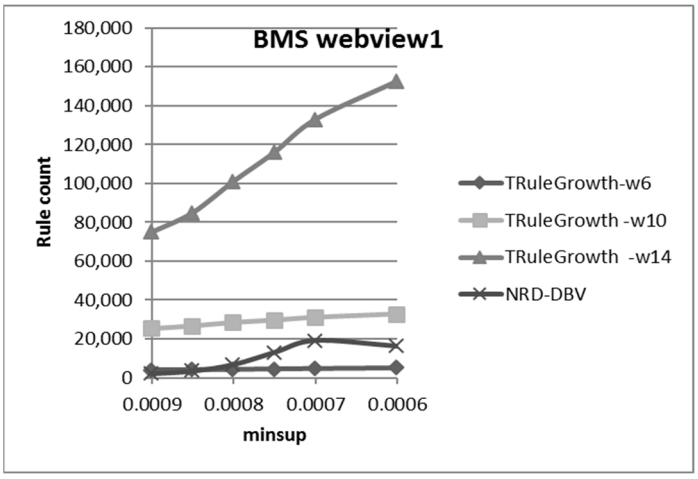

Figure 5. Comparison between the No. of rules generated for the TRuleGrowth and the NRD-DBV for (BMS webview1) dataset.

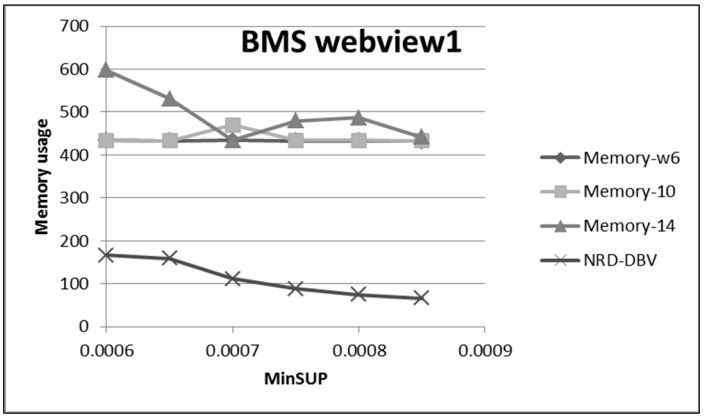

Figure 6. Comparison of memory usage for the TRuleGrowth with the NRD-DBV algorithm for (BMS webview1) dataset.

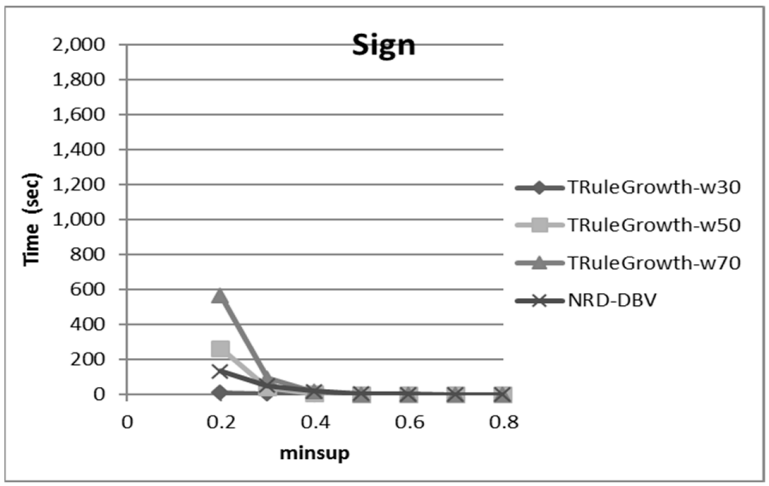

Figure 7. Comparison of the runtime of the TRuleGrowth with the NRD-DBV for (Sign) dataset.

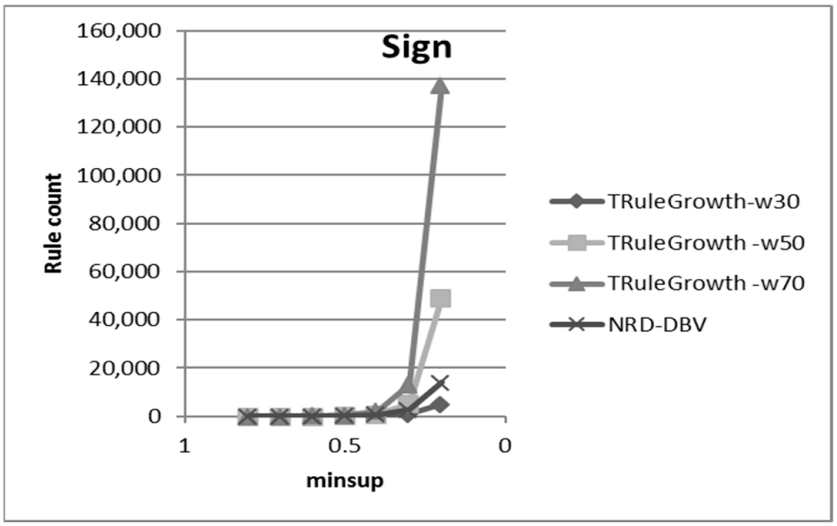

Figure 8. Comparison between the No. of rules generated for the TRuleGrowth and the NRD-DBV for (Sign) dataset.

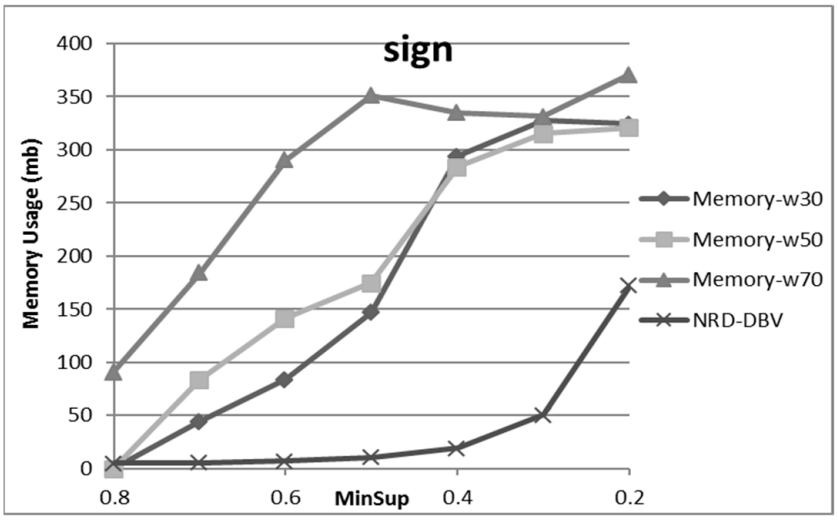

Figure 9. Comparison of memory usage for the TRuleGrowth with the NRD-DBV algorithm for (Sign) dataset.

TRuleGrowth algorithm with lower window size constraint still achieved the least time required for generating the sequential rules. This is because of lower computations needed for the smallest number of rules. NRD-DBV takes more time than TRuleGrowth as it generates more sequential rules based on ordering.

While in the Korsarak database, the NRD-DVB proved its efficiency in producing the sequential rules. It generated approximately 21 rules in 200 (s), whereas the TRuleGrowth stopped generating any sequential rules, as it was restrained by the overhead. We managed this issue by employing a subset of the Korsarak with 25,000 groupings.

By applying the alternative solution to solve the problem with stops generating rules in the TRuleGrowth algorithm, the subset of the Korsarak database with only 25,000 sequences is used. The results demonstrated the availability of the TRuleGrowth algorithm, especially when assigning a low value to the minSup threshold. It generates rules faster than the NRD-DBV algorithm. The high speed of the TRuleGrowth algorithm is evident in generating rules when diminishing the minSup value, whereas the NRD-DBV takes three times more time than the TRuleGrowth. The runtime and the number of the generated rules are not affected by different values of window size, as shown in Figures 10 and 11. The NRD-DBV achieved the best utilization of memory as shown in Figure 12.

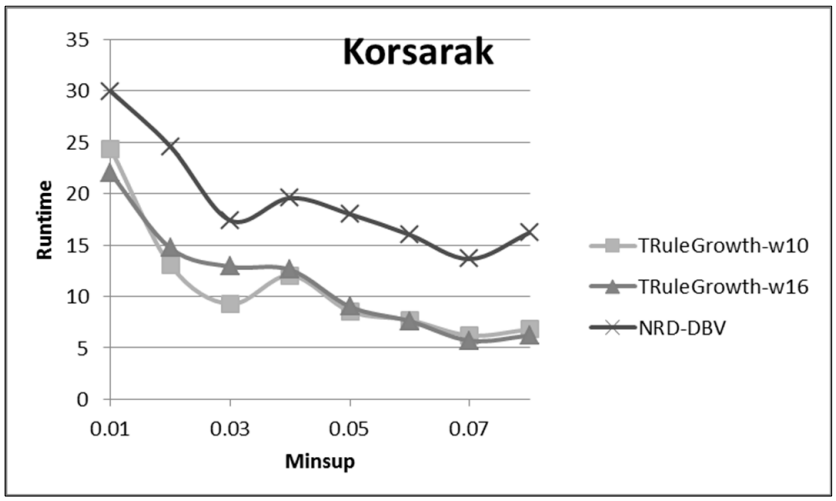

Figure 10. Comparison of the runtime of the TRuleGrowth with the NRD-DBV for (Korsarak) dataset.

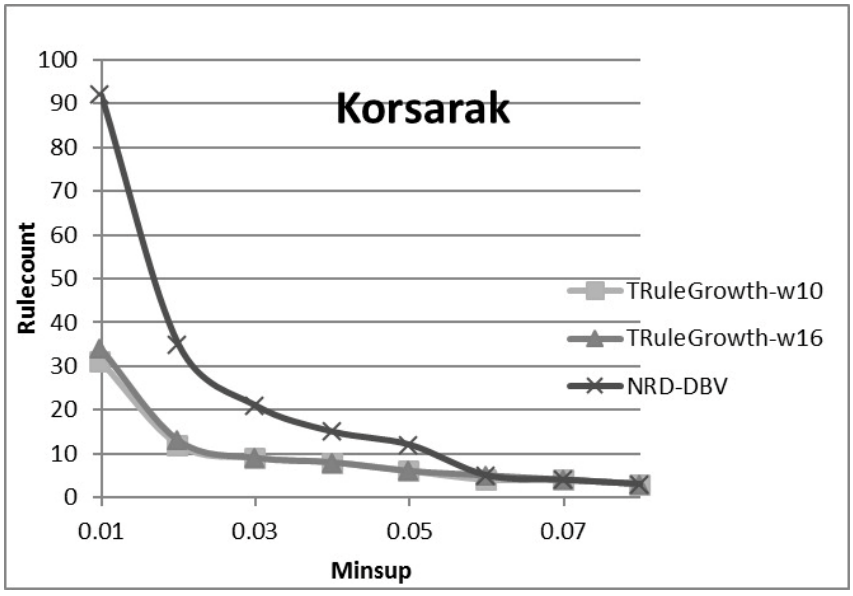

Figure 11. Comparison between the No. of rules generated for the TRuleGrowth and the NRD-DBV for (Korsarak) dataset.

The last database is named BMSwebview2; it has different features from the first database, BMSwebview1. It has 358,278 overall instances of items, whereas the BMSwebview1 has only 149,639 that forced us to assign fewer values to minSup as mentioned before. The higher the values assigned to the window size parameter, the more time taken to generate the rules, particularly when assigning a very low value to the minSup at the same time. However, the TRuleGrowth still achieved the most parcel of the execution time to produce the sequential rules, as clarified in Figure 13.

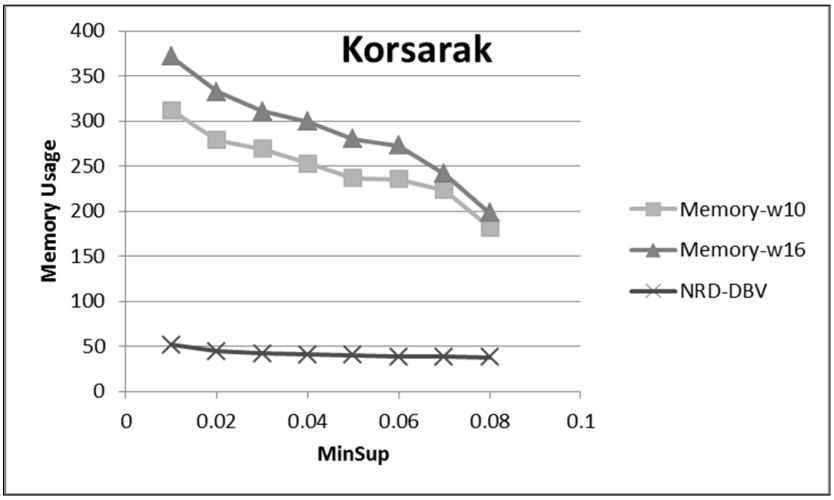

Figure 12. Comparison of memory usage for the TRuleGrowth with the NRD-DBV algorithm for (Korsarak) dataset.

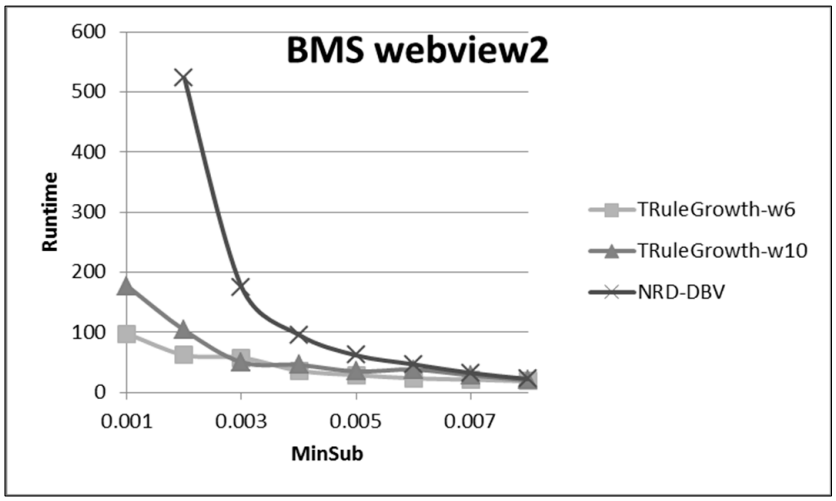

Figure 13. Comparison of the runtime of the TRuleGrowth with the NRD-DBV for (BMS webview2) dataset.

Concerning the number of rules, the TRuleGrowth algorithm generated the lowest number of rules with a low window size constraint value. With a larger window size or a higher minSup value, the number of consecutive rules in TRuleGrowth grew in the NRD-DBV algorithm because computations take longer time on a large number of rules, as shown in Figure 14, and the NRD-DBV algorithm still achieved the least memory usage as shown in Figure 15.

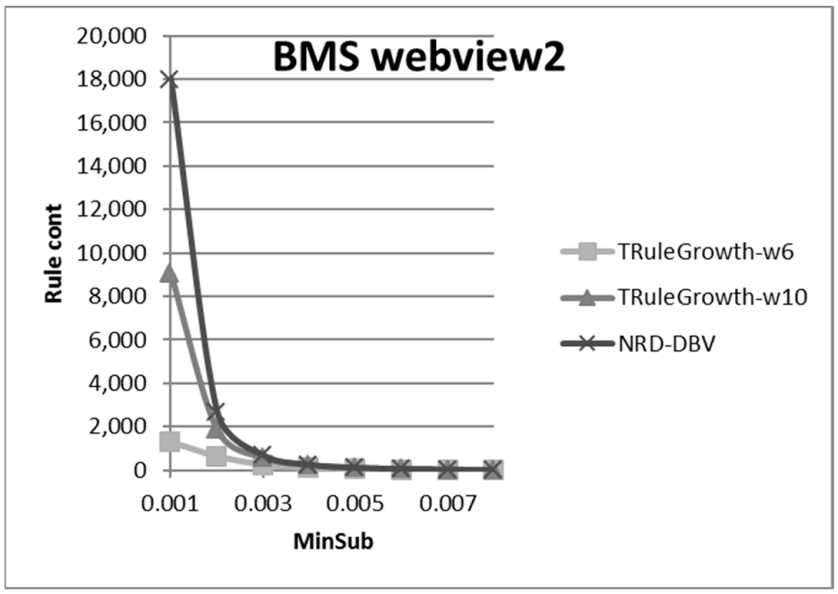

Figure 14. Comparison between the No. of rules generated for the TRuleGrowth and the NRD-DBV for (BMS webview2) dataset.

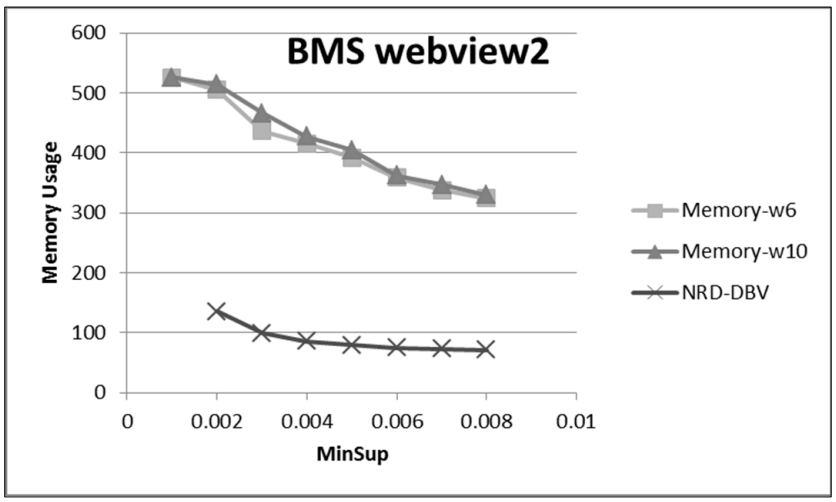

Figure 15. Memory usage for the BMS webview2 dataset with different minSup values and (minConf = 0.5).

Likewise, when using the TRuleGrowth algorithm with a small value of window size constraint or high values of minSup, the smallest number of rules was generated. When the minSup value was reduced or values of window size constraint were increased, the number of rules in TRuleGrowth increased and it took longer to make a computation.

4.2. Comparing the TNS Algorithm with the Other Two Algorithms

We compared the TNS algorithm with the other two algorithms by setting the K parameter equal to the number of rules generated from each of the other algorithms, either with different window size constraints or different minSup values.

The TNS algorithm stopped generating any rules with (BMSwebview1). That is because of the nature of this database's features: its items are rarely repeated.

In the sign database, the TRuleGrowth algorithm generated sequential rules in less time than the TNS algorithm, as shown in Figure 16. The TNS achieved its efficiency in generating sequential rules faster than the NRD-DBV algorithm (see Figure 17), and NRD-DBV achieved good performance in memory usage at the maximum value of minSup as presented in Figure 18.

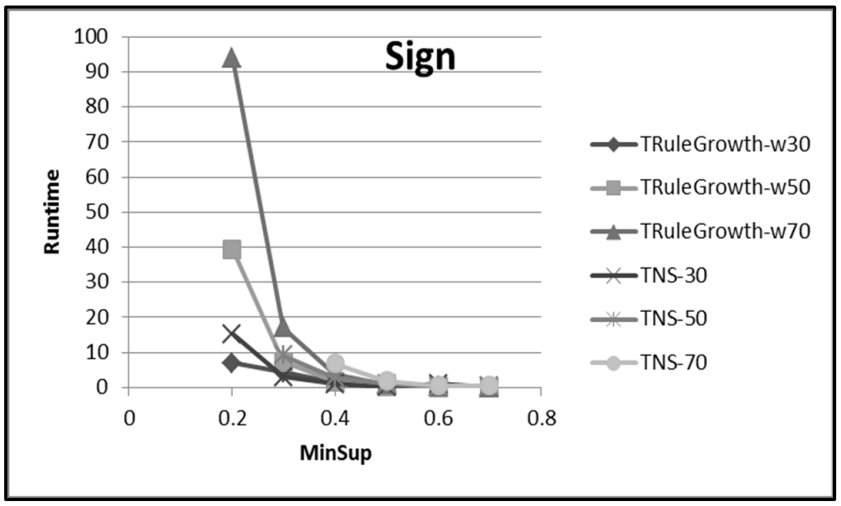

Figure 16. Comparison of the runtime of the TNS with the TRuleGrowth for (Sign) dataset.

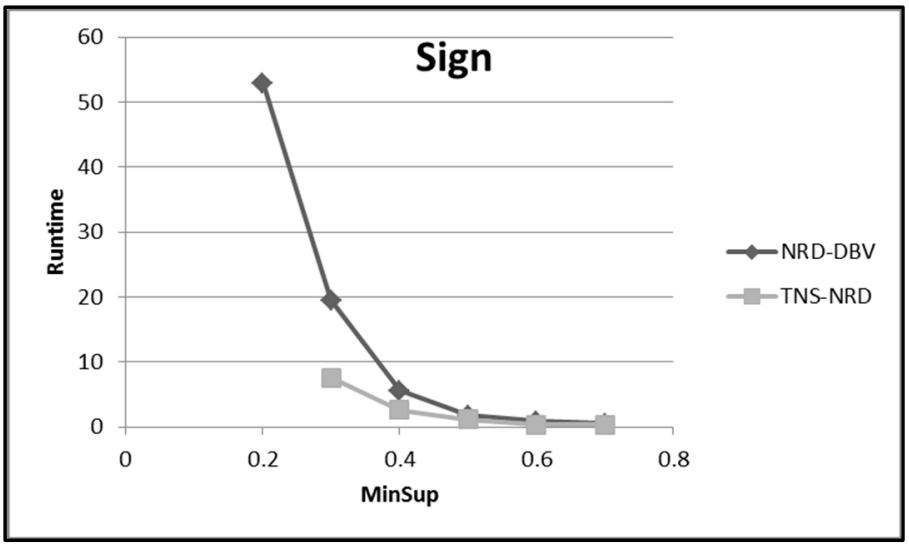

Figure 17. Comparison of the runtime of the TNS with the NRD-DBV for (Sign) dataset.

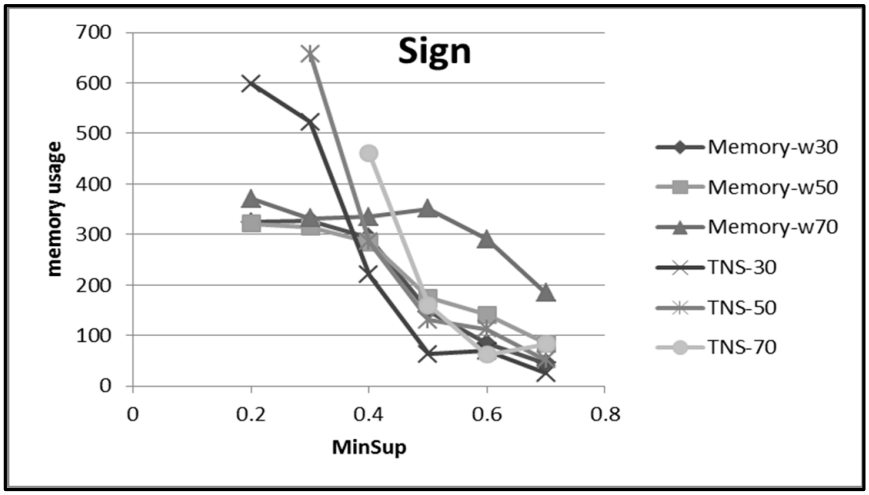

Figure 18. Comparison of memory usage for the TNS with the TRuleGrowth algorithm for (Sign) dataset.

As seen in Figures 19 and 20 in the Korsarak dataset, the TNS algorithm generated almost four times fewer sequential rules than the two other algorithms.

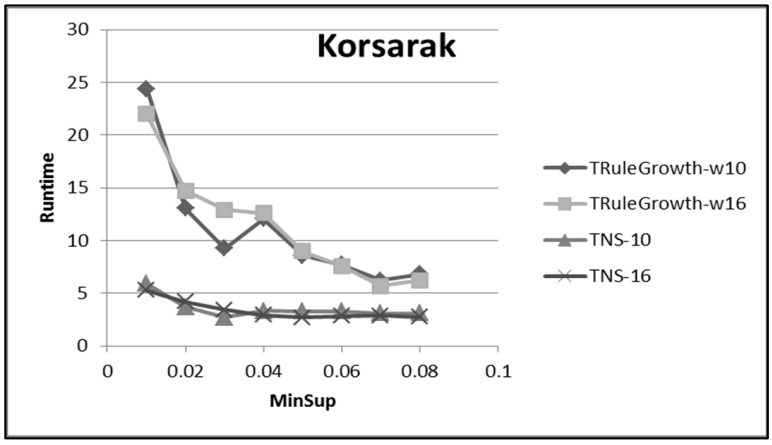

Figure 19. Comparison of the runtime of the TNS with the TRuleGrowth for (Korsarak) dataset.

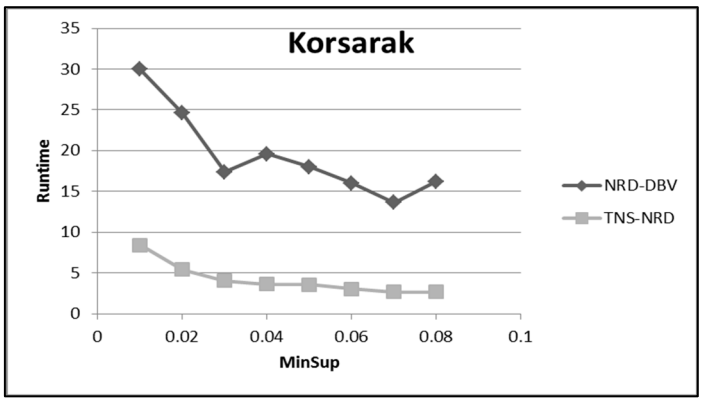

Figure 20. Comparison of the runtime of the TNS with the NRD-DBV for (Korsarak) dataset.

In the BMSwebview2 dataset, rules were generated in half the time taken for generating rules with the NRD-DBV algorithm. Additionally, rules were generated in less or approximately close to the time taken for the TRuleGrowth algorithm, as shown in Figures 21 and 22.

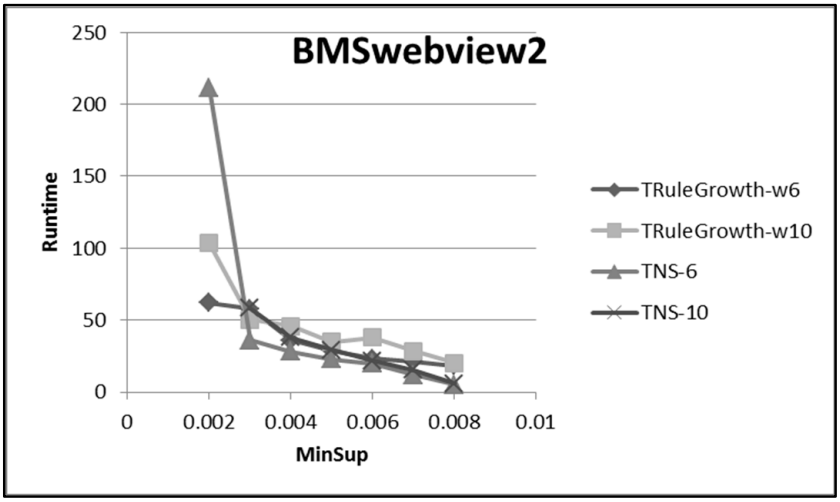

Figure 21. Comparison of the runtime of the TNS with the TRuleGrowth for (BMS webview2) dataset.

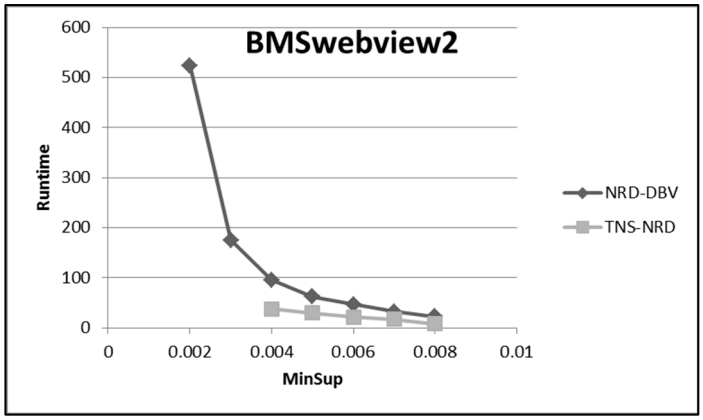

Figure 22. Comparison of the runtime of the TNS with the NRD-DBV for (BMS webview2) dataset.

Regarding the memory usage for each algorithm, it makes sense that the amount of memory demanded increased with the lowering of the minSup value, since the number of sequential rules is growing. However, in all experiments, the NRD-DBV method was demonstrated to be more memory efficient than the TRuleGrowth and TNS algorithms. This is because it has the advantage of the DBV structure and prunes all prefix child nodes to remove unnecessary rules, as shown before in Figures 6, 9, 12, 15 and 18, and also the following Figures 23–27.

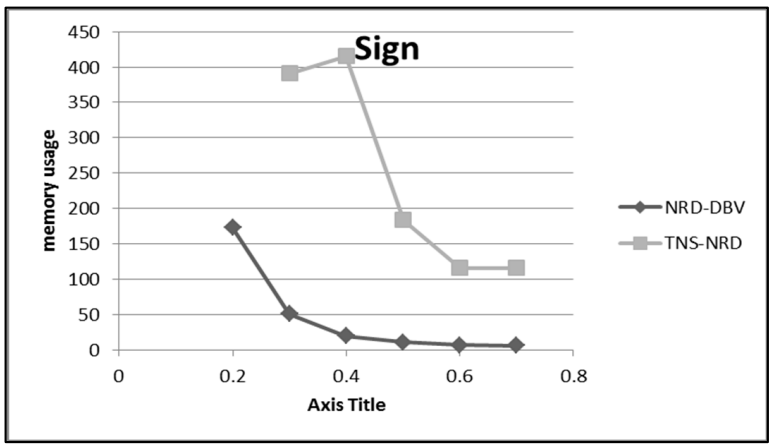

Figure 23. Comparison of memory usage for the TNS with the NRD-DBV algorithm for (Sign) dataset.

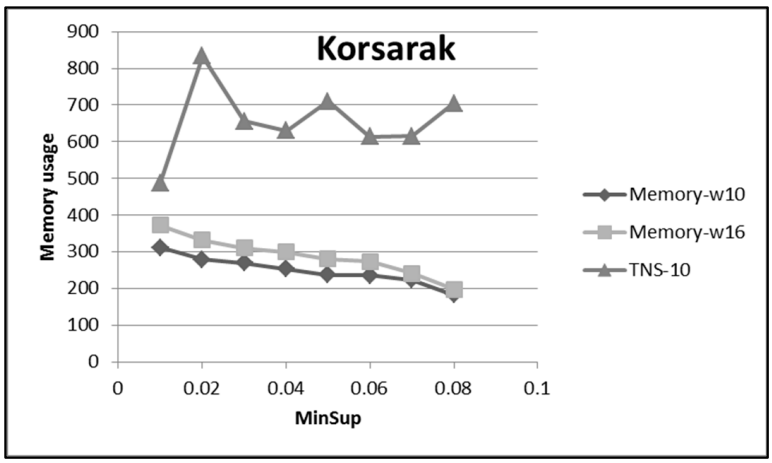

Figure 24. Comparison of memory usage for the TNS with the NRD-DBV algorithm for (Korsarak) dataset.

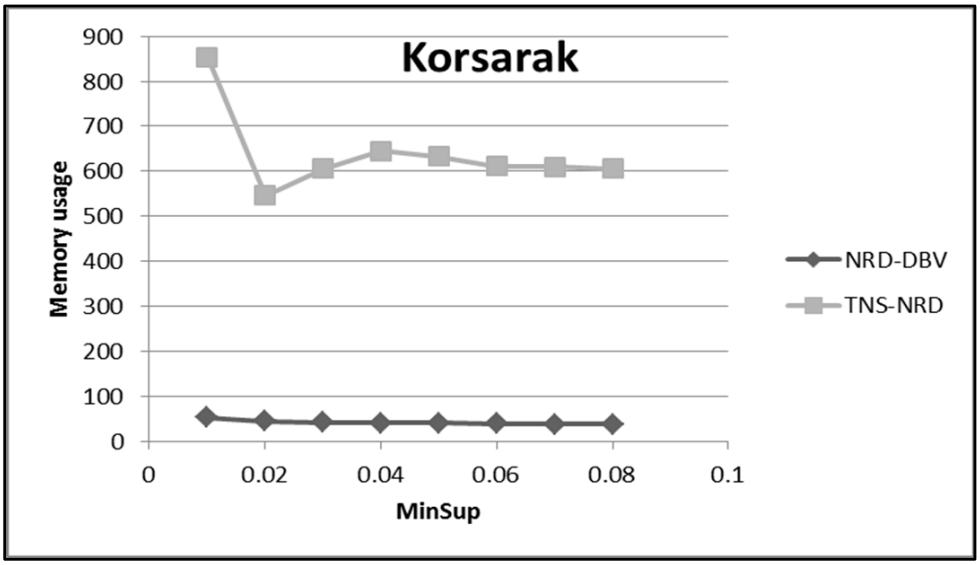

Figure 25. Comparison of memory usage for the TNS with the NRD-DBV algorithm for (Korsarak) dataset.

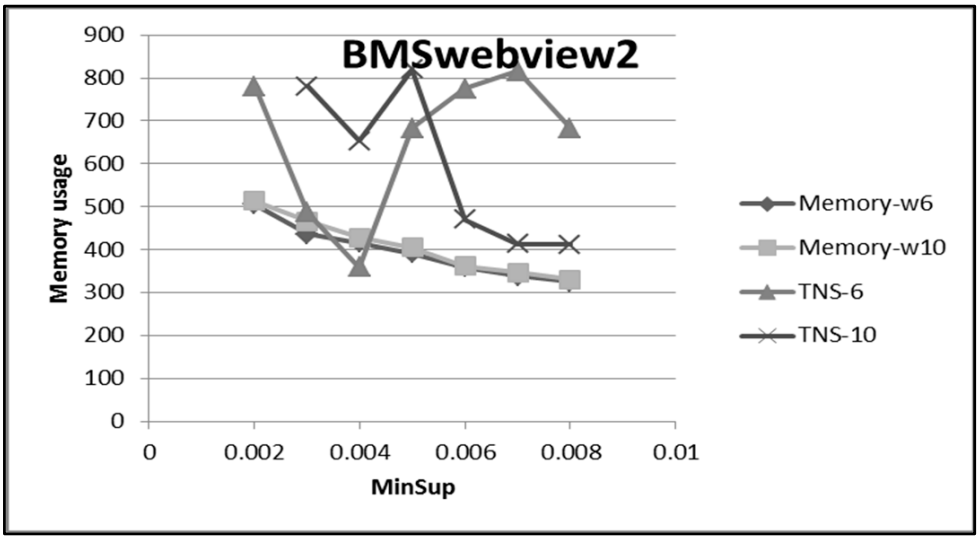

Figure 26. Comparison of memory usage for the TNS with the TRuleGrowth algorithm for (BMS webview2) dataset.

Computational complexity has been measured for each algorithm; we found that both the TRuleGrowth and NRD-DBV algorithms are less complex. The complexity of the TRuleGrowth algorithm is linear regarding the number of sequential rules in the dataset; either one or two recursive calls are applied to expand the right and left sides.

In the NRD-DBV, producing non-redundant rules has a complexity of o (n*c), where n is the number of nodes and c is the average number of child nodes. We must implement (n−1) procedures for validating and producing sequential rules such as k << n for each sequence. As a result, NRD- DBV complexity is ≈ o (n). TNS algorithm is efficient when setting the parameter k up to 2000 rules. Otherwise, it performs more complexity because of high computational expenses on mining sequential rules.

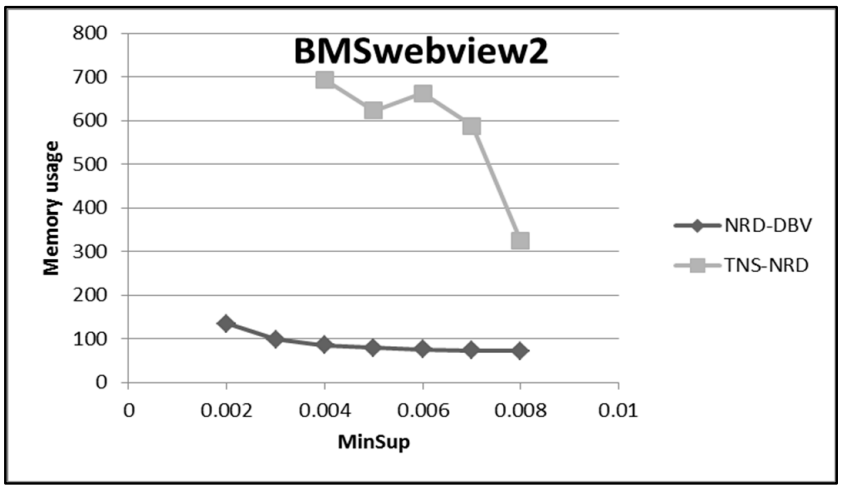

Figure 27. Comparison of memory usage for the TNS with the NRD-DBV algorithm for (BMS webview2) dataset.

5. Conclusions

This paper presents a meaningful comparison of three algorithms designed for the task of sequential rule mining, which is especially common to several sequences. Each algorithm was analyzed utilizing four real datasets with different features. For example, one dataset had long sequential patterns with low diversity of items that forced us to use a low minSup value to generate the sequential rules. Another dataset had short sequential patterns with a high diversity of items. Another one was a huge dataset that caused an overhead limit excess, which made us resort to using a subset of the original dataset.

Our experiments indicated that the performance of the algorithms is associated with the features of the datasets. Moreover, experience shows that it can reduce the execution time and control the number of valid rules generated by several orders of size restrictions such as adjusting the window size constraint or determine the number of generated rules or change the value of the minSup threshold.

With a specified value of a window size constraint, mining the TRuleGrowth algorithm can run faster and the correctness of discovered sequential rules that are not constrained by the arrangement can be increased.

The NRD-DBV algorithm could be used to reduce the number of sequential rules and memory requirements. It relies on a DBV structure with a prefix-tree that leads to early pruning of child nodes to reduce the search space. In addition, the NRD-DBV method generates more rules for taking item arrangement into account than the TRuleGrowth algorithm.

The researchers conclude that each algorithm has its own use in the fields of application of sequential rule mining to achieve the highest possible efficiency. NRD-DBV algorithm has many applications in error detection, intervention, and bugs. It is useful in domains that require the arrangement of items such as in medical area (for example, if the patient is suffering from a fever, which is followed by a decrease in the level of coagulation, followed by the appearance of red marks on the body, it is reasonable that the patient will need to be treated for dengue fever. This order in events is important in predicting an appropriate type of treatment). Additionally, it can be used in marketing to design the most personalized strategy, and also in software engineering where ordering is mostly important to accomplish its tasks.

The TRuleGrowth algorithm implementation allows the allocation of optional parameters like maximizing the number of items that appear in the antecedent and consequent of a rule. It can be useful in making product recommendations and performing fast decision making.

For future work, we intend to improve the NRD-DBV algorithm by using another concise representation, such as maximal or generator patterns, to improve performance in the large sequences database.

Author Contributions: Conceptualization, A.A. and N.Y.; methodology, A.A.; software, N.Y.; validation, A.A. and N.Y.; formal analysis, N.Y.; investigation, A.A.; resources, A.A.; data curation, N.Y.; writing—original draft preparation, N.Y.; writing—review and editing, A.A.; visualization, N.Y.; supervision, A.A.; project administration, A.A.; funding acquisition, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported through the Annual Funding track by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [Project No. AN000519].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The dataset is available on https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php, accessed on 20 April 2021.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Algorithm A1: The TRuleGrowth algorithm

Input
Sequence database
Minimum support
Minimum confidence
Output
Set of sequential rules
The executed time and memory usage of the generated rules
Algorithm
Begin
scanning dataset
For (each item m in DS) do {
Store each side that contains item m
If sup of each pair of item x,y $\geq$ minSup {
Set sides (x$\rightarrow$y) equal to zero}}
Calculate sides for generating rule (x$\rightarrow$y)
else If (sup of rule (x$\rightarrow$y) $\geq$ minSup) then
If (size of left side of rule $\leq$ maxleft && size of
right side of rule $\leq$ maxright) then
else If (rule <= window size Constraint)
Then Expand left & Expand right
End if
If confidence of rule (x$\rightarrow$y) $\geq$ minConf
Then generate rule {x}{y} with its confidence Support
End if
End for
End

Algorithm A2: TNS algorithm

Input
Sequence database
K= specified numbers of rules
Minimum support
Minimum confidence
Output
Set of sequential rules
The executed time and memory usage of the generated rules
Algorithm
Begin
R = ø. L= ø. minSup = 0.
Scan DB once & Record each item in variable(s)
FOR each pair of items I, j such that ∣sids(i)∣ ≥ minsup and ∩ 4. ∣sids(j)∣ ≥ minsup: sids(i⇒j) := Ø. sids(j⇒i) := Ø.
FOR each sid s ∈ (sids(i) ∩ sids(j):
IF i occurs before j in s THEN sids(i⇒j) := sids(i⇒j)∪7. {s}.
IF j occurs before i in s THEN sids(j⇒i) := sids(j⇒ i) ∪ 9.{*s*}.
END FOR
IF ∣*sids(i⇒j)*∣ / ∣S∣ ≥ *minsup* THEN *conf*({i}⇒{j}) := ∣*sids(i⇒j)*∣ / ∣ *sids*(*i*).
IF *conf*({i}⇒{j}) ≥ *minconf* THEN SAVE({i}⇒{j}, L, *k, minsup*).
Set flag expandLR **of** {i}⇒{j}to true.
R := R∪{{i}⇒{j}}.
END IF
. . . [lines 11 to 17 are repeated here with *i* and *j* swapped] . . .
END FOR
WHILE ∃r ∈ R AND *sup*(r) ≥ *minsup* DO
Select the rule *rule* having the highest support in R
IF *rule*.expandLR = true THEN
EXPAND-L(*rule*, L, R, *k*, *minsup, minconf*).
EXPAND-R(*rule*, L, R, *k*, *minsup, minconf*).
ELSE EXPAND-R(*rule*, L, R, *k*, *minsup, minconf*).
REMOVE *rule* from R. REMOVE from R all rules r ∈ R ∣ *sup*(r) <*minsup*.
END WHILE
SAVE(*r*, R, *k, minsup*)
L := L∪{*r*}.
IF ∣L∣ ≥ k THEN
IF *sup*(*r*) > *minsup* THEN
WHILE ∣L∣ > k AND ∃s ∈ L ∣ *sup*(s) = *minsup*
REMOVE s from L.
END IF
Set *minsup* to the lowest support of rules in L.
END IF
FOR (NRD = 1, NRD < Δ, NRD + +) Do
The result is exact (generated TNS)
ELSE Return with higher Δ value
END FOR
End

Algorithm A3: NRD-DBV algorithm

Input
Sequence database
Minimum support
Minimum confidence
Output
Set of sequential rules
The executed time and memory usage of the generated rules
Algorithm
Begin
Initialize a root→ null
NRD-seqRule→∅
Find out frequent closed sequence (FCS) by converting
pattern into DBV pattern |x| in DS &
sup(x) ≥ minSup
Set FCS as child node of a root
For (each child node cn) do
Call Closed Pattern-Extension (cn, minSup);
For (each child node cn) do
Call Generate-NRD-SeqRule (cn, minConf,
NRD-SeqRule)
End

Algorithm A4: Closed Pattern-Extension method

Input
Frequent sequential patterns
Minimum support
Minimum confidence
Output
Set of sequential rules
The executed time and memory usage of the generated rules
Algorithm
Begin
Set listNode →child nodes of root
For (each prefix Sequence in listNode) do
If sequential patterns not pruned, then
for (each prefix sequential patterns in listNode) do
If (sup (PrefixSp →Sequence-extension ≥ minSup) then
Add prefixSP as a new itemset after the last itemset of the sequence
Else If (sup (Sp →Itemset-extension ≥ minSup)
Add prefixSP as a new item in the last itemset of sequence
End For
Call Closed Pattern-Extension (prefixSP, minSup);
End If
Check & put the attribute of SP: closed pattern, prefixed generator or NULL;
End For
End

References

1. Mooney, C.H.; Roddick, J.F. Sequential pattern mining—Approaches and algorithms. *ACM Comput. Surv.* **2013**, *45*, 1–39. [CrossRef]
2. Hemeida, A.; Alkhalaf, S.; Mady, A.; Mahmoud, E.; Hussein, M.; Eldin, A.M.B. Implementation of nature-inspired optimization algorithms in some data mining tasks. *Ain Shams Eng. J.* **2020**, *11*, 309–318. [CrossRef]
3. Huynh, B.; Bay, V.O.; Vaclav, S. An efficient parallel method for mining frequent closed sequential patterns. *IEEE Access* **2017**, *5*, 17392–17402. [CrossRef]
4. Kour, A. Sequential Rule Mining, Methods, and Techniques: A Review. *Int. J. Comput. Intell. Res.* **2017**, *13*, 1709–1715.

5. Toussaint, B.-M.; Luengo, V. Mining surgery phase-related sequential rules from vertebroplasty simulations traces. In Proceedings of the Conference on Artificial Intelligence in Medicine in Europe, Pavia, Italy, 17–20 June 2015; pp. 35–46.
6. Werke, M. Principles for Modelling of Manufacturing Sequences. Ph.D. Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2015.
7. Noughabi, E.A.Z.; Albadvi, A.; Far, B.H. How Can We Explore Patterns of Customer Segments' Structural Changes? A Sequential Rule Mining Approach. In Proceedings of the 2015 IEEE International Conference on Information Reuse and Integration, San Francisco, CA, USA, 13–15 August 2015; pp. 273–280.
8. Jannach, D.; Jugovac, M.; Lerche, L. Adaptive recommendation-based modeling support for data analysis workflows. In Proceedings of the 20th International Conference on Intelligent User Interfaces, Atlanta, GA, USA, 29 March–1 April 2015; pp. 252–262.
9. Leemans, M.; van der Aalst, W.M. Discovery of frequent episodes in event logs. In *International Symposium on Data-Driven Process Discovery and Analysis*; Springer: Cham, Switzerland, 2014; pp. 1–31.
10. Bhoomika, A.P.; Selvarani, R. A Survey on Web Page Recommender Systems. In Proceedings of the Alliance International Conference on Artificial Intelligence and Machine Learning (AICAAM), Bangalore, India, 26–27 April 2019.
11. Jamshed, A.; Mallick, B.; Kumar, P. Deep learning-based sequential pattern mining for progressive database. *Soft Comput.* **2020**, *24*, 17233–17246. [CrossRef]
12. Bajaj, S.B.; Garg, D. Survey on Sequence Mining Algorithms. *Int. J. Eng. Appl. Sci. Technol.* **2016**, *1*, 58–64.
13. Alja'am, J.M.; El Saddik, A.; Sadka, A.H. (Eds.) *Recent Trends in Computer Applications: Best Studies from the 2017 International Conference on Computer and Applications, Dubai, UAE*; Springer: Berlin/Heidelberg, Germany, 2018.
14. Kiran, R.U.; Kitsuregawa, M.; Reddy, P.K. Efficient discovery of periodic-frequent patterns in very large databases. *J. Syst. Softw.* **2016**, *112*, 110–121. [CrossRef]
15. Setiawan, F.; Yahya, B.N. Improved behavior model based on sequential rule mining. *Appl. Soft Comput.* **2018**, *68*, 944–960. [CrossRef]
16. Senthilkumar, R.; Deepika, R.; Saranya, R.; Govind, M.D. Generating adaptive partially ordered sequential rules. In Proceedings of the International Conference on Informatics and Analytics, Pondicherry, India, 25–26 August 2016; ACM: New York, NY, USA, 2016; pp. 1–8.
17. Pham, T.T.; Luo, J.; Hong, T.P.; Vo, B. An efficient method for mining non-redundant sequential rules using attributed prefix-trees. *Eng. Appl. Artif. Intell.* **2014**, *32*, 88–99. [CrossRef]
18. Tran, M.T.; Le, B.; Vo, B.; Hong, T.P. Mining non-redundant sequential rules with dynamic bit vectors and pruning techniques. *Appl. Intell.* **2016**, *45*, 333–342. [CrossRef]
19. Van, T.T.; Vo, B.; Le, B. IMSR_PreTree: An improved algorithm for mining sequential rules based on the prefix-tree. *Vietnam. J. Comput. Sci.* **2014**, *1*, 97–105. [CrossRef]
20. Fournier-Viger, P.; Wu, C.W.; Tseng, V.S.; Cao, L.; Nkambou, R. Mining partially-ordered sequential rules common to multiple sequences. *IEEE Trans. Knowl. Data Eng.* **2015**, *27*, 2203–2216. [CrossRef]
21. Pujari, M.S.D.; Mane, M.R.; Ghorpade, V.R. Analysis of TRuleGrowth algorithm for discovery of sequential rules. *Int. J. Eng. Res. Technol.* **2017**, *10*, 396–399.
22. Indhumathi, V.; Karthika, S.K. An Efficient Way to Handle the High Dimensional Problem with Fuzzy Association Rule. *Int. J. Eng. Res. Technol.* **2016**, *4*, 1–6.
23. Saritha, P.C.; Senthil Prakash, T.; Rajesh, M.; Remya, K.S. Discovering Sequential Rules for Web Usage Analysis. *Int. J. Eng. Technol. Sci.* **2015**, *II*, 57–62.
24. Fournier-Viger, P.; Tseng, V.S. Mining top-k sequential rules. In Proceedings of the International Conference on Advanced Data Mining and Applications, Beijing, China, 17–19 December 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 180–194.
25. Mollenhauer, D.; Atzmueller, M. Sequential Exceptional Pattern Discovery Using Pattern-Growth: An Extensible Framework for Interpretable Machine Learning on Sequential Data Minign. In Proceedings of the XI-ML@ KI, Bamberg, Germany, 21 September 2020.
26. Bou Rjeily, C.; Badr, G.; Al Hassani, A.H.; Andres, E. Overview on Sequential Mining Algorithms and Their Extensions. In *Recent Trends in Computer Applications*; Springer: Cham, Switzerland, 2018; pp. 3–16.
27. Fournier-Viger, P.; Tseng, V.S. TNS: Mining top-k non-redundant sequential rules. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal, 18–22 March 2013; pp. 164–166.
28. Jamsheela, O.; Raju, G.K. Parallelization of Frequent Itemset Mining Methods with FP-tree: An Experiment with PrePost + Algorithm. *Int. Arab J. Inf. Technol.* **2021**, *18*, 208–213.
29. Nguyen, H.Q.; Pham, T.T.; Vo, V.; Vo, B.; Quan, T.T. The predictive modeling for learning student results based on sequential rules. *Int. J. Innov. Comput. Inf. Control* **2018**, *14*, 2129–2140.
30. Wang, C.-S. Mining Non-Redundant Inter-Transaction Rules. *J. Inf. Sci. Eng.* **2015**, *31*, 1849–1865.
31. Youssef, N.; Abdulkader, H.; Abdelwahab, A. Evaluating Non-redundant Rules of Various Sequential Rule Mining Algorithms. In Proceedings of the International Conference on Advanced Intelligent Systems and Informatics, Cairo, Egypt, 19–21 October 2020; Springer: Cham, Switzerland, 2020.

32. Wu, Y.; Zhu, C.; Li, Y.; Guo, L.; Wu, X. NetNCSP: Nonoverlapping closed sequential pattern mining. *Knowl.-Based Syst.* **2020**, *196*, 105812. [CrossRef] [PubMed]
33. Fournier-Viger, P.; Lin, J.C.W.; Gomariz, A.; Gueniche, T.; Soltani, A.; Deng, Z.; Lam, H.T. The SPMF open-source data mining library version 2. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Riva del Garda, Italy, 20–22 September 2016; pp. 36–40.