Efficient Methods for Mining Clickstream Patterns

Huy Minh Huynh, Ph.D.

Doctoral Thesis Summary

Tomas Bata University in Zlín

Tomas Bata University in Zlín
Faculty of Applied Informatics

Doctoral Thesis Summary

Clickstream Efektivní metody pro dolování clickstream vzorů

Efficient Methods for Mining Clickstream Patterns

Author: **Huy Minh Huynh, Ph.D.**

Degree programme: P0613D140027 Information Technologies

Supervisor: Prof. Ing. Zuzana Komínková Oplatková, Ph.D.

Consultant: Assoc. Prof. Bay Vo, Ph.D.

External examiners: Prof. RNDr. PaedDr. Eva Volná, Ph.D.
Doc. Ing. Radek Šilhavý, Ph.D.
Prof. Ing. Pavel Krömer, Ph.D.

Zlín, August 2025

Published by **Tomas Bata University in Zlín** in the Edition **Doctoral Thesis Summary.**
The publication was issued in the year 2025

Key words in Czech: *clickstream pattern mining, sekvenční dolování vzorů, dolování vzorů, vážené dolování vzorů, paralelní výpočty, inkrementální algoritmy, dolování dat*

Key words: *clickstream pattern mining, sequential pattern mining, pattern mining, weighted pattern mining, parallel computing, incremental algorithms, data mining*

Full text of the doctoral thesis is available in the Library of TBU in Zlín.

ISBN 978-80-7678-357-7

ABSTRACT

Mining and extracting informative data have been of great interest to researchers in recent years. One form of extracted information is frequent patterns, which express the common behaviors that often happen throughout historical records. A distinct type of such pattern, clickstreams, has emerged due to the Internet and online commercial business. A simple example is a sequence of web pages that users often visit, and knowing such behaviors can help website owners customize their web pages to help improve the user experience or recommend helpful information. The task of discovering clickstreams can be called clickstream pattern mining. This work explores some aspects of clickstream pattern mining as well as the possibilities for improving the performance of algorithms in this topic. Through the experiments in all 5 cases of study, the thesis' proposed approaches were compared with other state-of-the-art methods. They were shown not only effective and efficient but also faster than the other methods on most test databases.

Keywords: *clickstream pattern mining, sequential pattern mining, pattern mining, weighted pattern mining, parallel computing, incremental algorithms, data mining*

ANOTACE

Vytěžování a získávání informačních dat jsou v posledních letech předmětem velkého zájmu. Jednou z forem extrahovaných informací jsou často se opakující vzory, které vyjadřují obvyklé chování, k němuž často dochází v historických záznamech. Výrazný typ takových vzorů, tok kliků (clickstream), se objevil díky internetu a online komerčnímu podnikání. Jednoduchým příkladem je posloupnost webových stránek, které uživatelé často navštěvují, a znalost takového chování může majitelům webových stránek pomoci přizpůsobit jejich webové stránky tak, aby pomohli zlepšit uživatelský zážitek nebo doporučit užitečné informace. Takovou úlohu zjišťování clickstreamů lze nazvat clickstream pattern mining (dolování vzorů clickstreamů). Tato práce zkoumá některé aspekty dolování vzorů clickstreamů a také možnosti zlepšení výkonnosti algoritmů v tomto tématu. V práci byly provedeny experimenty v 5 případových studiích a v práci navržené přístupy byly porovnány s jinými moderními metodami. Na většině testovacích databází se ukázalo, že jsou nejen účinné a efektivní, ale také rychlejší než ostatní metody.

Keywords: *clickstream pattern mining, sekvenční dolování vzorů, dolování vzorů, vážené dolování vzorů, paralelní výpočty, inkrementální algoritmy, dolování dat*

TABLE OF CONTENTS

1. INTRODUCTION

Frequent patterns, or common behaviors, exist everywhere in our daily lives, explicitly or implicitly. Somehow, people generate and use these patterns more than they know. Consider a simple situation: someone is preparing dinner but lacks a necessary ingredient. They know their sister often passes by a supermarket on her way home from work around this time. Without hesitation, they call and ask her to stop by the supermarket and purchase the missing item. In this case, the sister's usual route becomes a frequent pattern, demonstrating how knowing people's typical behaviors can be helpful.

Of course, this usefulness did not easily fly under the radar of people working in business and commercial. Frequent patterns generated by customers, such as products often purchased together, are valuable assets as they can help boost profits and sales. For instance, when a customer is about to buy cigarettes, a store assistant might suggest some bottles of beer. Because these items are often purchased together, the customer is likely to accept the suggestion, resulting in increased sales.

As computers and the Internet appeared, information has mostly shifted from storing on paper to digital storage. People, as well, have accustomed themselves to browsing the Internet and online shopping. This shift in habits produces an abundance of digital data. Nonetheless, manually finding out useful frequent patterns from such large raw data is not an easy job. This gave an opportunity for pattern mining, whose task is to dig out interesting and useful frequent patterns, to emerge and advance. To the best of the author's knowledge, it was not until 1993 that Agrawal et al. formally defined the problem of frequent pattern mining [1]. Since then, more variants of frequent patterns have attracted people's interest.

As the Internet and e-commerce bloomed, website owners tried to understand the users' browsing behaviors to improve their websites or recommend useful information. Frequent clickstream patterns emerged as one method to achieve these objectives. Nevertheless, frequent clickstream pattern mining was mostly viewed through the lens of another problem: sequential pattern mining. Thus, frequent clickstream pattern mining has not yet been reasonably studied. Additionally, performance may not be optimal when sequential pattern algorithms are employed to mine clickstream patterns, as they represent a distinct subset of sequential patterns. To address those issues, the author's thesis will be an attempt to explore and study some aspects of clickstream pattern mining.

The remaining content of this thesis summary is structured as follows (the full names of the proposed algorithms are in Table 1.1):

- The second chapter briefly reviews both clickstream pattern mining (CPM) and related fields such as frequent itemset mining (FIM) and sequential pattern mining (SPM).
- The third chapter introduces the first contribution of this thesis: the novel CUP algorithm designed to solve the frequent CPM problem.
- In the fourth chapter, the CUP algorithm is extended to handle the SPM problem, and the novel SUI algorithm is introduced.
- The fifth chapter explores weighted CPM as well as parallelism and introduces the novel Compact-SPADE, DPCompact-SPADE, and APCompact-SPADE.
- The sixth chapter addresses the problem of incremental CPM, where new data is continuously added to a database. Three novel algorithms, PF-CUP, PSB-CUP, and PSB-CUP+, are introduced to tackle this issue.
- The final chapter summarizes the main contributions of the thesis, highlighting the successful development of multiple novel algorithms for CPM and SPM.

Table 1.1. List of author's proposed algorithms and their abbreviations.

Term	Meaning
CUP	Clickstream pattern mining Using Pseudo-IDList
SUI	Sequential pattern mining Using IDList
Compact-SPADE	Compact Sequential Pattern Discovery using Equivalence classes
DPCompact-SPADE	Depth-first Parallel Compact-SPADE
APCompact-SPADE	Adaptive Parallel Compact-SPADE
PF-CUP	Pre-Frequent Clickstream mining Using Pseudo-IDList
PSB-CUP	Progressive Search Border Clickstream mining Using Pseudo-IDList
PSB-CUP+	Enhanced version of PSB-CUP

1.1 Thesis Aims and Scope

This thesis revolves around adopting, improving, and proposing algorithms for mining clickstream patterns. Clickstream pattern mining can have many derived problems; however, this thesis focuses specifically on the following five aims.

1. Propose approaches that can exploit certain clickstreams' characteristics for mining frequent clickstream patterns.
2. Extend the proposed algorithm to mine sequential patterns in which each itemset can include multiple items/events.
3. Integrate weight factor to give another option for ranking patterns other than the general support measure (i.e., the frequency of which the patterns appear).
4. Improve performance by integrating parallelism into existing clickstream pattern mining algorithms.
5. Propose methods to tackle the situation where new clickstream data is being added on a daily basis.

The main method to verify those goals is through the evaluation of experiments. The factor for judging algorithms is performance regarding runtime and maximum memory consumption, as they are commonly used measurements. Some additional metrics, such as speedup or scalability, will also be employed. Some additional experimental results will be analyzed to provide more insights into how the proposed algorithms are effective and efficient.

This thesis only abides by the problems defined within this thesis. The scope does not go beyond exploring the clickstream problems defined by other work. Furthermore, this thesis does not judge the patterns' quality but mainly focuses on the performance of algorithms.

1.2 Datasets for Experiments

Each chapter of the thesis summary uses different groups of test sets. In summary, the characteristics of all the test databases are as follows with details given in the corresponding section.

Table 1.2. Summary of all test databases.

Database	Database size	Average sequence length	Distinct items
D900S20N3	900,000	17.3	2,279
FIFA	20,450	36.24	2,990
Kosarak	990,002	8.1	41,270

MSNBC	989,818	4.75	17
BMS2	77,512	4.62	3,340
C50S15T3	49,060	39.97	2,611
C150S40T2	150,000	76.64	954
C200S12T5	183,950	51.57	1,922
BIBLE	36,369	13,905	21.6
SIGN	730	310	51.99
Chainstore	1,112,949	46,086	7.2
D9000S4	9,000,000	5,000	3.72

2. RELATED WORK

This section provides a brief overview of advancements in pattern mining, with a particular emphasis on frequent itemset mining (FIM), sequential pattern mining (SPM), weighted pattern mining, and incremental pattern mining.

Frequent Itemset Mining (FIM). Frequent itemset mining is one of the earliest forms of pattern mining [1] and serves as the foundation for many other advanced problems. The goal is to identify patterns of frequently co-occurring items in datasets, such as products frequently bought together in retail. Early methods focused on efficiently pruning the search space using properties like anti-monotonicity. The thesis builds upon these foundational techniques to propose algorithms tailored to clickstream patterns.

Sequential Pattern Mining (SPM). Sequential pattern mining extends FIM by considering the order in which events occur [2]. This approach is geared towards analyzing customer behaviors over time or detecting trends. Classic well-known SPM methods include PrefixSpan [3], SPADE [4], SPAM [5], and recently CM-SPAM and CM-SPADE [6]. Those algorithms can be categorized based on the taxonomy of their database representations. For example, if an algorithm utilizes a mainly horizontal database (e.g., Fig. 3.1 on Page 11), it is considered a horizontal algorithm, whereas vertical algorithms use vertical databases (e.g., Fig. 3.3 on Page 13). PrefixSpan can be put into the horizontal group, while the other four algorithms can be considered in the vertical group.

Weighted Pattern Mining. Over time, researchers realized that treating all items with equal importance was not practical for many applications. Weighted pattern mining started to develop [7, 8] to address this by assigning varying levels of significance (weights) to different items. This concept complicated the mining process due to the loss of anti-monotonicity, a property that enables efficient pruning of infrequent patterns. Yun and Leggett [7] were the first to introduce weight constraints in SPM, but this violated anti-monotonicity, so they proposed weight ranges to preserve it. In FIM, similar ideas were explored with the

introduction of structures like WMFP-tree [9], which incorporated weights while ensuring anti-monotonicity was preserved. Researchers also extended these methods to variants like erasable itemsets [10], though adding weights to alternative patterns such as closed patterns [11] proved more difficult due to potential information loss.

Incremental Pattern Mining. In real-world applications, data grows continuously (e.g., retail transactions), making rerunning mining algorithms inefficient and resource-heavy. Non-incremental approaches overlook previously mined patterns, leading researchers to explore incremental mining, which updates existing patterns as new data is added, rather than starting from scratch. Early work, such as the FUP algorithm [12], reused previous mining results to reduce candidate patterns but still required full rescans. Hong et al. introduced the FUFP-tree [13] and later the pre-large concept [14] to minimize these rescans by setting thresholds. The pre-large concept was also adapted for sequential pattern mining (SPM) in algorithms like the PreFUSP-Tree-Ins algorithm [13].

Clickstream Pattern Mining (CPM). The clickstream pattern mining problem is a more specific variation of sequential pattern mining. CPM focuses on analyzing user interactions, often on websites, to identify navigation patterns. It has been historically treated as a subset of sequential pattern mining, with limited algorithms developed specifically for CPM. For example, Tarus et al. [15] built a recommendation system for e-learning that suggested suitable learning resources. The system incorporates an SPM algorithm to extract clickstream patterns from web logs to set up ranked candidates' lists.

3. FREQUENT CLICKSTREAM PATTERN MINING

Frequent clickstream pattern mining is a fundamental problem, laying the foundation and providing motivation for newer challenges. This problem involves the identification of useful patterns via the support values (i.e., the frequencies or the numbers of times the pattern appears in a given data). However, issues can arise when existing algorithms are used to mine clickstream patterns. This chapter aims to tackle some issues of the existing algorithms.

The content of this chapter is overlapped with the author's previous research presented in a journal article [A.7], and its preliminary version in a conference paper [A.13]. The content has been refined to align with the overall themes of this thesis on clickstream pattern mining.

3.1 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of integer values, and each value represents an event (e.g., a click on a website's URL), **a clickstream** $S = \langle x_1, x_2, \ldots, x_m \rangle$ is a sequence

of events, where $x_i \in I$. An event can appear more than once in the same sequence at various positions. The event's location denotes the order in which the event happens.

The number of events in a clickstream denotes the clickstream **length**, and the clickstream with a length k is called a k-clickstream. For example, <2, 3, 2> is a 3-clickstream because it has three events.

Let $X = \langle x_1, x_2, \ldots, x_{k-1}, x_k \rangle$ and $Y = \langle y_1, y_2, \ldots, y_{j-1}, y_j \rangle$ be two clickstream with $k < j$. $X \subseteq Y$ indicates that Y is a **super-clickstream** of X or X is a **sub-clickstream** of Y if there exist integers $1 \leq t_1 < t_2 < \ldots < t_j \leq j$ such that $x_1 = y_{t1}, x_2 = y_{t2}, \ldots, x_j = y_{tj}$. For example, <2, 3, 2> and <5, 3, 6> are sub-clickstreams of <2, 5, 6, 6, 3, 6, 2>.

A clickstream $Z = \langle z_1, z_2, \ldots, z_{k-1} \rangle$ is called a (k-1)-**prefix** of X if $x_1 = z_1, x_2 = z_2, \ldots, x_{k-1} = z_{k-1}$. Consequently, Z is a sub-clickstream of X, and $X = \langle Z, x_k \rangle$. For example, <1, 4, 3> is a prefix of <1, 4, 3, 2>. Additionally, X is considered to belong to the **equivalence class** [Z] if Z is X's prefix. For example, <1, 4, 3, 2> belong to the equivalence class [<1, 4, 3>]. In other words, the class [Z] is a set of patterns that share the same prefix Z.

A **user (generated) clickstream** is a clickstream generated by a user via various actions such as browsing the Internet or navigating folders on a computer.

UCID	User clickstream
100	<2, 3, 1, 1, 2, 3, 4, 2, 3, 2, 1>
200	<2, 5, 6, 6, 3, 6, 2>
300	<2, 1, 4, 1, 2>
400	<1, 5, 6, 3, 2, 3, 2>
500	<4, 1, 1, 2, 4>

Fig. 3.1. An example of a clickstream database.

A **clickstream database** (*CDB)* is a collection of user clickstreams, each of which is assigned a ***UCID*** (user clickstream id). For example, Fig. 3.1 is a *CDB* with five user clickstreams.

A **(clickstream) pattern** is a sub-clickstream of one or more user clickstreams. In other words, the pattern **appears** in one or more user clickstreams, or one or more user clickstreams **contain** the pattern. For example, <2, 3, 4> is a pattern in Fig. 3.1, but <2, 3, 6> is not.

A **pattern candidate** is a clickstream formed from two patterns. The candidate that may or may not appear in the database being examined. For example, given two patterns <2, 5, 6> and <2, 5, 2>. Possible candidates are <2, 5, 6, 2> and <2, 6, 2, 5>. The former exists in Fig. 3.1 but not the latter.

The **support (count)** of a pattern X is denoted by $supp(X)$. It is the number of user clickstreams that are the pattern's super clickstreams, i.e., $supp(X) = |\{Y \mid \forall Y \in CDB: X \subseteq Y\}|$. For example, $supp(<2, 3, 2>) = 3$ because <2, 3, 2> appears 3 times in user clickstreams 100, 200 and 400.

Given a **minimum support threshold** δ, a pattern X is **frequent** if its support is equal to or larger than δ, i.e., $supp(X) \geq \delta$. For example, given $\delta = 3$. Pattern <2, 3, 2> is frequent because $supp(<2, 3, 2>) \geq \delta$.

The task of frequent clickstream pattern mining (CPM) is to all frequent clickstream patterns with support $\geq \delta$ in a given clickstream database *CDB*.

3.2 Challenges and Goals

As stated above, this thesis progresses with vertical format algorithms because they were reported to have more advantages and performance. In particular, the SPADE algorithm family was mainly used to propose the author's new algorithms.

The vertical format algorithms mostly focus on mining sequential patterns. Those patterns are similar to clickstream patterns, except that each event in a sequential pattern can contain more than one item. For example, given clickstream pattern $C = <1, 2, 3>$ and sequential pattern $S = <(1, 2), 2, (1, 3)>$, the first event in C is "1" while it is "(1, 3)" in S.

Applying SPM algorithms to CPM directly can be unoptimized. This happens because existing vertical algorithms tend to copy and store duplicate data. On large databases, duplicate information occurs more noticeably, causing the memory requirements to increase.

Pattern: <2>		
Data id	UCID	Position list
1	100	1, 5, 8, 10
2	200	1, 7
3	300	1, 5
4	400	5, 7
5	500	4

Pattern: <1, 2>		
Data id	UCID	Position list
1	100	5, 8, 10
3	300	5
4	400	5, 7
5	500	4

Fig. 3.2. An example of duplicate data.

The duplicate data is usually a part of the Data-IDList (section 3.3). In Fig. 3.2, the example shows that there are two patterns <2> and <1, 2> and their respective Data-IDLists. All data of pattern <1, 2>'s Data-IDList is contained within pattern <1>'s Data-IDList. For example, the first row with UCID = 100 is quite similar in both <1, 2> and <1>'s Data-IDLists, as the <1, 2>'s position lists are partly

duplicated from <1>'s position lists. This usually results in duplicate data being carried over and further to other patterns during the mining process.

Not to mention that the vertical group usually uses generate-candidate-and-test approaches. They can produce large numbers of pattern candidates, and each candidate has their own data stored in memory. This also contributes to the memory issue.

This chapter's goal is to propose a CPM algorithm based on the existing vertical family algorithm with the following proposed improvements by the author:

- Reducing the duplicate information to ease the memory problem and,
- Enhancing the runtime by reducing the number of generated candidates.

Those improvements are achieved through the Pseudo-IDList (Section 3.4) and the dynamic intersection upper bound constraint (Section 3.5).

3.3 Data-IDList

A Data-IDList is a data structure that records where a pattern appears in the horizontal database, specifying which user clickstreams contain the pattern and the positions where the pattern's last item appears. In this chapter, Data-IDLists are represented as matrices.

M

Pattern: <1>		
Data id	UCID	Position list
1	100	3, 4, 11
2	300	2, 4
3	400	1
4	500	2, 3

Pattern: <3>		
Data id	UCID	Position list
1	100	2, 6, 9
2	200	5
3	400	4, 6

Fig. 3.3. An example of Data-IDLists of frequent 1-patterns.

Following [16], only the positions of the pattern's last item in a user clickstream are recorded. For example, for the user clickstream 400 = <1, 5, 6, 3, 2, 3, 2> and the pattern <5, 6, 3>, the position list is <4, 6>. A set of Data-IDLists forms a vertical database (Fig. 3.3). Each Data-IDList contains:

- *P*: the pattern for which the position is stored.
- *M*: a matrix with three columns: {*Data id*, *UCID*, *Position list*}. The position list contains positions where the last item of the pattern occurs in the user clickstream. The data id is a locally assigned identifier for user clickstreams.
- The support *supp*(*P*) is the number of row elements in *M*.

3.4 Pseudo-IDList

A Pseudo-IDList holds an index matrix that indirectly represents a Data-IDList for a pattern through a pointer to a Data-IDList (DIP). It includes the following components:

- *P*: The pattern whose position information is recorded.
- *DIP*: A pointer to the Data-IDList of the frequent 1-pattern that matches the last item of P (e.g., for <1, 2, 3>, DIP points to the Data-IDList of <3>).
- *M*: An index matrix with columns {Local id, Data id, Start index}. Using the start index, the position list for the pattern P is a sub-list of the corresponding Data-IDList.
- The support *supp*(*P*) is the number of row elements in *M*.

While Data-IDLists are created only for frequent 1-patterns, Pseudo-IDLists are generated for all pattern candidates during mining. For *k*-patterns ($k \geq 2$), only Pseudo-IDLists are used. Fig. 3.4 demonstrates how a Pseudo-IDList retrieves the relevant data by indexing into a corresponding Data-IDList, such as for pattern <1, 3>.

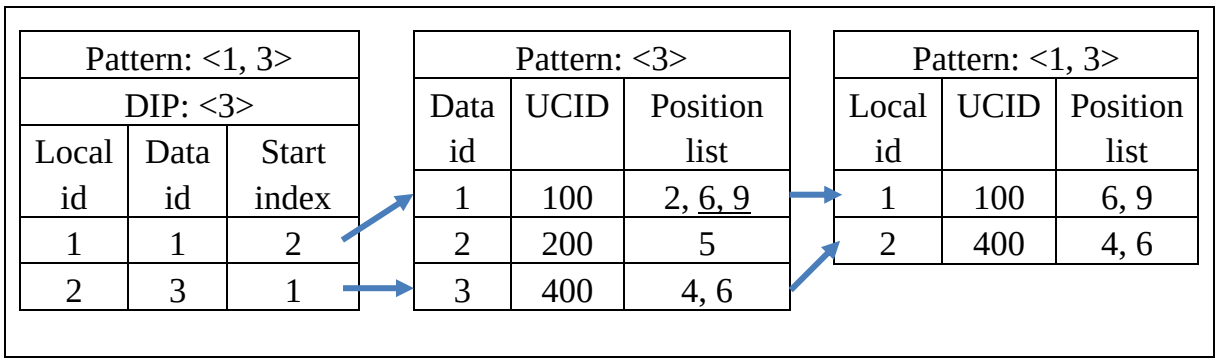

Fig. 3.4. Transformation of Pseudo-IDList of pattern <1, 3> (left-most) to Data-IDList of pattern <1, 3> (right-most).

3.5 Pruning with Dynamic Intersection Upper Bound Constraint

Let *X* and *Y* be two frequent clickstream patterns that share the same prefix, and $P_1 = \langle X, last_Y \rangle$ and $P_2 = \langle Y, last_X \rangle$ be two candidates generated from *X* and *Y*. Let S_X and S_Y be sets of user clickstreams that contain *X* and *Y*, respectively and let $S_\cap = S_X \cap S_Y$.

3.5.1 Dynamic Intersection Upper Bound Constraint (DUB)

Theorem 3.1. The number of sequences in the intersection set $S_{\cap}$, denoted as $supp(S_{\cap})$, is greater or equal to the support count of either P_1 or P_2; i.e., $supp(S_{\cap}) \geq supp(P_1)$ and $supp(S_{\cap}) \geq supp(P_2)$. The interpretation is that $supp(S_{\cap})$ denotes the max possible support value of extended patterns P_1 and P_2. This is considered a tighter upper bound than the downward closure property where $supp(X) \geq supp(S_{\cap}) \geq supp(P_1)$ and $supp(Y) \geq supp(S_{\cap}) \geq supp(P_2)$.

Based on Theorem 3.1, the author proposes the Dynamic Intersection Upper Bound (DUB) heuristic. This heuristic utilizes the established upper bound of $supcount(S_{\cap})$ to prune candidate patterns efficiently during mining.

3.5.2 Usage

Suppose the new candidates do not satisfy DUB. In that case, they are not frequent patterns and can be discarded immediately, and thus, the later steps (i.e., creating Pseudo-IDList and checking support) are unnecessary. Fig. 3.5 illustrates an example of DUB, where DUB bitmaps of <3, 3> and <3, 6> are joined to create an intersection bitmap. From this, the $supp(S_{\cap})$ for DUB is identified as one.

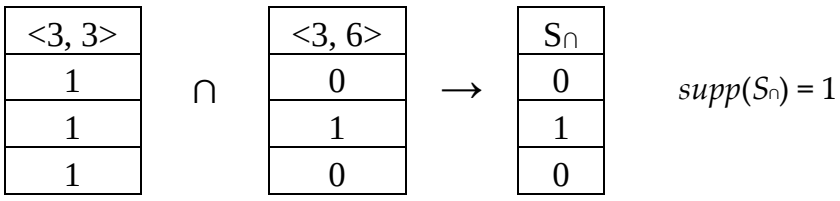

Fig. 3.5. An example of DUB.

3.6 Candidate Generation

A similar method to that of (CM-)SPADE is employed to generate candidates. Specifically, given two frequent k-patterns X and Y, that share the same (k-1)-prefix, let $last_X$ and $last_Y$ be the two events of X and Y, respectively. If $last_X$ is not the same as $last_Y$, then a set of two candidates is generated, which is {<X, $last_Y$>, <Y, $last_X$>}. Otherwise, the set only contains one candidate {<X, $last_Y$>}. This step also includes the pruning heuristic that was introduced earlier.

3.7 Algorithm: Clickstream Pattern Mining Using Pseudo-IDList (CUP)

Generally, the mining processes of CUP use the idea of traversing search space in a depth-first search manner to enumerate the entire set of frequent patterns. It contains the following main steps.

Step 1. Identifying frequent 1-patterns in a given horizontal database and transforming the horizontal database to a set of Data-IDLists (i.e., a vertical database as illustrated in Fig. 3.3). Each Data-IDList holds the information necessary for frequent 1-patterns (Section 3.3).

Step 2. Generating pattern candidates with length (k+1) from two given frequent k-patterns that share the same (k-1)-prefix. Frequent 1-patterns are considered as sharing the 0-prefix (or an empty prefix). The proposed pruning heuristic (Section 3.5) is used right after this to discard redundant candidates.

Step 3. Creating data for the generated candidates and checking support values. Any candidate that does not satisfy the minimum support threshold γ is discarded. Their support is obtained via the proposed Pseudo-IDList. Each candidate is assigned one Pseudo-IDList, and the Pseudo-IDList is created by joining data from two parents' IDLists. The process then loops back at the generating candidate step (step 2) until no candidates can be found.

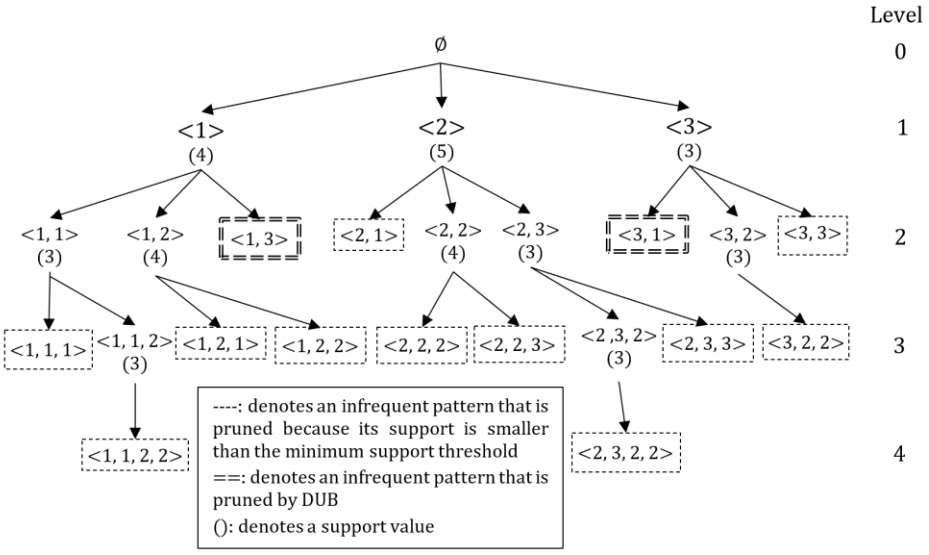

Fig. 3.6. The pattern search space corresponding to the example database.

3.8 Experiment

In this section, the setups and environments for running experiments are described. The algorithms are CUP (the author's proposed algorithm with Pseudo-IDList and Data-IDList), CUP+DUB (in which the pruning heuristic DUB is

integrated into CUP), and two existing algorithms, PrefixSpan [3], CM-SPADE [6] and SPAM [5].

3.8.1 Environment Setup

The algorithms are implemented in Java, and the tests are carried out on Windows 10 64-bit edition and JDK 8. The hardware specifications include 2.2GHz Intel I7 8750H (with the Turbo Boost function deactivated for more stable runtimes) and 16GB RAM. CM-SPADE and PrefixSpan are obtained via the SPMF package [17]. Four real-life clickstream databases are collected via the SPMF site for the performance benchmark. The experiments in this chapter use the FIFA, BMS2, Kosarak, and MSNBC databases in Table 1.2. All algorithms are executed on all test databases while the minimum support threshold decreases until one of the algorithms takes too long to finish.

3.8.2 Performance Analysis

On all test databases, CUP(+DUB) outperforms SPAM, PrefixSpan, and CM-SPADE in terms of execution time, while CM-SPADE runs faster than PrefixSpan on FIFA, Kosarak, and MSNBC. SPAM has the highest runtime among the four algorithms and cannot run on Kosarak due to exceeding the allowed memory. On large databases (Kosarak and MSNBC), CUP(+DUB) runs noticeably faster than the other two. When minimum support thresholds are high, not many frequent patterns are produced; the differences in runtime among the four algorithms are small. However, as the minimum support threshold gets smaller, the gaps in runtimes get bigger. The runtimes of PrefixSpan and CM-SPADE also increase dramatically at a faster rate. CUP(+DUB) works well because Pseudo-IDList does not have to copy redundant data; instead, this method only produces indices on existing Data-IDLists.

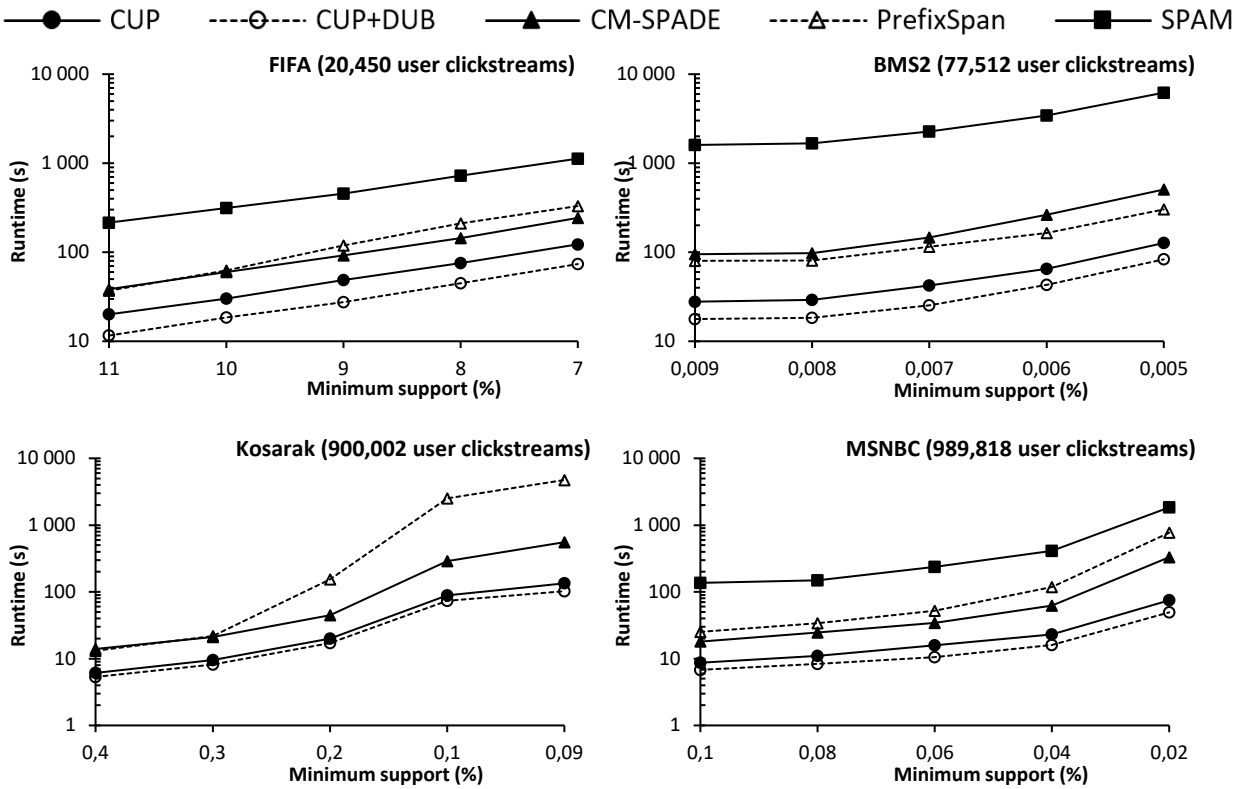

Fig. 3.7. Runtime for four databases.

4. FREQUENT SEQUENTIAL PATTERN MINING

This chapter looks into frequent sequential pattern mining, a more general problem of clickstream pattern mining. The content of this chapter is overlapped with the author's previous research presented in a journal article [A.3] and its preliminary version in a conference paper [A.12].

4.1 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of integer values. Each $i_1 \in I$ represents an item (e.g., a pair of shoes). **A sequence** $s = \langle E_1, E_2, \ldots, E_m \rangle$ $(1 \leq i \leq m)$ is a list of itemsets in an order, where **an itemset** $E_i \in s$ is a subset of I and m is the **sequence size** of s. For example, let $I = \{1, 2, 3, 4, 5\}$, two possible 3-itemsets can be $\{2, 4, 5\}$ and $\{1, 2, 5\}$.

A **user transaction sequence** is generated according to the customer's orders when a customer purchases items in a store. A sequence is enclosed with "<" and ">" symbols while "{" and "}" enclose an itemset. For example, <{2, 3},{1, 2, 3},{1, 3}> is a transaction sequence with three itemsets. <{2, 3},{1, 2, 3},{1, 3}> means that a user purchased 2 and 3 together, then 1, 2, and 3, and lastly 1, and 3.

A list of user transaction sequences makes up a **sequence database** $SDB = \{S_1, S_2, \ldots, S_q\}$.

Problem statement. Given a sequential database *SDB* and a minimum support threshold δ, the goal of SPM is to find out all frequent patterns with supports $\geq \delta$ in *SDB*.

4.2 Challenges and Goals

Patterns in SPM are classified into two categories to facilitate the explanation of the issue. Patterns end with itemsets containing a single item, and the others end with two or more items. The former is called *s*-tailed patterns (short for singleton-tailed patterns), and the latter is called *ns*-tailed patterns (short for non-singleton-tailed patterns). For example, let $X = \langle\{2, 4\}, \{3\}\rangle$ and $Y = \langle\{5\}, \{1, 3\}\rangle$ be two patterns, then *X* is an *s*-tailed pattern and *Y* is an *ns*-tailed pattern.

According to the categorizing above, clickstream patterns comprise multiple 1-itemsets. This indicates that clickstream patterns are always *s*-tailed. A sequential pattern, on the other hand, can be either *s*-tailed or *ns*-tailed. Thus, Pseudo-IDList alone is not suitable for SPM unless there are adjustments.

4.3 Adaptation of Pseudo-IDList

The author proposes the SUI (Sequential pattern mining Using IDList) algorithm by making the following changes to the original CUP algorithm:

- Data-IDLists are still used for *ns*-tailed patterns, as Pseudo-IDLists do not work for *ns*-tailed patterns.
- Pseudo-IDLists are still used for *s*-tailed patterns.

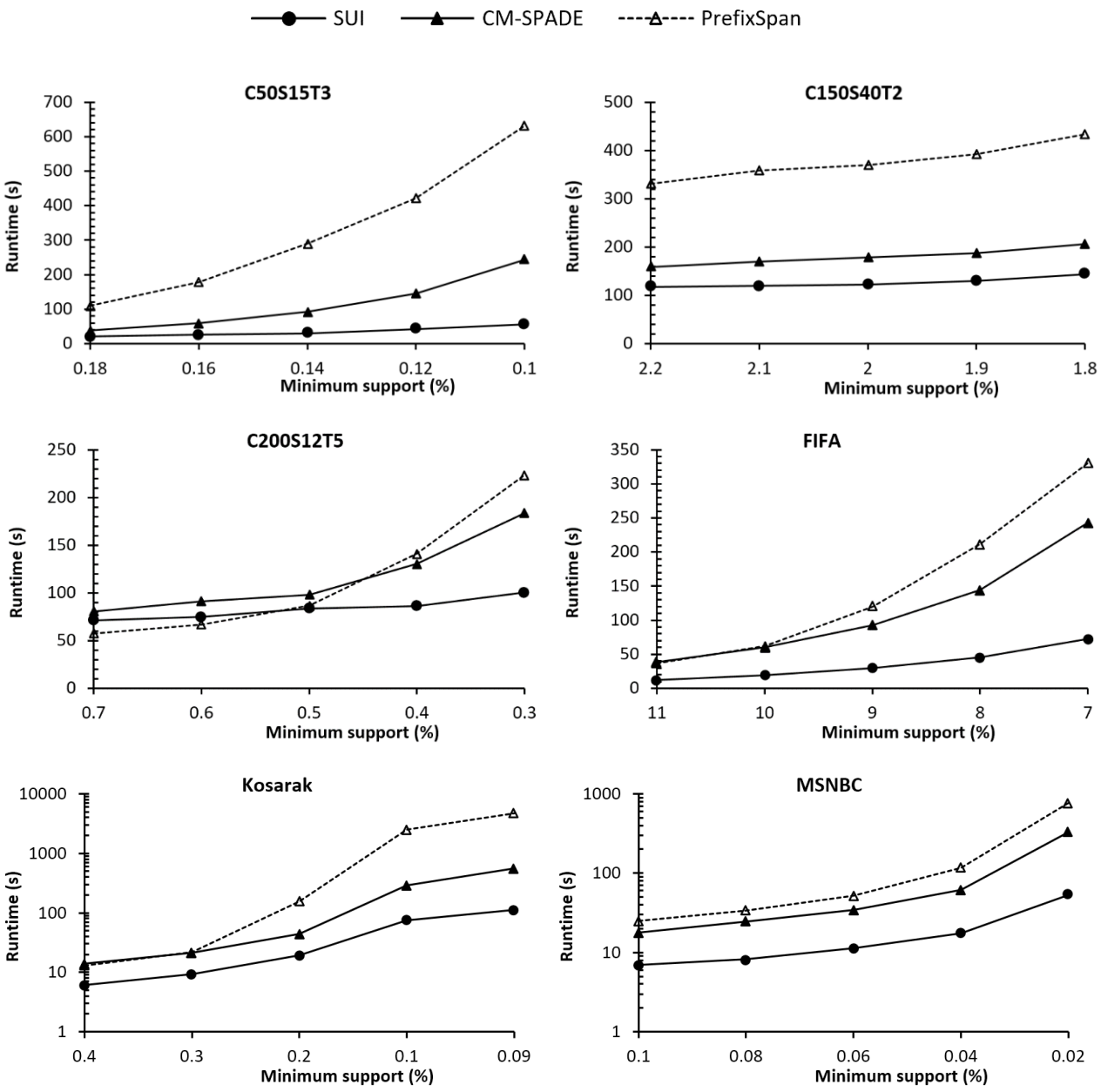

Fig. 4.1. Runtime on six databases.

4.4 Experiment

The SUI algorithm is evaluated by comparing its performance and scalability to PrefixSpan [3] and CM-SPADE [6], which are two state-of-the-art SPM algorithms. The PrefixSpan algorithm belongs to the horizontal group, while the CM-SPADE algorithm belongs to the vertical group. According to [6], both are very effective algorithms.

4.4.1 Environment Setup

All algorithms are implemented in Java, and PrefixSpan (version 2016) and CM-SPADE come from the SPMF package [17]. The computer used for the experiments was installed with Intel Core I7 8750H 2.2GHz, 16GB RAM, Windows 10 64-bit, and JDK 8.

Three real-life databases are used: Kosarak and MSNBC, which are considered big databases, and FIFA, which is a medium database. MSNBC is also considered a dense database, while FIFA and Kosarak are sparser. Three synthetic test databases are also used for the experiments: C50S15T3, C150S40T2, and C200S12T5. The databases' characteristics are summarized in Table 1.2.

4.4.2 Performance Analysis

According to the results in Fig. 4.1, SUI performs better than both PrefixSpan and CM-SPADE regarding runtime. SUI's runtime increases more gradually compared to the other two algorithms towards smaller values of δ. For example, PrefixSpan has a lower runtime than the other two on the C200S12T5 dataset at δ = 0.7% and 0.6%. However, because PrefixSpan has a steeper increase in runtime, SUI and CM-SPADE are faster than PrefixSpan when $\delta \leq 0.5\%$. At δ = 0.3%, SUI has the fastest runtime among the three.

5. WEIGHTED CLICKSTREAM PATTERN MINING

The methods in previous chapters only use support values to pick the interesting patterns. Additionally, these methods are sequential-oriented, running only on one thread. This chapter looks into a slightly different problem called the frequent weighted clickstream pattern. This problem provides an alternative scoring measurement using the weight factor. Furthermore, this chapter looks into the parallel approaches to improve the performance of the mining process. The content of this chapter is overlapped with the author's previous research presented in two journal articles ([A.4], [A.8]).

5.1 Problem Definition

Let there be a set of symbols $I = \{i_1, i_2, \ldots, i_n\}$ representing different events (e.g., a click on a website's URL) and a set of positive real values $W = \{w_1, w_2, w_3, \ldots, w_n\}$, in which each value w_i is a weight indicating the importance of the event $x_i \in I$.

UCID	User clickstream
1	<1, 3, 3, 4, 6>
2	<4, 3, 2, 1>
3	<6, 2, 6, 3, 2>
4	<5, 1, 3, 2, 3, 2, 6>
5	<1, 2, 3, 5>

Fig. 5.1. An example of horizontal clickstream database weighted clickstream pattern mining.

Event	Weight
1	0.5
2	0.9
3	0.2
4	0.4
5	0.7
6	0.6

UCID	Weight
1	0.38
2	0.5
3	0.64
4	0.57
5	0.58
CDB_w	2.67

Fig. 5.2. An example of event weights (on the left) and weights of user clickstreams (on the right).

Assuming that *Y* is a user clickstream, the **weight** of *Y* is defined as follows:

$$cw(Y) = \frac{\sum_{c_i \in Y} w_i}{|Y|} \quad (1)$$

The weight of a clickstream database *CDB*, denoted by CDB_w, is calculated as follows:

$$CDB_w = \sum_{Y \in CDB} cw(Y) \quad (2)$$

The weighted support of a clickstream pattern *Y* is the sum of the weights of user clickstreams, where *Y* appears, divided by the database weight. Formally, it is defined as:

$$ws(Y) = \frac{\sum_{X \in CDB \wedge Y \subseteq X} cw(X)}{CDB_w} \quad (3)$$

A clickstream *Y* is called a **frequent weighted clickstream pattern** if $ws(Y) \geq \omega$. For a given clickstream database *CDB*, the **problem of weighted clickstream pattern mining** is to find all frequent weighted patterns.

5.2 Challenges and Goals

Most existing algorithms used the anti-monotonicity property (i.e., downward closure) as their foundation. The anti-monotonicity property allows algorithms to discard redundant search spaces and helps reduce the runtime. When weights are integrated into mining patterns, one issue is the inability to preserve anti-monotonicity without additional restrictions.

This chapter aims to propose effective algorithms for mining weighted clickstream patterns. To achieve this:

- The proposed weight formula presented in Section 5.1 is incorporated into the proposed algorithm, which maintains the original weights of events.
- Various parallel processing techniques, including the newly proposed method, are integrated to optimize runtime performance.

5.3 Mining Weighted Clickstream Pattern: Compact-SPADE

This section presents the author's proposed Compact-SPADE algorithm in more detail with the pseudo-code in Algorithm 5.1. Similar to CUP (section 3.7 on page 16), Compact-SPADE traverses recursively in a depth-first manner, commencing at node $\emptyset$ (line 5) (level zero). Assuming that P_i is the first 1-pattern of $[\emptyset]$, the algorithm moves to node P_i (level one). However, no child nodes have yet been discovered for node P_i. To traverse further, the next set of 2-patterns of $[P_i]$ (level two) is created through combining P_i with every other 1-pattern in $[\emptyset]$, including P_i itself (lines 7 to 13). The WCMAP, a weighted version of CMAP in [6], is used here to quickly filter out some candidates violating the anti-monotonicity constraint (line 10). Any 2-patterns that satisfy ω are retained and assigned to their respective equivalence classes based on their prefix (lines 11 to 13).

This process repeats, with each pattern being expanded into larger patterns by combining with others. The traversal continues until all possible frequent patterns have been generated and no further expansion is possible. The algorithm then backtracks and repeats the steps for the next 1-pattern in $[\emptyset]$ until all possible patterns are explored.

Algorithm 5.1. Compact-SPADE.

Input: minimum weighted support threshold $\boldsymbol{\omega}$ and the clickstream database *CDB*

Output: the set $\boldsymbol{\mathcal{F}}$ containing all frequent weighted clickstream patterns in *CDB*

1: $\mathcal{F} \leftarrow \emptyset$, a global variable (that is shared across methods)
2: Scan CDB to determine the equivalence class $[\emptyset]$ containing all frequent weighted 1-patterns and their respective WICLists
3: $\mathcal{F} \leftarrow \mathcal{F} \cup [\emptyset]$

4: Populate $\mathcal{W}$, a global WCMAP, from $[\emptyset]$
5: Execute Node-Expand() with class $[\emptyset]$

Algorithm 5.2. Node-Expand.

Input: an equivalence class $[\boldsymbol{X}]$

6: **for** each pattern P_i in class $[X]$ **do**
7: $[P_i] \leftarrow \emptyset$
8: **for** each pattern P_j in class $[X]$ **do**
9: $\alpha \leftarrow$ the pattern generated from P_i and P_j and P_i is α's prefix
10: Use WCMAP $\mathcal{W}$ and WICList of α to determine if α is frequent or not
11: **if** α is frequent **then**
12: $[P_i] \leftarrow [P_i] \cup \alpha$
13: $\mathcal{F} \leftarrow \mathcal{F} \cup \alpha$
14: Execute Node-Expand() with class $[P_i]$ ▷ *Recursively expand* $[P_i]$ *further*

5.4 Mining Weighted Clickstream Pattern: Parallel Approaches

This section presents four parallelism methods for Compact-SPADE. They consist of two main components: 1) the core algorithm, Compact-SPADE, and 2) a load balancing method.

5.4.1 Static Load Balancing

Static load balancing is a common method for parallelism due to its ease of implementation. It divides tasks into several partitions corresponding to the number of assigned threads. A thread processes each partition, and none of the threads interfere with other tasks. The static load balancing splits all the 1-class elements in a 0-class $[\emptyset]$ into n partitions, starting from the left-most to the right-most unprocessed 1-class. For example, assuming that 0-class $[\emptyset]$ = {[1], [2], [3], [5], [6]} and the number of threads n = 3, then the first thread expands {[1], [2]}, the second processes {[3], [5]} and the third works on {[6]}. The parallel algorithm that uses this static load balancing is called StaticPCompact-SPADE.

5.4.2 Horizontal Dynamic Load Balancing

Another common load balancing method is horizontal (dynamic) load balancing, which is used in [18, 19]. Unlike static load balancing, horizontal load balancing does not split the tasks into several workloads. Instead, it schedules unprocessed 1-classes and assigns each to an idle thread. If there is no idle thread, it waits until a thread is available again to assign the next unprocessed 1-class. The algorithm with horizontal load balancing is called HPCompact-SPADE (Horizontal Parallel Compact-SPADE).

For example, considering that 0-class [∅] = {[1], [2], [3], [5], [6]} and the number of threads n = 3. The first thread processes {[1]}, the second processes {[2]} and the third works on {[3]}. Two unprocessed 1-classes are {[5], [6]}, and since all threads are taken, the load balancer goes into a waiting state. If the second thread finished its work first, the load balancer will assign {[5]} to the second thread. After that, when the first thread becomes available, it will be assigned with {[6]}.

5.4.3 Recursive Depth Dynamic Load Balancing

In this sub-section, the author proposes (recursive) depth (dynamic) load balancing based on the depth-first-search. The parallel Compact-SPADE that uses this load balancing is called DPCompact-SPADE.

Depth load balancing has two characteristics that help generate and distribute workloads. Firstly, it uses recursion to generate parallelism. Secondly, the tasks assigned to a thread can be a class at any level. Any thread can transfer its partial workload, which is any k-class, to other idle threads so all threads remain active until the end of the work.

For example, consider the 0-class [∅] = {[1], [2], [3], [5], [6]} and a system with three threads (t_1, t_2, t_3). Initially, DPCompact-SPADE assigns t_1 to start working on [∅], which then submits [<1>] to the work queue Q. The load balancer transfer [<1>] to t_2, while t_1 continues processing [<2>]. Because t_3 is still free, the load balancer transfers [<1>, <2>], which is a part of [<1>] to t_3, while t_2 works on the remaining part of [<1>]. The process transfers part of an unprocessed class to free threads and repeats until all frequent patterns are enumerated.

5.4.4 Adaptive Dynamic Load Balancing

Adaptive (dynamic) load balancing is based on the author's proposed heuristic sampling called join estimation via sampling in the following paragraphs.

Join estimation via sampling. This idea aims to switch the parallel method to the more efficient one, depending on the distribution of patterns and joins in a database.

Let $join_k$ be the number of joins between k-patterns, $join_S$ be the number of all joins, $SC = \frac{\sum_{k=1}^{3} join_k}{join_S}$, and γ be the proportional ratio that is given by the user. If $SC > \gamma$, the load balancing switches from the default DPCompact-SPADE to HPCompact-SPADE. In the experiments, γ = 70% is used. That is if the joins of 1,2,3-patterns take more than 70% of all joins, horizontal load balancing is preferred over depth load balancing.

In real situations, the distribution is not always known in advance. Therefore, the author proposes a heuristic sampling to estimate the distribution and the *SC* value. The sample *S* is decided as follows:

- 1% of 1-classes in [∅] are picked to make *S*. Additionally, the size of *S* must be at least two and no more than 50.
- The sample *S* must include 1-class [α] where α is the 1-pattern with the highest weighted support among all 1-patterns in [∅].
- The remaining 1-classes are selected from left-most to right-most in [∅].

5.5 Experiment

To evaluate the effectiveness and efficiency of DPCompact-SPADE and APCompact-SPADE, they are compared with CM-SPADE [6], Parallel DBV [20], StaticPCompact-SPADE, and HPCompact-SPADE. Two algorithms, CM-SPADE [6] and Parallel DBV [20], are the state-of-the-art algorithms for SPM.

5.5.1 Environment Setup

To evaluate the proposed algorithm, experiments were conducted on a computer running Windows 8.1 64 bits, with Java 8, 16 GB of RAM, an Intel Core i7-8750H 2.20 GHz processor with six physical cores, and hyper-threading technology.

Experiments were done using six benchmark databases. Their characteristics are presented in Table 1.2. FIFA, BIBLE, and SIGN are small-to-medium clickstream databases, while Korasak, Chainstore, and D9000S4 are big databases. Chainstore and D9000S4 are short pattern databases, while the others are long pattern databases.

5.5.2 Performance Analysis

It is observed in Fig. 5.3 that APCompact-SPADE, DPCompact-SPADE (the two algorithms proposed by the author), and HPCompact-SPADE achieve better overall runtime than StaticPCompact-SPADE, CM-WSPADE, WDBV, and Parallel WDBV. However, their runtimes can fluctuate depending on database types. The fastest algorithm on short databases can be the slowest on normal databases.

On the four normal databases (FIFA, BIBLE, SIGN, and Korasak), the general order based on runtimes is **DPCompact-SPADE** ≥ **APCompact-SPADE** > HPCompact-SPADE > StaticCompact-SPADE > Compact-SPADE > Parallel WDBV > CM-WSPADE > WDBV.

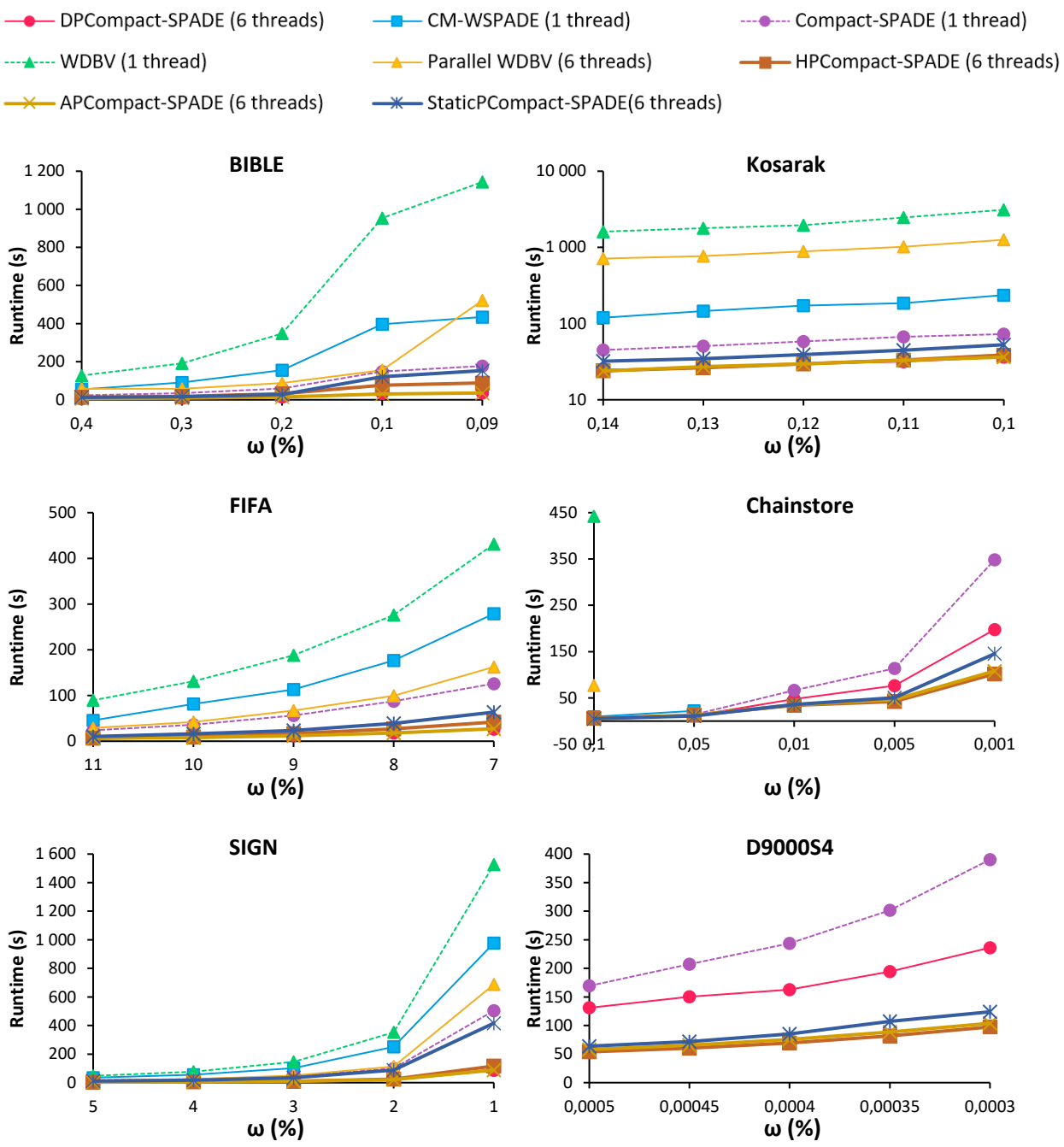

Fig. 5.3. Runtimes for various minimum weighted support values.

However, on the two short pattern databases, Chainstore and D9000S4, their performance switches around as HPCompact-SPADE becomes the fastest, followed by APCompact-SPADE, StaticCompact-SPADE, DPCompact-SPADE, Compact-SPADE, CM-WSPADE, Parallel WDBV, and finally WDBV.

APCompact-SPADE is neither faster than DPCompact-SPADE on long pattern databases nor HPCompact-SPADE on short pattern databases. This is because its sampling process takes small runtimes to estimate the distributions of joins. The time required for sampling is around 0% to 6.5%. The longest sampling time is 6.5% on D9000S4. Thus, on every test database, APCompact-SPADE ranks as

the second-fastest algorithm, proving to be consistent in runtime. DPCompact-SPADE takes the lead on four normal databases, and HPCompact-SPADE is the fastest in the two short pattern databases.

6. INCREMENTAL CLICKSTREAM PATTERN MINING

In the previous sections, the proposed algorithms are considered batch algorithms as they operate on static databases. This chapter looks into a different angle of mining in dynamic databases, in which the data grows over time. The content of this chapter is overlapped with the author's previous research presented in two journal articles ([A.1], [A.2]).

6.1 Problem Definition

Assume that a new set of data, referred to as Δdb, is being incorporated into the existing database db to make a new database DB. In this context, Δdb is an incremental database, while db represents the original database. The combined result of this integration is called the updated database, denoted as DB (thus, $DB = db + \Delta db$). It is important to note that Δdb must include a set of records not present in db previously (i.e., new user clickstreams with new *CID*s).

The function $supp_{db}(X) = |\{Y \mid \forall Y \in db: Y \text{ contains } X\}|$ denotes pattern X's support count in the database db, and it indicates the number of user clickstreams containing X. Meanwhile, the relative support of X in db is denoted as $rsupp_{db}(X) = \frac{supp_{db}(X)}{|db|}$. This value measures how prevalent pattern X is concerning the entire database size.

Let ω_f be a relatively frequent support threshold given by users. If $rsupp_{db}(X) \geq \omega_f$, pattern X is frequent in the database db. For example, given $\omega_f = 0.3$ and $X = \langle 3, 6 \rangle$. Because it is contained in 3 clickstreams with *CID*s = {1, 4, 8}, $supp_{db}(X) = 3$. The pattern X is then considered frequent because $rsupp_{db}(X) = \frac{supp_{db}(X)}{|db|} = 3/10 = 0.3 \geq \omega_f$, where $\omega_f = 0.3$.

db	
UCID	User clickstream
1	1, 3, 6, 1, 1, 4
2	3, 5, 4, 8

Δdb	
UCID	User clickstream
11	2, 3, 4, 6, 5, 1, 8, 6
12	1, 4, 8, 10, 6

3	1, 2, 4
4	3, 4, 5, 6, 8, 1, 3
5	1, 3, 7, 9
6	1, 6, 4, 7, 6
7	1, 4, 1, 5, 8
8	3, 1, 6, 2, 2, 4, 1, 1, 6
9	1, 9, 8, 3
10	1, 8, 7

Fig. 6.1. A horizontal clickstream database, an incremental database, and the updated database.

Let ω_{pf} be a relative pre-frequent support threshold. If $\omega_f > rsupp_{db}(X) \geq \omega_{pf}$, X is pre-frequent. Otherwise, if $\omega_{pf} > rsupp_{db}(X)$, then X is infrequent. Let $\Delta\omega = \omega_f - \omega_{pf}$, $\Delta\omega$ is called a differential threshold.

Problem statement. Given a **minimum support threshold** δ, incremental clickstream mining aims to discover frequent clickstream patterns while new data is added over time.

6.2 Challenges and Goals

Mining incremental databases using non-incremental methods can lead to high resource consumption and long response times. This occurs because each update requires a full rerun of the mining algorithm, which becomes increasingly inefficient as databases grow larger. For example, in scenarios such as web log data, where records are frequently added, non-incremental methods re-extract patterns from scratch, delaying real-time analysis.

To address this, the pre-large concept [14] introduces a buffering zone to store previous incremental updates, reducing the need for full rescans by triggering them only when certain conditions are met. Multiple studies have applied this concept for SPM and FIM, but not CPM.

This chapter aims to propose algorithms suitable for mining frequent clickstream patterns in incremental situations based on the pre-large concept. The author proposes three different algorithms; each latter is shown to be more effective. The three algorithms are PF-CUP, PSB-CUP, and PSB-CUP+.

Table 6.1. Nine cases and the possible results.

Case	Original database – Incremental database	Result (Updated original database)
1	Frequent - frequent	Always frequent
2	Frequent - pre-frequent	Either frequent or pre-frequent, depending on the existing data
3	Frequent - infrequent	Frequent, pre-frequent, or infrequent, depending on the existing data
4	Pre-frequent - frequent	Either frequent or pre-frequent, depending on the existing data
5	Pre-frequent - pre-frequent	Always pre-frequent
6	Pre-frequent - infrequent	Either pre-frequent or infrequent, depending on the existing data
7	Infrequent - frequent	Either pre-frequent or infrequent, re-enumerating on the original database if necessary
8	Infrequent - pre-frequent	Either pre-frequent or infrequent, re-enumerating on the original database if necessary
9	Infrequent - infrequent	Always infrequent

6.3 Pre-Large Concept

The pre-large concept [14] introduces two values, *f*, and *c*, to reduce the frequency of rescans during incremental mining. f represents the maximum number of new records that can be inserted without requiring a rescan, while c is the number of records added since the last rescan. If $c > f$, a rescan is performed, f is recalculated, and c is reset to zero. Otherwise, the algorithm updates the current database incrementally.

The $\boldsymbol{f}$ value is calculated as (4):

$$f = \left\lfloor \frac{\Delta\omega * |db|}{1-\omega_f} \right\rfloor = \left\lfloor \frac{(\omega_f - \omega_{pf}) * |db|}{1-\omega_f} \right\rfloor \quad (4)$$

The algorithm tracks frequent and pre-frequent patterns. Nine possible cases may arise during incremental mining (as detailed in Table 6.1). Cases 1, 5, 6, 8, and 9 retain the frequent patterns, while cases 2 and 3 may involve frequent pattern deletion. For cases 7 and 8, rescans are required only when $c > f$ to ensure no missing information for frequent patterns will occur.

6.4 Algorithm: Pre-Frequent Clickstream Mining Using Pseudo-IDList (PF-CUP)

This section presents the author's proposed PF-CUP (Pre-Frequent Clickstream mining Using Pseudo-IDList) and its components. The general flow of the PF-CUP algorithm can be split into three main stages: preparing data, discovering patterns, and producing output.

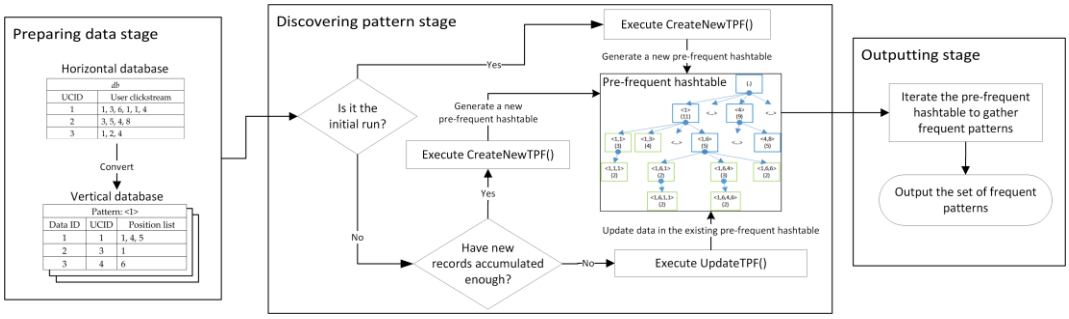

Fig. 6.2. Overall system flow of PF-CUP.

In the preparing data stage, input data are converted into suitable data for the algorithm to process. After that, the algorithm uses the converted data to discover patterns and keeps necessary information for later incremental updates. In the last phase, the algorithm collects frequent clickstream patterns and returns them to users. The flow of PF-CUP is summarized in Fig. 6.2. The step-by-step details of the algorithm flow is described in the full version of the thesis.

6.4.1 Pre-Frequent Hash Table

The main idea of a pre-frequent hash table T_{pf} is to store and keep the necessary information of frequent and pre-frequent patterns for the current mining process and future incremental mining. To do so, T_{pf} stores pattern information under pattern nodes. Each of them has the following fields:

- Pattern: the *k*-pattern the current node represents.
- Support: the support count of the pattern.

Each node represents a pattern on the pattern tree in the form of $\{X, (n)\}$, where X is the pattern and n is the pattern support count.

6.5 Algorithm: PSB-CUP and PSB-CUP+

The **PSB-CUP** algorithm [A.1] was developed to enhance PF-CUP by introducing the **progressive search border (PSB)** mechanism and the **partial imbalance join** strategy. PSB-CUP aims to improve runtime by reducing unnecessary pattern generation and focusing on keeping only relevant pattern data during incremental updates.

The progressive search border limits the generation of redundant patterns by maintaining a "border" of pre-frequent patterns that are only expanded when necessary. This border consists of *k*-pattern nodes where each pattern has either pre-frequent or frequent parents. When the border expands due to an incremental update, only the buffering candidates - child patterns of the border nodes - are processed. This selective expansion reduces unnecessary computations and avoids recalculating patterns that are not yet critical to the mining process.

However, one challenge introduced by PSB is the inability to generate IDLists using traditional methods, as only the main parent of a candidate pattern has its IDList available. To address this, PSB-CUP implements the partial imbalance join, which allows the creation of a candidate's IDList by joining the main parent's IDList with a frequent 1-pattern IDList. For example, consider the pattern <1, 1, 3>, where the parents are <1, 1> and <1, 3>. If the IDList for <1, 1> is available but not for <1, 3>, the partial imbalance join creates the IDList for <1, 1, 3> by joining the IDList of <1, 1> with the IDList of the frequent 1-pattern <3>. This ensures that the algorithm can still calculate support values efficiently without storing redundant amounts of intermediate data.

Building upon PSB-CUP, **PSB-CUP+** [A.1] introduces the **recursive imbalance join** to address the high memory consumption caused by storing border pattern IDLists. In recursive imbalance join, the candidate's IDList is built step-by-step by recursively joining 1-pattern IDLists. For instance, the IDList of pattern <1, 3, 6> is reconstructed by first joining the IDLists of <1> and <3> to generate the IDList for <1, 3>, and then combining it with the IDList of <6>. This recursive approach reduces memory usage as PSB-CUP+ no longer needs to store IDLists of border nodes or maintain links between parent nodes.

With these improvements, PSB-CUP+ achieves better scalability and efficiency in memory management than its predecessor. The algorithm only needs three fields in its data structure: support, pattern, and child list.

6.6 Experiments

The author's algorithm, CUP (Chapter 3), is used to compare with this chapter's incremental algorithms because CUP was used to propose PF-CUP, PSB-CUP,

and PSB-CUP+. They are also compared with a recent incremental clickstream mining algorithm, Eff-InCMUB [21].

6.6.1 Environment Setup

All algorithms were implemented in Java and executed on JDK 1.8 in a Windows 10 64-bit environment. These implementations utilized various functions from the open-source SPMF package [17]. A maximum heap size of 10 GB was set for those algorithms. The hardware was an Intel Core I-8750H (2.2 GHz processor) with 16 GB of RAM.

Four clickstream databases are employed for the experiments: BIBLE, FIFA, Kosarak, and D900S20N3. BIBLE and FIFA are medium size real-life databases. The remaining two, Kosarak and D900S20N3, are considered large databases. Kosarak contains real-life data, and D900S20N3 is a synthetic database. The four databases' characteristics are summarized in Table 1.2.

6.6.2 Performance Analysis

The standard test starts with initial ω_f values and then reduces them four times. Furthermore, the IR and $\Delta\omega$ were kept constant and are indicated in the titles of charts.

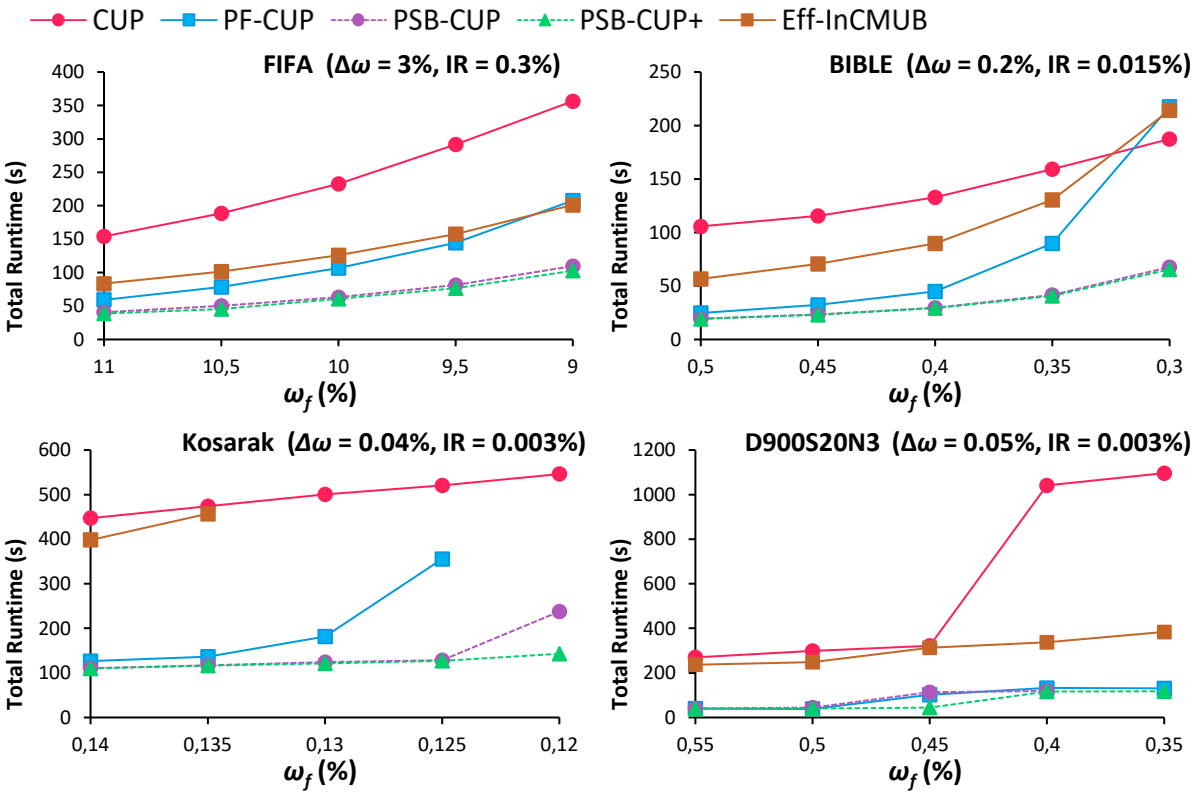

Fig. 6.3. Algorithms' total runtimes on four databases.

Fig. 6.3 shows the overall algorithms' runtime in order, which is PSB-CUP+ ≥ PSB-CUP > PF-CUP > Eff-InCMUB > CUP. When ω_f decreases, PSB-CUP+ is more stable regarding runtime increase than the other three. Meanwhile, the performance of PF-CUP and Eff-InCMUB can sharply fall and become slower than CUP. For example, on BIBLE and Kosarak, the PF-CUP and Eff-InCMUB runtime curves rise when the relative minimum support threshold ω_f drops below 0.35%. In particular, PF-CUP and Eff-InCMUB cannot operate after relative minimum support threshold ω_f was set to 0.125% and 0.135%, respectively, on Kosarak due to exceeding the memory limit. PF-CUP's performance degradation was due to the large number of nodes it has to keep in T_{pf}. This made the algorithm need more time and memory to traverse the search space. In this case, the benefits of shortening the incremental runs with the pre-large concept in PF-CUP were outweighed by normal non-incremental runs of CUP.

7. CONTRIBUTION AND CONCLUSION OF THE THESIS

Data mining has become increasingly important as time goes by, and many areas have been left with little to no attention. CPM is a specific problem that fell into such a situation because it was mostly observed as the SPM problem. Therefore, this thesis attempts to explore CPM and its variants by adopting, improving, and proposing algorithms for problems revolving around clickstreams.

The topic is introduced in chapter one among the reasons for picking this topic. The chapter also introduces the thesis' proposed aims and scope, along with the main method for evaluation.

The following chapter looks into the related work and the state of CPM. Several derived problems of FIM and SPM are briefly introduced to give readers a sense of how the two topics have been focused on compared to CPM. Furthermore, some well-known algorithms for SPM are briefly discussed. Those algorithms can be categorized into two groups, vertical and horizontal format, each with its own advantages.

The third chapter is the author's study of the most basic problem, the frequent CPM problem. This chapter fulfilled the first aim of this thesis: "Propose approaches that can exploit certain clickstreams' characteristics for mining frequent clickstream patterns." This goal was achieved through the author's proposed algorithm CUP because it was built with the proposed Pseudo-IDList and DUB. The Pseudo-IDList and DUB made use of the characteristics of the clickstream to reduce memory consumption and improve runtimes for CPM. More specifically, they exploited the characteristic of which two events in a clickstream cannot happen simultaneously. As a result, the CPU algorithm outperformed other algorithms in the test datasets. The algorithms and

experimental results were published in a conference [A.13] and a highly-rated journal [A.7].

Chapter four addresses the second aim of this thesis: "Extend the proposed algorithm to mine sequential patterns." The main contribution of this chapter is the author's proposed SUI algorithm, an effective method for mining sequential patterns. Since the SPM problem is more general than the CPM problem, the original CUP algorithm required modifications to work with SPM. These modifications introduce distinct ways of handling two types of patterns (*s*-tailed and *ns*-tailed) through Pseudo-IDList and Data-IDList. The SUI algorithm outperformed other SPM algorithms in the author's experiment. The author's research was published in a highly-rated journal [A.3] and a conference [A.12].

In the fifth chapter, the third aim, "Integrate weight factor to give another option for ranking patterns," and the fourth aim, "Improve performance by integrating parallelism into existing clickstream pattern mining algorithms," were fulfilled. The first contribution of this chapter is a ranking method using average weight, which the author proposed as an alternative approach to finding frequent patterns. The novel Compact-SPADE was then created as the baseline algorithm in this chapter by integrating the weight formula. The second contribution is that different parallel strategies were proposed to improve mining performance. The author created the novel APCompact-SPADE based on depth-first search and adaptive load balancing technique using heuristic sampling. APCompact-SPADE sacrificed a bit of performance in exchange for more reliable runtime and effectiveness. The studies of this chapter were presented in two highly-rated journals ([A.4], [A.8]).

The sixth chapter fulfilled the last aim of this thesis: "Propose methods to tackle the situation where new clickstream data is being added on a daily basis." This chapter looked into the problem of incremental CPM, in which data is added over time. The main contribution of this chapter is three novel algorithms that deal with this situation. The author started by transforming the non-incremental algorithm CUP in chapter three into the incremental algorithm PF-CUP. Afterward, PSB-CUP and PSB-CUP+ were proposed to further improve the performance of PF-CUP with the novel concepts of "progressive search border," "partial imbalance join," and "recursive imbalance join." The content of this chapter was published in two highly rated journals ([A.1], [A.2]).

The thesis is an attempt by the author to study the arena of pattern mining that has not yet been well-studied by other researchers. This thesis proved some evidence that clickstream characteristics could be exploited to craft better algorithms and improve mining performance for the CPM problem. The author, however, did not claim that his work outperformed other algorithms in every case. The author hopes this work will enable other researchers to study more on the CPM topic.

REFERENCES

[1] AGRAWAL, R., IMIELIŃSKI, T. and SWAMI, A. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*. Online. June 1993. Vol. 22, no. 2, p. 207–216. DOI 10.1145/170036.170072.

[2] AGRAWAL, R. and SRIKANT, R. Mining sequential patterns. In : *Proceedings of the International Conference on Data Engineering (ICDE)*. 1995. p. 3–14. DOI 10.1109/ICDE.1995.380415.

[3] PEI, J., HAN, J., CHEN, Q., HSU, M.-C., MORTAZAVI-ASL, B., PINTO, H. and DAYAL, U. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In : *Proceedings of the International Conference on Data Engineering (ICDE)*. 2001. p. 215–224. ISBN 0-7695-1001-9. DOI 10.1109/ICDE.2001.914830.

[4] ZAKI, M.J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*. 2001. Vol. 42, no. 1–2, p. 31–60. DOI 10.1023/A:1007652502315.

[5] AYRES, J., FLANNICK, J., GEHRKE, J. and YIU, T. Sequential pattern mining using a bitmap representation. In : *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, New York, USA : ACM Press, 2002. p. 429–435. ISBN 158113567X. DOI 10.1145/775047.775109.

[6] FOURNIER-VIGER, P., GOMARIZ, A., CAMPOS, M. and THOMAS, R. Fast vertical mining of sequential patterns using co-occurrence information. In : *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 2014. p. 40–52. ISBN 9783319101590. DOI 10.1007/978-3-319-06608-0_4.

[7] YUN, U. and LEGGETT, J.J. WSpan: Weighted sequential pattern mining in large sequence databases. In : *Proceedings of the International IEEE Conference Intelligent Systems*. Online. 2006. p. 512–517. ISBN 1-4244-0195-X. DOI 10.1109/IS.2006.348472.

[8] YUN, U. Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. *Information Sciences*. 2007. Vol. 177, no. 17, p. 3477–3499. DOI 10.1016/j.ins.2007.03.018.

[9] YUN, U., LEE, G. and RYU, K.H. Mining maximal frequent patterns by considering weight conditions over data streams. *Knowledge-Based Systems*. 2014. Vol. 55, p. 49–65. DOI 10.1016/j.knosys.2013.10.011.

[10] LEE, G., YUN, U., RYANG, H. and KIM, D. Erasable itemset mining over incremental databases with weight conditions. *Engineering Applications of Artificial Intelligence*. 2016. DOI 10.1016/j.engappai.2016.03.003.

[11] YUN, U., PYUN, G. and YOON, E. Efficient mining of robust closed weighted sequential patterns without information loss. *International Journal on Artificial Intelligence Tools*. 2015. Vol. 24, no. 01, p. 1550007. DOI 10.1142/s0218213015500074.

[12] CHEUNG, D.W., HAN, J., NG, V.T. and WONG, C.Y. Maintenance of discovered association rules in large databases: An incremental updating technique. In : *Proceedings - International Conference on Data Engineering*. 1996. DOI 10.1109/icde.1996.492094.

[13] HONG, T.P., LIN, C.W. and WU, Y.L. Incrementally fast updated frequent pattern trees. *Expert Systems with Applications*. 2008. Vol. 34, no. 4. DOI 10.1016/j.eswa.2007.04.009.

[14] HONG, T.P., WANG, C.Y. and TAO, Y.H. A new incremental data mining algorithm using pre-large itemsets. *Intelligent Data Analysis*. 2001. Vol. 5, no. 2. DOI 10.3233/ida-2001-5203.

[15] TARUS, J.K., NIU, Z. and KALUI, D. A hybrid recommender system for e-learning based on context awareness and sequential pattern mining. *Soft Computing*. 2018. Vol. 22, no. 8. DOI 10.1007/s00500-017-2720-6.

[16] ZAKI, M.J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*. 2001. Vol. 42, no. 1–2, p. 31–60. DOI 10.1023/A:1007652502315.

[17] FOURNIER-VIGER, P., LIN, J.C.W., GOMARIZ, A., GUENICHE, T., SOLTANI, A., DENG, Z. and 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*. Online. 2016. p. 36–40. ISBN 9783319461304. DOI 10.1007/978-3-319-46131-1_8.

[18] ZAKI, M. Parallel Sequence Mining on Shared-Memory Machines. *Journal of Parallel and Distributed Computing*. 2001. Vol. 61, no. 3, p. 401–426. DOI 10.1006/jpdc.2000.1695.

[19] DEMIRIZ, A. webSPADE: a parallel sequence mining algorithm to analyze web log data. In : *Proceedings of the International Conference on Data Mining*. 2002. p. 755–758. ISBN 0-7695-1754-4. DOI 10.1109/ICDM.2002.1184046.

[20] HUYNH, B., VO, B. and SNASEL, V. An efficient method for mining frequent sequential patterns using multi-Core processors. *Applied Intelligence*. 2017. Vol. 46, no. 3, p. 703–716. DOI 10.1007/s10489-016-0859-y.

[21] VO, B., NGUYEN, H.C., HUYNH, B. and LE, T. Efficient Methods for Clickstream Pattern Mining on Incremental Databases. *IEEE Access*. 2021. Vol. 9. DOI 10.1109/ACCESS.2021.3131577.

THE AUTHOR'S LIST OF PUBLICATIONS

Journals with Impact Factor

[A.1] **HUYNH, H.M.**, PHAM, N.N., OPLATKOVA, Z.K., NGUYEN, L.T.T., THANH NGUYEN, N., YUN, U. and VO, B. Incremental clickstream pattern mining with search boundaries. *Information Sciences*. 2024. Vol. 662. (2024, Q1, IF 8.1).

[A.2] **HUYNH, M.H.**, VO, B., OPLATKOVÁ, Z.K. and PEDRYCZ, W. An Approach for Incremental Mining of Clickstream Patterns as a Service Application. *IEEE Transactions on Services Computing*. (2023, D1, IF 8.1)

[A.3] **HUYNH, H.M.**, NGUYEN, L.T.T., PHAM, N.N., OPLATKOVÁ, Z.K., YUN, U. and VO, B. An efficient method for mining sequential patterns with indices. *Knowledge-Based Systems*. (2022, Q1, IF 8.8)

[A.4] **HUYNH, H.M.**, NGUYEN, L.T.T., VO, B., OPLATKOVÁ, Z.K., FOURNIER-VIGER, P. and YUN, U. An efficient parallel algorithm for mining weighted clickstream patterns. *Information Sciences*. 2022. Vol. 582. (2022, D1, IF 8.1)

[A.5] HUYNH, B., NGUYEN, L.T.T., **HUYNH, H.M.**, KOZIERKIEWICZ, A., YUN, U., OPLATKOVÁ, Z.K. and VO, B. A Novel Approach for Mining Closed Clickstream Patterns. *Cybernetics and Systems*. (2021, Q3, IF 1.7)

[A.6] BUI, T., NGUYEN, T., **HUYNH, H.M.**, VO, B., CHUN-WEI LIN, J. and HONG, T.P. Multiswarm Multiobjective Particle Swarm Optimization with Simulated Annealing for Extracting Multiple Tests. *Scientific Programming*. (2020, Q4, IF 1.025)

[A.7] **HUYNH, H.M.**, NGUYEN, L.T.T., VO, B., YUN, U., OPLATKOVÁ, Z.K. and HONG, T.-P. Efficient algorithms for mining clickstream patterns using pseudo-IDLists. *Future Generation Computer Systems*. (2020, D1, IF 7.5)

[A.8] **HUYNH, H.M.**, NGUYEN, L.T.T., VO, B., NGUYEN, A. and TSENG, V.S. Efficient methods for mining weighted clickstream patterns. *Expert Systems with Applications*. (2020, Q1, IF 8.5)

[A.9] HUYNH, B., TRINH, C., **HUYNH, H.**, VAN, T.T., VO, B. and SNASEL, V. An efficient approach for mining sequential patterns using multiple threads on very large databases. *Engineering Applications of Artificial Intelligence*. (2018, Q1, IF 8.0)

Conferences

[A.10] PHAM, N.N., OPLATKOVA, Z.K., **HUYNH, H.M.** and VO, B. Mining Top-K High Utility Itemsets Using Bio-Inspired Algorithms with a Diversity within Population Framework. In : *International Conference on Computing and Communication Technologies (RIVF)*. 2022.

[A.11] PHAM, N.N., OPLATKOVÁ, Z.K., **HUYNH, M.H.** and VO, B. Mining Top-K High Utility Itemset Using Bio-Inspired Algorithms. In : *The sixth Workshop on Complexity in Engineering COMPENG*. 2022.

[A.12] **HUYNH, H.M.**, PHAM, N.N., OPLATKOVÁ, Z.K., NGUYEN, L.T.T. and VO, B. Sequential Pattern Mining Using IDLists. In : *International Conference on Computational Collective Intelligence*. 2020.

[A.13] **HUYNH, H.M.**, NGUYEN, L.T.T., VO, B., OPLATKOVA, Z.K. and HONG, T.-P. Mining clickstream patterns using IDLists. In : *IEEE International Conference on Systems, Man and Cybernetics*. 2019.

LIST OF SYMBOLS, ACRONYMS, AND ABBREVIATIONS

CPM	Clickstream pattern mining
SPM	Sequential pattern mining
FIM	Frequent itemset mining
CUP	Clickstream pattern mining Using Pseudo-IDList
DUB	Dynamic intersection Upper Bound constraint
SUI	Sequential pattern mining Using Indices
s-tailed	Singleton-tailed
ns-tailed	Non-singleton-tailed
N/A	Not applicable

LIST OF TABLES

LIST OF FIGURES

CURRICULUM VITAE

Personal Information:	
Name	Huy Minh Huynh
Date of Birth	1st January 1988
Nationality	Vietnamese
Address	Zlín
Contact	Phone: +420 608 266 057 Email: huynh@utb.cz
Education:	
2019 – present	Doctoral Student Tomas Bata University in Zlín (UTB), Czechia Faculty of Applied Informatics Degree programme: Information Technologies
2011 – 2014	Master of Engineering Ho Chi Minh City University of Technology (HCMUT), Vietnam Faculty of Computer Science and Engineering Degree programme: Computer Science
2006 – 2010	Bachelor of Information Technology Ton Duc Thang University (TDTU), Vietnam Faculty of Information Technology Degree programme: Information Technology

Working Experience:	
2024 – Present	Software Engineer at FNZ, Brno, Czechia • Backend C# Engineer
2023 – 2024	Software Engineer at AT&T, Brno, Czechia • Backend Python Developer • Prompt Engineering • Researching and Applying Large Language Models and Machine Learning
2014 – 2018	Software Developer at Ton Duc Thang University, Vietnam • E-Learning Management Systems and Online Quiz Examination Systems • .Net Web Applications • Open Journal Systems
2013	Visiting Lecturer at Ton Duc Thang University, Vietnam • Computer Programming Fundamentals
2010 – 2015	Teaching Assistant at Ton Duc Thang University, Vietnam • Computer Programming Fundamentals • Advanced Programming • Data Structures and Algorithms • Object Oriented Programming • Artificial Intelligence
2011	Teaching Assistant at Ho Chi Minh City University of Technology (HUTECH), Vietnam • Object Oriented Programming
Language:	
Vietnamese	Native Speaker
English	TOEIC (Score: 940)
Professional Achievement:	
Best Runner-Up Paper Award	The 10th International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2013

Efektivní metody pro dolování clickstream vzorů

Efficient Methods for Mining Clickstream Patterns

Doctoral Thesis Summary

Published by: Tomas Bata University in Zlín,

nám. T. G. Masaryka 5555, 760 01 Zlín.

Edition: published electronically

Typesetting by: Huy Minh Huynh, Ph.D.

This publication has not undergone any proofreading or editorial review.

First Edition

Publication year: 2025

ISBN 978-80-7678-357-7