# Improving the Efficiency of Interactive Sequential Pattern Mining by Closed Pattern Discovery

Yui Aoyagi[1], Hieu Hanh Le[1][0000−0003−3702−8974], Ryosuke Matsuo[1], Tomoyoshi Yamazaki[1], Kenji Araki[1], Haruo Yokota[2][0000−0001−9788−0443], and Masato Oguchi[1]

[1] Ochanomizu University, Tokyo, Japan
yui@ogl.is.ocha.ac.jp,le@is.ocha.ac.jp,matsuo@ldi.or.jp,
{yamazaki,araki6925}@gmail.com,oguchi@is.ocha.ac.jp
[2] Josai University, Tokyo, Japan
yokota.h.aa@gmail.com

**Abstract.** With the increasing adoption of big data, sequential pattern mining (SPM) —a technique for identifying frequent patterns within sequential data— has attracted significant attention. As the optimal support threshold depends on the dataset, interactive SPM, which allows users to dynamically adjust algorithm parameters, is essential for practical analysis. However, conventional interactive SPM methods rely on known frequent patterns and often overlook closed frequent patterns. This limitation may hinder analytical efficiency, especially when comprehensive yet concise pattern sets are required. In domains, such as medical records, closed frequent patterns alone can offer a comprehensive understanding of treatment processes. In this study, we propose a method to accelerate interactive SPM by focusing on closed frequent patterns. Our approach enhances pattern analysis by efficiently reusing previously mined closed patterns. Furthermore, we demonstrate the effectiveness of incorporating closed pattern consideration through evaluations on public datasets. The results demonstrate improved efficiency in pattern discovery compared to conventional approaches.

**Keywords:** Sequential pattern mining · Closed frequent patterns · Frequent pattern mining.

## 1 Introduction

With the accumulation of various types of big data, sequential pattern mining (SPM)[2], which focuses on sequences, has been attracting attention. SPM is a data analysis method that extracts frequent sequential patterns (FSP) by specifying a minimum support threshold (minsup). It can yield more valuable insights than frequent itemset extraction methods[1] in databases where the order of item occurrences is crucial. In SPM, minsup is expressed as a percentage of the total sequence count in the dataset. The frequency with which a sequence appears in a dataset is the support value, and a sequence is considered frequent if the

support value is greater than the minsup. As the optimal minsup depends on the dataset, a significantly small minsup will lead to an increase in FSP whereas an excessively large minsup may result in no patterns being generated. Therefore, an interactive SPM that is skilled at performing the analysis while adjusting the minsup is essential. The conventional method, KISP[3], stores discovered frequent patterns in a knowledge base (KB) and retrieves them as needed. However, FSP often contain significant redundancy owing to numerous patterns generated.

This study aims to accelerate the mining process by limiting candidate sequences to only closed FSP. To extract only closed patterns, we generate candidate sequences exclusively from closed sequences. Accordingly, we propose a modified KB structure suitable for managing closed patterns. We evaluate the proposed method using real datasets, measuring runtime and analyzing the effectiveness of closed pattern consideration, KB utilization, and the management of positional information of candidate sequences.

The remainder of this paper is organized as follows. Section 2 describes the related works in this study. Section 3 presents our proposed method of interactive SPM considering closed patterns. Section 4 reports on our experiments using two public datasets and discusses the results. Finally, Section 5 concludes the paper and outlines directions for future work.

## 2 Related Work

This section describes related topics to our study, including SPM, PrefixSpan, frequent closed sequential patterns, and KISP.

### 2.1 Sequential Pattern Mining and PrefixSpan

Sequential pattern mining (SPM), proposed by Agrawal et al., is a method for extracting all FSP from a sequence database (SDB). In SPM, a pattern is considered frequent if its frequency exceeds the specified minsup.

PrefixSpan [5] proposed by Jian Pei et al. is an SPM method that obtains FSP by depth-first search. Given an SDB and a minsup, the algorithm finds all frequent sequential patterns whose support is at least minsup. The key idea is to recursively construct projected databases based on prefixes, avoiding the need to generate all candidate combinations.

### 2.2 Frequent Closed Sequential Patterns

Many frequent sequential patterns (FSP) are redundant. To reduce this, frequent closed sequential patterns (FCSP) [6] are extracted. An FCSP has no supersequence with the same support, so shorter redundant FSP are removed.

### 2.3 KISP

KISP is an interactive SPM method using a knowledge base (KB), suitable for repeated executions with different minsup values. The KB stores previously discovered patterns, their support values, and the smallest minsup encountered (KB.base). If a new minsup is larger than KB.base, frequent patterns can be retrieved directly from the KB, improving performance.

## 3 Proposed Method

Figure 1 shows the structure of the proposed method. The proposed interactive SPM extends the mechanism of KISP, which utilizes previously mined frequent patterns stored in KB. There are two main proposals.

### 3.1 Candidate Sequence Generation

The first is to generate only closed sequences as candidate sequences. Thus, only FCSP can be extracted. Moreover, it is expected that the mining process will become faster owing to the reduced number of candidate sequences.

Figure 2 shows the procedure for generating candidate sequences. First, given a candidate sequence and its position information, verification is performed to ensure that the minsup and closed conditions are satisfied. If they are satisfied, the sequence is considered to be a frequent sequence and stored in the KB. Subsequently, the position information of the new candidate sequence is obtained. If the position information is already stored in the KB, it is retrieved from there. Otherwise, it is calculated and stored in the KB. The method for calculating position information follows the same approach as PrefixSpan, where a projected database is constructed and then used to compute the positions. Finally, candidate sequences are pruned, with those failing to satisfy the minsup and closed conditions are excluded. By repeating this process, all frequent patterns can be obtained.

### 3.2 Structure of KB

The second contribution is related to the structure of the KB. In KISP, the KB stores the smallest minsup used, denoted as KB.base, along with FSP and their

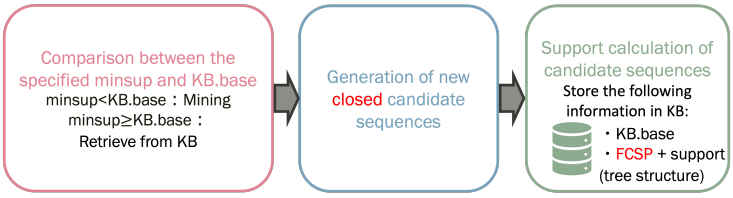


**Fig. 1.** Proposed Method

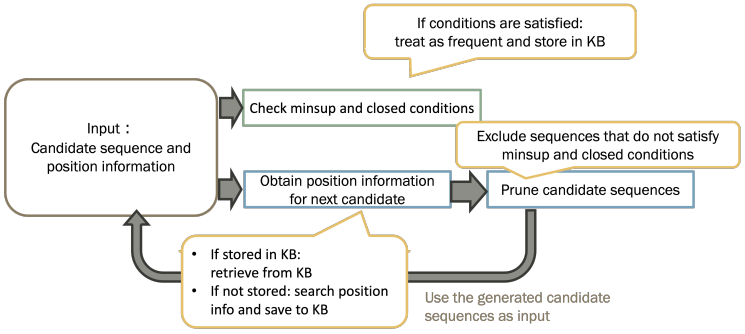


**Fig. 2.** Candidate sequence generation

support values. FSP are grouped by their sequence length to facilitate the generation of candidate sequences. In the proposed method, the KB stores KB.base, FCSPs, and their support values. As this method considers only closed sequential patterns when determining frequency, it is sufficient to store these three types of information. In addition, the proposed method modifies the candidate generation strategy compared to KISP, eliminating the need to group patterns by size. Instead, patterns are managed using a tree structure. In addition to the above, the position information of sequences examined during candidate sequence generation is stored in a hash structure. A hash table is implemented using Python's dict type, and a custom dictionary structure is designed to manage hash collisions with linked lists. To retain the original keys, a list is used, allowing a single key to be associated with multiple values. By storing the position information of sequences that are not considered frequent, the method allows the reuse of support values in future mining steps, eliminating the needing for recomputation.

## 4 Experiments

The primary objective of this experiment is to evaluate the effectiveness of the proposed method in utilizing KB, closed considerations, and maintaining position information.

**Table 1.** Experimental Environment

| | |
|---|---|
| Server | Dell PowerEdge R740xd |
| CPU | Intel Xeon Gold 5218 16 cores x 2 |
| OS | Ubuntu 24.04.1 LTS |
| Memory | 64GB x 6 |
| python | 3.12.3 |

**Table 2.** Overview of the Datasets

| | **BMSWebView1** | **BMSWebView2** |
|---|---|---|
| Number of sequences | 59,601 | 77,512 |
| Average number of items per sequence | 2.42 | 4.62 |
| Size (MB) | 1.5 | 3.6 |
| Description | Clickstream data from an EC site | |

### 4.1 Experimental Environment

Table 1 shows the experimental environment. The detailed contents of the dataset[4] used are shown in Table 2.

### 4.2 Experimental Details

We implemented the proposed method and conducted three types of experiments.

First, we compared the execution time of the proposed method with PrefixSpan to examine the effect of using KB on the execution time of the proposed method. Using BMSWebView1 dataset, we measured the execution time as the minsup value by gradually increasing from 0.01 to 0.07. Additionally, we measured the execution time while gradually decreasing minsup from 0.07 back to 0.01.

Subsequently, to investigate the significance of generating only closed candidate sequences, we measured and compared the execution times with and without considering closed sequences. We used BMSWebView1 and BMSWebView2 datasets, and executed the mining process with minsup incremented by 0.0001 from 0.00065 to 0.00070. Furthermore, we investigated the change in the number of candidate sequences.

Finally, we evaluated the effectiveness of storing the position information of candidate sequences. For this experiment, we employed the BMSWebView1 dataset and expressed minsup as an absolute value, not as a percentage. We ran the mining process with minsup from 60 to 55 in steps of 1, comparing the execution time with and without storing the position information. The details are shown in Figure 3. In the proposed method, when obtaining the position information of a new candidate sequence, if the position information is stored in the KB, it is retrieved from the KB; otherwise, it is newly computed. We refer to this approach as Pattern 1. However, repeatedly recomputing the position information is referred to as Pattern 2. We compared the execution times of these two patterns. In addition, we measured the memory size of the position information in the KB generated during Pattern 1.

### 4.3 Experimental Results

**Verification of Execution Time Using the KB** The average execution time is shown in Figure 4. When using a step size of 0.0005, PrefixSpan was faster.

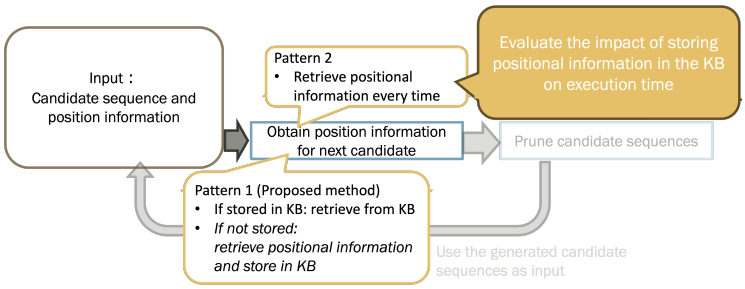


**Fig. 3.** Comparison of positional information retrieval methods

| Increment | PrefixSpan(s) | Proposed Method(s) | Proposed Method /PrefixSpan |
|---|---|---|---|
| 0.0005 | 14.86 | 15.16 | 102% |
| 0.0004 | 18.43 | 15.29 | 83% |
| 0.0003 | 24.75 | 16.02 | 65% |

**Fig. 4.** Execution Time with KB Usage (by Step Size of minsup)

However, with a step size of 0.0004, the proposed method outperformed PrefixSpan. Specifically, the execution time of proposed method was approximately 83% of PrefixSpan's as a step size of 0.0004, and approximately 65% at a step size of 0.0003. Therefore, the proposed method is more effective as the number of executions increases. PrefixSpan performs mining again when the minsup changes. However, although the proposed method performs similar mining when minsup is initially 0.01, it only retrieves sequences that satisfy the conditions from the KB; thus, it is assumed to be faster.

Furthermore, we gradually decreased minsup from 0.07 to 0.01. Each was measured three times, and the average execution times are shown in Figure 5. As with increasing minsup, the results demonstrate that the proposed method becomes faster as the number of executions increases. However, the advantage of the proposed method only became evident when the step size was reduced to 0.0001 and the number of executions increased to 600. This is presumably because, after the second execution, the proposed method does not simply extract frequent sequences from the KB, but instead generates new candidate sequences and calculates their support values.

**Verification of Execution Time with Closed Consideration** The results are shown in Figure 6. In both datasets, the execution times were faster when closed sequences were considered. This improvement is attributed to the reduced number of candidate sequences owing to the closed-sequence filtering. Figure 7 shows a comparison of the number of generated candidate sequences. Considering the previous results as well, it is evident that in both datasets, the method incorporating closed sequence consideration was faster, as it reduced the number

| Decrease | PrefixSpan(s) | Proposed Method(s) | Proposed Method /PrefixSpan |
|---|---|---|---|
| 0.0005 | 14.67 | 27.35 | 186% |
| 0.0004 | 18.44 | 29.71 | 161% |
| 0.0003 | 24.75 | 34.41 | 139% |
| 0.0002 | 36.74 | 40.91 | 111% |
| 0.0001 | 73.98 | 62.67 | 85% |

**Fig. 5.** Execution Time with KB Usage (by Step Size of minsup Decrease)

| | With Closed Check(s) | Without Closed Check(s) | With / Without Closed Check |
|---|---|---|---|
| BMSWebView1 | 392.39 | 816.75 | 48% |
| BMSWebView2 | 323.33 | 325.61 | 99% |

**Fig. 6.** Closed Check Execution Time

of candidate sequences. Furthermore, the difference in execution time depending on whether closed sequences are considered varies by dataset, which indicates a correlation with the amount of reduction in candidate sequences. Therefore, it has been confirmed that considering closed sequences causes a reduction in candidate sequences; thus, contributing to faster execution.

**Retention of Candidate Sequence Position Information** We define the proposed method that stores position information as Pattern 1, and the method that searches for position information each time as Pattern 2. The execution times of these two approaches are shown in Table 3. A comparison shows that the method without the position information storage performs faster. To investigate this further, the memory size of the position information stored in the KB during Pattern 1 was measured. The results are shown in the last line of Table 3. It is presumed that owing to the large size of the data structure used to manage position information, the operations of inserting and searching within this structure require considerable time.

## 5 Conclusion

### 5.1 Summary

SPM requires specifying an appropriate minsup. Therefore, using interactive SPM facilitates in identifying a suitable minsup more effectively. In this study, we confirmed that levering previously mined frequent patterns and incorporating closed pattern constraints can reduce the execution time of interactive SPM. The results show that the use of KB is faster than PrefixSpan for both increasing and decreasing minsup. Moreover, because a smaller number of candidate sequences

| | With Closed Check | Without Closed Check | With / Without Closed Check |
|---|---|---|---|
| BMSWebView1 | 85,186,590 | 195,240,402 | 44% |
| BMSWebView2 | 53,330,915 | 67,693,468 | 79% |

**Fig. 7.** Candidate Sequence Count Comparison

**Table 3.** Runtime and Memory for Position Info Storage

| minsup | 60 | 59 | 58 | 57 | 56 | 55 |
|---|---|---|---|---|---|---|
| Pattern 1 (s) | 43.42 | 487.45 | 539.31 | 600.99 | 674.84 | 753.35 |
| Pattern 2 (s) | 7.41 | 7.76 | 8.14 | 8.53 | 9.00 | 9.56 |
| Memory Size (GB) | 1.29 | 1.36 | 1.43 | 1.51 | 1.61 | 1.71 |

causes faster processing, extracting only FCSPs reduces the number of patterns and contributes to the overall speedup of the algorithm.

### 5.2 Future Work

Future work will focus on evaluating the structure for managing FSP in the KB. As storing position information with the current structure increases execution time, alternative data structures for storing position information will be explored. Furthermore, we plan to validate the effectiveness of the proposed method by comparing it with KISP.

**Acknowledgments** This research was partially supported by a grant from the Japan Society for the Promotion of Science (#24K02943).

## References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94). pp. 487–499 (1994)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering. pp. 3–14 (1995)
3. Lin, M.Y., Lee, S.Y.: Improving the efficiency of interactive sequential pattern mining by incremental pattern discovery. In: International Conference on System Sciences. pp. 68–76 (2002)
4. P., F.V.: An open-source data mining library. http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php (2024), accessed: 2024-12-20
5. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: PrefixSpan : Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceeding of 2001 international conference on data engineering. pp. 215–224 (2001)
6. Yan, X., Han, J., Afshar, R.: Clospan: Mining: Closed sequential patterns in large datasets. In: 2003 SIAM International Conference on Data Mining. pp. 166–177 (2003)