

Mining Cost-Effective Patterns in Event Logs

Philippe Fournier-Viger^{a,*}, Jiaxuan Li^b, Jerry Chun-Wei Lin^c, Tin Truong Chi^d, R. Uday Kiran^e

^a*School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong, China, 518055*

^b*School of Computer Sciences and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong, China, 518055*

^c*Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway, 5020*

^d*Division of Computational Mathematics and Engineering, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam, 700000*

^e*National Institute of Informatics, Tokyo, Japan, 101-8430*

Abstract

High Utility Pattern Mining is a popular task for analyzing data. It consists of discovering patterns having a high importance in databases. A popular application of high utility pattern mining is to identify high utility (profitable) patterns in customer transaction data. Though such analysis can be useful to understand data, it does not consider the cost (e.g. effort, resources, money or time) required for obtaining the utility (benefits). In this paper, we argue that to discover interesting patterns in event sequences, it is useful to consider both a utility model and a cost model. For example, to identify cost-effective ways of treating patients from medical pathways data, it is desirable to consider not only the ability of treatments to inhibit symptoms or cure a disease (utility) but also the resources consumed and the time spent (cost) to provide these treatments. Based on this perspective, this paper defines a novel task of discovering Cost-Effective Event Sequences in event logs. In this task, cost is modeled as numeric values, while utility is represented either as binary or numeric values. Measures are proposed to evaluate the trade-off and correlation between cost and utility of patterns to identify cost-effective patterns (patterns having a low cost but providing a high utility). Three efficient algorithms called CEPB, corCEPB and CEPN are designed to extract these patterns. They rely on a tight lower-bound on the cost and a memory buffering technique to find patterns efficiently. Experiments show that the proposed algorithms achieve high efficiency, that proposed optimizations improve efficiency, and that insightful cost-effective patterns are found in real-life e-learning data.

Keywords: Event logs, sequences, pattern mining, sequential patterns, cost-effective patterns, utility, cost

1. Introduction

Discovering patterns in symbolic data is an important research area in data mining. Early studies in this area have mainly focused on discovering frequent patterns. For instance, in frequent itemset mining [1, 2], the goal is to identify sets of items that appear at least a minimum number of times in a transaction database. For numerous applications, frequent patterns reveal important information that can help to understand the data or take decisions. For instance, discovering frequent itemsets in medical data can reveal that some symptoms frequently appear together, which provide useful information for disease diagnosis. However, a key limitation of frequent itemset mining is that the time dimension is ignored. To address this limitation, a more general problem called sequential pattern mining was proposed [3, 4, 5], which consists of identifying subsequences that appear frequently in a sequence database. A sequence is an ordered list of itemsets and can be used to represent various types of data such as protein sequences, text, click streams and event logs. Many studies have been devoted to SPM. The first SPM algorithms are AprioriAll and GSP [4, 5]. These latter apply a breadth-first search to count the support of sequences in a database and output all frequent sequences. Recently, more efficient sequential pattern mining algorithms were designed such as SPADE [6],

*Corresponding author

Email addresses: philfv8@yahoo.com (Philippe Fournier-Viger), jiaxuanliniki@gmail.com (Jiaxuan Li), jerrylin@ieee.org (Jerry Chun-Wei Lin), truongchitin@tdtu.edu.vn (Tin Truong Chi), uday_rage@tkl.iis.u-tokyo.ac.jp (R. Uday Kiran)

FreeSpan [7], PrefixSpan [8], CM-SPADE [9], VMSP [10] and FCloSM [11]. These latter adopt various strategies such as using a vertical database representation or a pattern-growth approach to find frequent sequences efficiently. To meet requirements of various domains, several sequential pattern mining extensions have been proposed, which allow to take into account constraints and more complex data types [3]. One of the most popular SPM extensions in recent years is High Utility Sequential Pattern Mining (HUSPM) [12, 13, 14, 15]. It consists of finding sequences having a high utility (e.g. sequences of purchases that yield a high profit). HUSPM generalizes SPM and is a much more difficult problem than SPM because the utility measure is not anti-monotonic nor monotonic. That is, the utility of a sequence may be equal to, greater or smaller than those of its supersequences or subsequences. For this reason, HUSPM cannot be performed using traditional SPM techniques, which requires using anti-monotonic or monotonic measures to reduce the search space [12]. HUSPM allows to find important patterns in data, where importance is assessed using a numerical utility measure. For example, in customer transaction analysis, the utility measure can model the profit yield by sequences of purchases to identify profitable patterns. Another example is website click-stream analysis, where the utility can model the time spent to find sequences of webpages where people spend a lot of time. Identifying such patterns can be useful for decision-making [12, 13, 14].

Albeit HUSPM is an emerging research problem with several applications, a major limitation is that utility is often insufficient to truly evaluate the usefulness of a pattern. In fact, traditional HUSPM algorithms assess the utility or benefits that each pattern provides, but ignores the resources, effort, time or cost required to apply these patterns. This problem is illustrated with an example. Consider medical pathway data indicating the various treatments received by patients of a hospital. Applying HUSPM on this data allows to discover high utility patterns, where the utility can represent whether patients are cured or not after receiving treatments. For example a pattern $\langle treatmentA, treatmentB, treatmentC, cured \rangle$ may be discovered by traditional HUSPM algorithms indicating that many people who have received these treatments are cured, where *cured* is a high utility item. Although such patterns may seem useful, a major problem is that HUSPM ignores the *cost* for applying these patterns, that is the money, time or resources spent to cure each patient using these treatments. As a result, a HUSPM algorithm may find many patterns that have a high utility but have a very high cost, which is undesirable. And similarly, a HUSPM algorithm would miss all patterns that do not have a very high utility but are still useful because they provide a good trade-off between cost and utility.

A second example of this issue can be found in the domain of e-learning. Consider the analysis of sequences of learning activities performed by learners in an e-learning environment, to identify sequences of learning activities that help obtain high scores. A HUSPM algorithm could find many sequences of activities leading to high scores (scores would be modeled as the utility values of items). But HUSPM algorithms would ignore the time spent (cost) on these activities to achieve these scores. Thus, many patterns could be found having a high utility but also a very high cost (requiring to spend a lot of time), which may represent ineffective ways of studying. And patterns having a better trade-off between cost and utility may be missed.

To address this limitation, this paper proposes to find patterns by not only considering a utility model indicating the benefits provided by patterns, but also a cost model to assess the resources used or time spent to apply the patterns. Combining the concept of cost and utility is desirable but challenging. A reason is that there are many ways of measuring utility and cost. Cost can for example be measured in terms of time or money, while utility may represent time, profit, evaluation scores or failure/success. Because utility and cost can be measured using different units, it is not possible to simply combine the concepts of utility and cost by subtracting the cost from the utility, and then to apply a traditional HUSPM algorithm. Another problem of this approach is that it would not allow to evaluate how good the trade-off is between cost and utility for each pattern, and how strong the relationship is between resources spent and utility. For applications such as analyzing medical pathways, it is generally desirable to find patterns that not only have a high utility but also offer an excellent trade-off between utility and cost. Thus, cost and utility must be modeled separately and a tailored solution needs to be designed.

This article addresses this challenge by defining the novel problem of extracting Cost-Effective Patterns (CEP). The aim is to find patterns that provide a good trade-off between cost and utility from sequences containing utility and cost information. The main contributions of this study are threefold.

- The task of discovering cost-effective patterns in sequences is defined. In particular, three variations of the problem are defined to address the needs of different applications. In the first problem, information about the utility is encoded as a binary label for each sequence. The utility represents a desirable or undesirable outcome

(e.g. passing or failing an exam, being cured or not after receiving some medical treatments). In the second problem, utility is encoded as a positive number (e.g. a score obtained after completing a course). In the third problem, the utility is a binary value and it is assumed that only records are available for the positive class. Properties of the three proposed problems are studied. Moreover, statistical measures are designed to assess the correlation between utility and cost for these three problems.

- Three pattern-growth algorithms are presented to find cost-effective patterns for the three problem variations. The algorithms are named CEPB, corCEPB and CEPN, respectively. To efficiently find patterns, it is necessary to design a strategy to avoid exploring all possible patterns. However, a challenge is that the average cost measure cannot be used to reduce the search space because it is neither anti-monotonic nor monotonic. To address this issue and prune the search space efficiently, two tight lower-bound on the average cost of patterns are proposed named ASC and AMSC.
- Moreover, to reduce memory usage and speed up the algorithms, a technique called Projected Database Buffer (PDB) is integrated into the designed algorithms.

Several experiments have been performed to assess the performance of the proposed algorithms on various benchmark datasets. Results indicate that the proposed optimizations and the pruning strategy based on the lower-bound decrease runtimes of the proposed algorithms by up to 10 times. Moreover, an analysis of patterns found in real-life e-learning data shows that insightful patterns are found using the proposed algorithms.

The following sections are organized as follows. Section 2 reviews related work. Section 3 defines the proposed problems of discovering cost-effective patterns. Section 4 describes the proposed algorithms. Section 5 presents the experimental evaluation. Finally, Section 6 draws the conclusion.

2. Related Work

This section is divided into five subsections. The first subsection introduces pattern mining and relevant concepts. The second and third subsections discuss frequent and high utility sequential pattern mining, respectively. Finally, the fourth and fifth subsections discuss early work toward the consideration of cost in pattern mining, and related work on emerging pattern mining.

2.1. Pattern mining as a subfield of data mining

In general, data mining techniques are either used to extract patterns or models from data, which can be used to make sense of the data [16, 17]. Data mining techniques can be generally categorized as predictive or descriptive. Predictive techniques are typically used to predict the future or perform tasks such as classification. Several predictive techniques such as neural networks are black box models, as they are designed to provide good prediction/classification results rather than being interpretable. On the other hand, descriptive techniques are used to extract interesting or meaningful patterns from data to explain the data. This includes clustering and pattern mining techniques. Descriptive techniques such as pattern mining are glass-box techniques, as they extract patterns that are interpretable by humans and can support decision-making.

Another useful categorization of data mining techniques is that of supervised versus unsupervised methods. Supervised methods require labeled training data to build a model. Data is often represented as database records having a target attribute providing class labels. For example, for the problem of classification, training consists of discovering a model (function) to map attribute values to class labels to discover hidden relationships between attributes and class labels [18, 19, 20]. Several supervised methods have been proposed for classification as well as clustering [21, 22, 23]. On the other hand, unsupervised techniques assume that the data is unlabeled. Generally, pattern mining techniques are unsupervised as the goal is to extract hidden knowledge from data. But for some pattern mining tasks such as mining association rules to build classifiers [24], and to discover patterns that are different for two classes (emerging or contrast patterns) [25], class labels are considered for each database record.

2.2. Frequent sequential pattern mining

In the field of pattern mining, techniques have been proposed to extract various types of patterns from data such as itemsets [2, 26, 27], episodes [28, 29], subgraphs [30, 31], and sequences (sequential patterns) [3, 4, 5]. Generally, to find interesting patterns in data, various interestingness measures have been proposed. In early studies on pattern mining, the frequency (support) measure was mainly used to extract frequent patterns [1, 2].

In frequent sequential pattern mining (SPM) [3, 4, 5], the goal is to extract all the subsequences that appear in at least *minsup* sequences of a database, where *minsup* is a threshold set by users. For example, consider the database of Table 1, which contains four sequences, representing customer data. The first sequence indicates that an item *b* was purchased by a customer, followed by the purchase of item *c*, then *f*, and finally *g*. If *minsup* is set to 2, several frequent sequential patterns are found in this database, including the pattern $\langle a, f \rangle$ (meaning that *a* is followed by *f*), which has a support (occurrence frequency) of 3 (since it appears in the first three sequences).

Table 1: A sequence database containing four sequences (considered in traditional SPM)

Sequences
$\langle (a), (c), (f), (g), (e) \rangle$
$\langle (a), (c), (b), (f) \rangle$
$\langle (a), (b), (f), (e) \rangle$
$\langle (b), (f) \rangle$

Numerous techniques were proposed for discovering sequential patterns in databases. The first SPM algorithms are AprioriAll and GSP [4, 5]. To find all sequential patterns, GSP first scans the database to calculate the support of each single item. Then, GSP combines these patterns to generate candidate patterns containing two items. Thereafter, GSP scans the database again to calculate the support of these candidate patterns and keep only those that are frequent sequential patterns. Then, GSP repeats this process to find sequential patterns containing three items, and so on until all frequent patterns have been found. To avoid exploring all possible patterns, GSP applies a pruning property that states that the support of a sequence cannot be greater than the support of its subsequences, or in other words that the support measure is anti-monotonic. Although GSP is a correct and complete algorithm, it suffers from the drawback that it can generate numerous candidates that do not exist in a database, and scans a database numerous times.

To address the drawback of time-consuming database scans, several algorithms adopting a vertical database representation have been proposed such as SPADE [6], CM-SPADE [9], VMSP [10] and FCloSM [11]. These algorithms first scan the database to calculate the support of single items. Then, they create a vertical structure for each frequent pattern, and search for patterns using a depth-first search. These algorithms combine small patterns to generate larger patterns. The vertical structure of a pattern containing more than one item is obtained by intersecting the vertical structures of some of its sub-patterns. The vertical structure of a pattern is useful because it allows to calculate its support without scanning the database [6]. Although these vertical algorithms reduce the number of database scans, they can still generate candidate patterns that do not exist in the database, which is time-consuming.

To avoid this drawback, several pattern-growth algorithms have been proposed such as FreeSpan [7], PrefixSpan [8], and MaxSP [32]. These algorithms scan the database to count the support of single items. Then, they create a projected database for each frequent items, and scan that projected database to count the support of items and find larger patterns extending the current pattern. This process is repeated recursively using a depth-first search to find all patterns. The advantage of the pattern-growth approach is that only patterns that exist in the database are considered. As sequential pattern mining is an active research topic, many other algorithms and extensions have been proposed in recent years.

2.3. High utility sequential pattern mining

Though frequent pattern mining has many applications, frequent patterns are often uninteresting to users. For instance, a pattern $\{bread, egg\}$ may be frequent in a customer transaction database but may yield a low profit as these items are typically inexpensive. On the other hand, some rare patterns may be considered as important because they yield a high profit. Because users may be more interested in finding patterns that have a high profit or importance rather than mining frequent patterns, the problem of frequent itemset mining was generalized as high utility pattern mining [33, 34]. The goal of high utility pattern mining is to identify patterns having a utility that is no less than a

minimum utility threshold called *minutil*, where the utility is a numerical measure representing the relative importance of patterns (e.g. the profit generated).

Several high utility pattern mining problems have been proposed extending corresponding frequent pattern mining problems [26, 27, 33, 35, 36, 37]. The extension of sequential pattern mining that considers a utility measure is called high utility sequential pattern mining (HUSPM) [33]. The input of HUSPM is a minimum utility threshold *minutil* and a database of quantitative sequences, where each sequence is an ordered list of transactions. A transaction is a set of items, each annotated with some internal utility value (e.g. a purchase quantity). Moreover, each item has a weight called the external utility, indicating its relative importance (e.g. unit profit). The output of HUSPM is the set of all high utility sequences that have a utility (importance) that is no less than *minutil*. The concept of utility can be used to model data in various domains. For instance, HUSPM can be applied to mine sequences of purchases made by customers that yield a high profit from sequences of customer transactions. Table 2 shows an example quantitative sequence database containing four sequences. The third sequence indicates that a customer purchased three units of item *a*, followed by three units of item *b*, then two units of item *b*, and finally one unit of item *c*.

Table 2: A quantitative sequence database containing four sequences (the database type considered in HUSPM).

Quantitative sequences with purchase quantities (internal utility)
$\langle (a, 1), (e, 3) \rangle$
$\langle (a, 6), (c, 7), (b, 8), (d, 9) \rangle$
$\langle (a, 3), (b, 3), (b, 2), (c, 1) \rangle$
$\langle (b, 3), (c, 1) \rangle$
Unit profits (external utility)
$a = 5\$, b = 1\$, c = 2\$, d = 1\$$

A major challenge in HUSPM is that the utility measure is not anti-monotonic. In other words, the utility of a sequence may be higher, lower or equal to that of its subsequences. Thus, it is not possible to simply apply traditional SPM algorithms for HUSPM. The first HUSPM algorithms are US and UL [33]. US adopts a candidate generation approach similar to GSP, while UL adopts a pattern growth approach similar to PrefixSpan. To efficiently reduce the search space, these algorithms rely on a novel measure called sequence-weighted utility (SWU) that is an upper-bound on the utility of patterns and their supersequences, and is anti-monotonic. Then, several more efficient algorithms have been proposed. PHUS [34] adopts a pattern-growth approach with an upper-bound that is tighter than the SWU to reduce the search space and improve the mining performance. Other recent algorithms are USpan [12], HupsExt [38], and HUS-Span [39].

Although HUSPM is useful in many domains such as market basket analysis, only considering the utility to evaluate patterns is insufficient for several other domains where not only the utility must be considered but also the cost to determine if a pattern is desirable or interesting. Hence, this paper proposes to utilize both a cost and a utility model to identify useful patterns in event sequences.

2.4. Integrating the concept of cost in pattern mining

Few work have considered cost in pattern mining. The closest work to the one presented in this paper is in the field of process mining, where cost was considered to perform behavior analysis. The main goal of process mining is to extract models or patterns from event logs that characterize a business process [40]. An event log is a list of events with timestamps, ordered by time. Techniques for process mining have been applied for many tasks such as analyzing education, business and healthcare event logs [41, 42, 43], using both supervised and unsupervised methods [44, 45]. To analyze medical pathway data, it was proposed to extract a graph [46], where nodes and edges, respectively represent events and their temporal relationships. Such graph was used to formulate guidelines for the medical treatments of inpatients with the sepsis condition. Though extracting this type of graphs can be useful, it does not consider the cost of treatments, and temporal relationships are limited to consecutive events. Moreover, the proposed graph structure can be very large and complex for large event logs and thus be difficult to interpret. An algorithm named Twinkle [47] was proposed for analyzing hospital event-logs, where each event is annotated with a cost value. The output is rules indicating that a set of events occurred before another set of events, and having a small cost (e.g. in terms of money for medical treatments). Such rules give insights about how to reduce medical costs.

Twinkle adopts a rule growth approach which starts by finding rules between pairs of events and then recursively grows these rules to find larger rules. To use Twinkle, the user must set multiple parameters, specified in terms of time, confidence, cost and rule length. An important limitation of Twinkle is that although it considers the cost of patterns, it ignores the utility. Thus, it can find patterns that have a low-cost but a low utility. For example, it could find a pattern representing cheap medical treatments but that provide a very low probability of being cured. In this paper, we consider that both utility and cost should be assessed to find cost-effective patterns (patterns providing a good cost/utility trade-off).

2.5. Emerging pattern mining

Emerging pattern mining is another research area that is related to the work presented in this paper. It consists of discovering patterns that appear significantly more often in records that belong to a class than to those of another class. For example, Poon et al. [48] proposed to discover emerging sequential patterns in educational data. The data is an event log from a statistic course, where each event is an access to a simulation, video or quiz, and has a timestamp. Moreover, the event log of each learner is annotated with a class label indicating if his quiz score result is below or above average. Such a sequence database with class labels is presented in Table 3. Then, patterns were extracted that discriminate between the two groups to recommend better ways of using e-learning resources.

A major limitation of this work is that it does not consider the cost (e.g. the time spent by students for each event to obtain high scores). Moreover, a score is defined as a binary variable rather than a numeric variable, which results in information loss. It is thus desirable to design an algorithm that can identify patterns having a low cost and high utility (e.g. high scores), and can consider not only binary classes but also numeric values if required.

Table 3: A sequence database with binary labels (a database type considered in emerging SPM)

Sequences	Class Labels
$\langle(a), (c), (f), (g), (e)\rangle$	above average
$\langle(a), (c), (b), (f)\rangle$	above average
$\langle(a), (b), (f), (e)\rangle$	below average
$\langle(b), (f)\rangle$	below average

The next section addresses the aforementioned limitations by proposing a novel problem of discovering cost-effective patterns by combining both the concepts of utility and cost. The cost is modeled by associating a numeric value to each event, while the utility is either defined by associating a binary class label (e.g. *died* or *cured*) or a numeric value (e.g. an exam score) to each sequence. Moreover, a concept of correlation or trade-off is introduced to assess the relationship between the utility and cost of each pattern.

3. Problem Definition

This section proposes the novel problem of discovering Cost-Effective Patterns (CEP). Three cases (variations of this problem) are presented to address the needs of different applications. The type of database that is considered is Sequential Event Logs (SEL) where events are annotated with cost values and each sequence has utility information either encoded as a binary or a numeric value. This section first introduces the type of data and important concepts. Then the proposed problem is defined, and key differences with previous work are highlighted.

Definition 1. (Sequential Event Log) A sequence S_s is a list of m events denoted as $S_s = \langle\{e_1[c_1], e_2[c_2], \dots, e_m[c_m]\} | Utility\rangle$. Each event e_i ($1 \leq i \leq m$) is annotated with a positive number c_i indicating the cost of the event. The cost is a measure of resources spent for an event such as an amount of time or money. An event e_i is said to have appeared before another event e_j of a sequence if $i < j$. Furthermore, each sequence has a utility value $Utility$. A sequential event log (SEL) consists of n sequences, denoted as $SEL = \{S_1, S_2, \dots, S_n\}$. The s -th sequence of a SEL, denoted as S_s , is said to have the sequence identifier s . Two types of SELs are considered. A *numeric SEL* contains sequences where utility values are positive numbers, while a *binary SEL* contains sequences where utility is a binary label ($-$ or $+$, denoting a negative or positive outcome).

Table 4: A binary SEL

Sid	Sequence (event[cost])	Class
1	$\langle (a[2]), (b[4]), (c[9]), (d[2]) \rangle$	+
2	$\langle (b[1]), (d[12]), (c[10]), (e[1]) \rangle$	-
3	$\langle (a[5]), (e[4]), (b[8]) \rangle$	+
4	$\langle (a[3]), (b[5]), (d[1]) \rangle$	-
5	$\langle (b[3]), (e[4]), (c[2]) \rangle$	+

Table 5: A numeric SEL

Sid	Sequence (event[cost])	Utility
1	$\langle (a[20]), (b[40]), (c[50]), (d[20]) \rangle$	80
2	$\langle (b[25]), (d[12]), (c[30]), (e[25]) \rangle$	60
3	$\langle (a[25]), (e[14]), (b[30]) \rangle$	50
4	$\langle (a[40]), (b[16]), (d[40]) \rangle$	40
5	$\langle (b[20]), (e[24]), (c[20]) \rangle$	70

For instance, consider the binary SEL database depicted in Table 4, which contains five sequences. The second sequence contains four events. The event b first occurred and has a cost of 1. Then, it was followed by d with a cost of 12, then by c with a cost of 10, and finally by e with a cost of 1. That sequence has a negative class label (denoted as -), indicating that it has lead to a negative outcome. Other sequences follow the same format. This type of database can be found in many domains. For example, it can model sequences of treatments received by patients where positive and negative class labels indicate whether a patient has cured or died, respectively. Another example is sequences of learning activities done by students where cost is the time spent and class labels indicate whether a student has then passed or failed an exam.

An example of numeric SEL is shown in Table 5. This database contains five sequences encoded using the same format as in the previous example, but where the utility is modeled as a numeric value. For example, the first sequence has a utility of 80. A numeric SEL can be used to encode data from various applications. For example, a sequence of learning activities done by students may be annotated with a numeric utility value indicating his final grade.

To discover interesting relationships between events in a SEL database, we consider a type of patterns that is a sequence of events.

Definition 2. (Pattern) A time-ordered list of events is called a sequence of events or *pattern*. Consider two patterns $p = \langle e_1, e_2, \dots, e_o \rangle$ and $v = \langle r_1, r_2, \dots, r_q \rangle$. The pattern v is said to extend p (to be an extension of p) if $r_{y_1} = e_1, r_{y_2} = e_2, \dots, r_{y_o} = e_o$ for some integers $1 \leq y_1 \leq y_2 \leq \dots \leq y_o \leq q$.

In the following, the extension of a pattern $p = \langle e_1, e_2, \dots, e_o \rangle$ with an item z to obtain a sequence $\langle e_1, e_2, \dots, e_o, z \rangle$ will be denoted as $v \cup z$.

Let there be a sequence $S_s = \langle \{e'_1[c_1], e'_2[c_2], \dots, e'_m[c_m]\} | Utility \rangle$. The sequence S_s is said to contain p (denoted as $p \subseteq S_s$) if $e'_{a_1} = e_1, e'_{a_2} = e_2, \dots, e'_{a_o} = e_o$ for some integers $1 \leq a_1 \leq a_2 \leq \dots \leq a_o \leq m$. If that condition is met, it is equivalently said that p occurs or appears in S_s . The first occurrence of p in a sequence S_s is the smallest set of integers a_1, a_2, \dots, a_o such that $p \subseteq S_s$. The sequence of events $\langle e'_{a_1}, e'_{a_2}, \dots, e'_{a_o} \rangle$ corresponding to that *first occurrence* is denoted as $first(p, S_s)$.

To select interesting patterns in sequential event logs, the support and (average) cost measures are used.

Definition 3. (Support measure) Let there be a pattern p . The **support** of p in a sequential event log SEL is the number of sequences where p appears. Formally, $sup(p) = |\{S_s | p \subseteq S_s \wedge S_s \in SEL\}|$.

The reason for using the support measure is the same as in frequent pattern mining, that is to eliminate infrequent patterns that may represent noise [49]. The cost measure is defined as follows.

Definition 4. (Cost measure) Let there be a sequence S_s and a pattern p . The cost of p in S_s is defined as the sum of the cost values of the first occurrence of p in S_s , that is $c(p, S_s) = \sum_{e_i \in first(p, S_s)} c(e_i, S_s)$ if $p \subseteq S_s$ and otherwise 0. Given a SEL, the cost of p is the sum of its cost in all sequences where it appears. Formally, it is defined as $c(p) = \sum_{p \subseteq S_s \wedge S_s \in SEL} c(p, S_s)$.

For instance, consider the binary SEL shown in Table 4. In the first sequence, the pattern $\langle a, b \rangle$ has a cost of $c(\langle a, b \rangle, S_1) = 2 + 4 = 6$. If cost values are represented in dollars, it would mean that 6 dollars were spent to apply this

pattern in S_1 . The cost of that pattern in the whole database is calculated as: $c(\langle a, b \rangle) = c(\langle a, b \rangle, S_1) + c(\langle a, b \rangle, S_3) + c(\langle a, b \rangle, S_4) = (2 + 4) + (5 + 8) + (3 + 5) = 27$.

In the above definition, the cost is defined based on the first occurrence of a pattern in each sequence. Without loss of generality, this definition can be reversed to consider the last occurrence instead of the first one. The reason for choosing either the first or last occurrence for computing the cost rather than considering all occurrences is that some cases are problematic if multiple occurrences are considered. For example, if many occurrences of a pattern share events, calculating the cost of all occurrences would not only be more computationally expensive than calculating the cost of only the first (last) occurrence but would result in overestimating the pattern's cost (due to adding multiple times the cost values of events shared by multiple occurrences).

The cost is a useful measure to evaluate the total amount of resources spent to apply a pattern in a SEL (e.g. total amount of money or time). However, because the cost of a pattern may not be the same in different sequences, it is useful to evaluate a pattern using the average for all sequences where it appears. The average cost is defined as follows.

Definition 5. (Average cost measure) Consider a pattern p . Its average cost in a SEL is the average of its costs in all sequences where it appears. Formally, it is denoted and defined as: $ac(p) = \frac{\sum_{p \subseteq S, S \in SEL} c(p, S)}{|sup(p)|}$.

For instance, consider the binary SEL shown in Table 4. The cost of pattern $\langle a, b \rangle$ in the sequences S_1 , S_2 and S_3 where it appears is 6, 13 and 8, respectively. Thus its average cost is $ac(\langle a, b \rangle) = \frac{27}{3} = 9$. If cost values represent an amount of time in hours, this pattern indicates that the average cost for applying the pattern $\langle a, b \rangle$ is 9 hours.

Using the average cost measure is more meaningful than using the cost for several applications such as e-learning. For example, consider that the above database is sequences of activities or events, representing the activities done by different learners, where the cost of an activity is the time spent, and the utility is to pass or fail an exam. The cost of the pattern $\langle a, b \rangle$ indicates that all students have totally spent 27 hours studying activity a and then b . But a problem with the cost measure is that it does not inform us about the resources typically spent by each student. This is a problem because each user may apply the same pattern with different cost values, i.e. students may spend different amount of time on these activities. Using the average cost measure addresses this issue as it lets the user know about the average amount of resources typically used for using each pattern. In this example, the average cost of $\langle a, b \rangle$ indicates that a student typically spent 9 hours on that pattern. Knowing this information is useful for users, as a user may want to know the amount of resources required to apply a pattern, and may not want to use a pattern that requires spending too much resources. Note that it would be possible to also compute other measures like the standard deviation of the cost. However, the average cost is especially interesting as we can define a lower bound that can be used for reducing the search space to improve the performance of our proposed algorithms.

Lemma 1 (anti-monotonicity of the support). Let there be two patterns p and q such that q extends p (or generally, p is included in q). Then, $sup(p) \geq sup(q)$ [4].

Lemma 2 (lack of monotonicity or anti-monotonicity of the average cost). Let there be two patterns p and q such that q extends p (p is included in q). The average cost of p may be smaller, equal or greater than that of q .

Proof 1. The above lemma is proved using an example. Consider the binary SEL of Table 4. The average cost of pattern $\langle d \rangle$ is $ac(\langle d \rangle) = c(\langle d \rangle, S_1) + c(\langle d \rangle, S_2) + c(\langle d \rangle, S_4) / sup(\langle d \rangle) = (2 + 12 + 1) / 3 = 5$, while $ac(\langle a \rangle) = c(\langle a \rangle, S_1) + c(\langle a \rangle, S_3) + c(\langle a \rangle, S_4) / sup(\langle a \rangle) = (2 + 5 + 3) / 3 = 3.3$ and $ac(\langle a, d \rangle) = c(\langle a, d \rangle, S_1) + c(\langle a, d \rangle, S_4) / sup(\langle a, d \rangle) = \frac{(4+4)}{2} = 4$.

Based on the above lemma, it is clear that the average cost cannot be directly used to reduce the search space. To still mine patterns efficiently using this measure, Section 4 will propose two lower-bounds on the average-cost that can be used for search space reduction.

It is to be noted that in the field of frequent sequential pattern mining, two strategies were proposed to prune the search space using the average measure when a numeric value is associated to each item from sequences [50]. However, these strategies are not applicable for the average cost measure since they assume that an item must always have the same value in all sequence. But in the problem studied in this paper, an event (item) may have different cost values in different sequences (which is more general).

Table 6: Cost-effective patterns found in positive sequences (Case 1)

Pattern	Average cost	Pattern	Average cost
$\langle a \rangle$	3.5	$\langle c \rangle$	5.5
$\langle e \rangle$	4.0	$\langle b, c \rangle$	9.0
$\langle b \rangle$	5.0	$\langle a, b \rangle$	9.5

Having presented the type of patterns that will be found and the cost and support measures that will be used to evaluate patterns, the next subsections present the problem of mining cost-effective patterns for three cases, corresponding to different scenarios. Additional pattern evaluation measures will be introduced for the second and third cases.

3.1. Case 1: Mining Cost-Effective Patterns in Positive Sequences of a Binary SEL

The first case is to extract cost-effective patterns in a binary SEL where only sequences annotated with the positive class are used. The goal is to extract patterns having a low cost that lead to a desirable outcome. The problem is defined as follows:

Definition 6. (Problem definition - Case 1) Let there be a binary SEL and two user-specified thresholds named $maxcost$ and $minsup$. The problem of discovering cost-effective patterns in positive sequences consists of identifying each pattern q that has a low average cost and is frequent, that is $sup(q) \geq minsup$ and $ac(q) \leq maxcost$.

For instance, consider the binary SEL of Table 4 and that $minsup = 2$ and $maxcost = 20$. This SEL contains three positive sequences (S_1 , S_3 and S_5). The cost-effective patterns extracted from these positive sequences are shown in Table 6. These patterns can be viewed as different sequences of events that have lead to the desirable outcome. It can be observed that these patterns contain different events and have different average cost. The assumption of this problem definition is that by discovering patterns in positive sequences of a binary SEL, cost-effective patterns that are likely to yield a positive outcome can be found.

3.2. Case 2: Mining Cost-Effective Patterns in Negative and Positive Sequences of a Binary SEL

Case 1 considers mining cost-effective patterns in positive sequences. This can reveal patterns having a low cost but a high utility (leading to the desirable outcome). This can be useful when sequences with negative labels are unavailable. However, there is an important drawback of Case 1. It is that negative sequences are not considered. As a result, some patterns may be discovered that appear not only in positive sequences but also in negative sequences. Such patterns can be misleading because they may contain events that actually do not influence the utility (outcome) of the sequence.

To address this issue, a second case is defined where cost-effective patterns are mined from all sequences of a binary SEL (by considering both positive and negative sequences), and where the correlation between a pattern and its utility is evaluated. By evaluating this correlation, it is possible to find patterns that are cost-effective and correlated with a positive utility. The following paragraphs propose a correlation measure and then the problem of Case 2 is described.

Definition 7. (Positive and negative sequences containing a pattern) Let there be a binary SEL where each sequence is annotated with positive or negative class labels. For a pattern p , let D_p^+ and D_p^- denote the set of positive and negative sequences containing the pattern p , respectively.

For instance, consider the binary SEL shown in Table 4. The set of positive sequences of pattern $\langle c \rangle$ is $D_{\langle c \rangle}^+ = \{S_1, S_5\}$, while its set of negative sequences is $D_{\langle c \rangle}^- = \{S_2\}$.

Table 7: Cost-effective patterns found in negative and positive sequences (Case 2)

Pattern	Correlation	Average cost	Pattern	Correlation	Average cost
$\langle e \rangle$	1.0	3.0	$\langle b \rangle$	0.42	4.2
$\langle a, b \rangle$	0.24	9.0	$\langle a \rangle$	0.19	3.3
$\langle b, c \rangle$	-0.28	9.7	$\langle d \rangle$	-0.43	5.0
$\langle b, d \rangle$	-0.50	8.3	$\langle c \rangle$	-0.60	7.0

Definition 8. (Correlation of a pattern with binary utility) For a binary SEL, the correlation of a pattern p to the utility of sequences where it appears is calculated as: $cor(p) = \frac{ac(D_p^+) - ac(D_p^-)}{Std} \sqrt{\frac{sup(D_p^+)}{|D_p^+|} \frac{sup(D_p^-)}{|D_p^-|}}$ where $ac(D_p^+)$ and $ac(D_p^-)$ respectively denote the pattern p 's average cost in D_p^+ and D_p^- , Std is the standard deviation of p 's cost and $sup(D_p^+)$, $sup(D_p^-)$ are respectively the support of p in D_p^+ and D_p^- . Positive and negative cor values indicate a positive and negative correlation, respectively

For instance, consider the pattern $\langle c \rangle$ and the binary SEL of Table 4. It is found that $cor(\langle c \rangle) = \frac{(\frac{9+2}{2} - \frac{10}{1})}{Std(9,2,10)} \sqrt{\frac{2}{3} \times \frac{1}{3}} \approx -0.60$.

The above correlation measure named cor was defined by adapting the biserial correlation measure [51], which is generally used to assess the correlation between a binary and a numeric attribute in statistics. Values of the cor measure are in the $[-1, 1]$ interval. The term $ac(D_p^+) - ac(D_p^-)$ is used in the cor measure to find patterns that have a large difference in terms of average cost for positive and negative sequences. For example, in e-learning, a same pattern representing different ways of studying may be interesting if it has different cost (amount of time) for students who fail and succeed a course. That cost difference is divided by the standard deviation of the cost to avoid using absolute values in the equation. The term $\sqrt{\frac{sup(D_p^+)}{|D_p^+|} \frac{sup(D_p^-)}{|D_p^-|}}$ is used in the cor measure to find patterns that are different in terms of occurrence frequencies for the positive and negative classes. The cor measure is interpreted as follows. A negative cor value indicates that a pattern is correlated with a negative utility. A positive cor value indicates that a pattern is correlated with a positive utility. The greater positive (smaller negative) the cor measure is, the more a pattern is correlated with a positive (negative) utility. Consider the patterns $\langle a, b \rangle$ and $\langle a, c \rangle$ in the SEL of Table 4. It is found that $cor(\langle a, c \rangle) = -0.5$ and $cor(\langle a, b \rangle) = 0.8$. Thus, the former pattern is correlated with a negative outcome, while the latter pattern is correlated with a positive outcome. It is to be noted that the proposed algorithms do not use the cor measure for search space pruning and thus its calculation could be adapted for a specific application if needed. The problem definition of Case 2 is defined as follows.

Definition 9. (Problem definition - Case 2) Given a binary SEL and user-defined $minsup$ and $maxcost$ thresholds, the problem of Case 2 consists of extracting each pattern p such that $(sup(p) \geq minsup) \wedge (ac(p) \leq maxcost)$ and p has a high positive correlation.

In other words, the aim is to identify cost-effective patterns where cost is correlated with a positive utility (the desirable outcome) rather than a negative utility. For instance, consider the binary SEL of Table 4. If the parameters are set to $minsup = 3$ and $maxcost = 10$, eight patterns are discovered for Case 1, depicted in Table 7. Interestingly, it is found that four of those patterns are correlated with a negative utility, i.e. $\langle c \rangle$, $\langle b, d \rangle$, $\langle d \rangle$, and $\langle b, c \rangle$. This shows the importance of considering a correlation measure to identify patterns that are not only cost-effective but also likely leading to a positive outcome.

3.3. Case 3: Mining Cost-effective Patterns in a Numeric SEL

Case 1 (Section 3.1) and Case 2 (Section 3.2) considered that utility is encoded as binary values. This section presents the third case to handle SEL with numeric utility values. A numeric SEL can be used to model various types

Table 8: Cost-effective patterns found in a numeric SEL (Case 3)

Utility	Pattern	Trade-off
70	$\langle b, c \rangle$	0.88
65	$\langle b, e \rangle$	0.72
60	$\langle b, d \rangle$	0.85
56	$\langle a, b \rangle$	1.01

of data such as event logs of e-learning sessions where final exam scores are expressed as numbers. An example numeric SEL is shown in Table 5. To find cost-effective patterns in such SEL, the *cor* measure cannot be applied as it is only defined for binary utility values. Hence, the next paragraphs redefine the concept of pattern utility and present a novel measure called *trade-off* to evaluate the relationship between cost and numeric utility in a numeric SEL.

Definition 10. (Utility of a pattern) Let there be a numeric SEL and a pattern p . The *utility of p in the SEL* is the average of the utility of sequences containing p , i.e. $u(p) = \frac{\sum_{p \subseteq S_s \wedge S_s \in SEL} su(S_s)}{|sup(p)|}$ where $su(S_s)$ denotes the numeric utility of sequence S_s .

Definition 11. (Trade-off) For a pattern p , the *trade-off* of p is the ratio of its average cost to its average utility, i.e. $tf(p) = \sum_{p \subseteq S_s \wedge S_s \in SEL} (c(p, S_s) / su(S_s))$.

Trade-off values are in the $(0, \infty]$ interval. The trade-off value of a pattern indicates how efficient the pattern is. A pattern having a small trade-off is viewed as being cost-effective as it provides utility at a low cost (it requires a small amount of resources to obtain a desirable outcome). For example, in e-learning, a pattern with a small trade-off may indicate that studying some learning activities (events) typically requires a small amount of time (cost) and is correlated with obtaining high scores (utility). On the other hand, patterns having a larger trade-off may be viewed as not being cost-effective, or being less efficient. For example, consider the Table 5 and the pattern $\langle a, d \rangle$. Its trade-off is calculated as: $tf(\langle a, d \rangle) = ac(\langle a, d \rangle) / u(\langle a, d \rangle) = \frac{[c(\langle a, d \rangle, S_1) + c(\langle a, d \rangle, S_4)] / sup(\langle a, d \rangle)}{[su(S_1) + su(S_4)] / 2} = \frac{[(20+20) + (40+40)] / 2}{[(80+40)] / 2} = 1$. It is to be noted that the proposed algorithms do not use the trade-off measure for search space pruning. Thus, it could be easily adapted for the requirements of specific applications if needed. The problem definition of Case 3 is given next.

Definition 12. (Problem definition - Case 3) Consider a numeric SEL and two user-defined thresholds, *minsup* and *maxcost*. The goal of Case 3 is to identify each cost-effective pattern p such that $(sup(p) \geq minsup) \wedge (ac(p) \leq maxcost)$, and p has a low trade-off and high utility.

Case 3 of the proposed problem can be used to find cost-effective patterns that provide a good trade-off between cost and utility. After extracting patterns using the minimum support and maximum cost, patterns can be ranked by descending order of trade-off before being presented to the user.

For instance, consider the numeric SEL of Table 8, *minsup* = 2 and *maxcost* = 100. Four patterns are found, shown in Table 5. In that example, $\langle b, e \rangle$ is viewed as the most efficient pattern since it has the lowest trade-off.

3.4. Relationship between the proposed problem and previous work

The problem studied in this paper and its three cases are quite different from problems studied in previous papers. To illustrate these differences, Table 9 compares the type of data and patterns extracted for the problems of (1) frequent SPM, (2) HUSPM, (3) emerging SPM, (4) low-cost rule mining, and (5) the proposed problem. From this table, it can be seen that the type of data and patterns considered in this paper are quite different from previous work.

In SPM [3, 4, 5], patterns are found strictly based on their support. In HUSPM [12, 13, 14], patterns are found based on their utility, and events are annotated with numeric utility values. In emerging pattern mining, frequent patterns are found in database records having class labels. These class labels represent categories of records. In low-cost rule mining [47], the goal is to extract rules having a low-cost from sequences where events are annotated with cost values.

Differently, from previous work, the proposed problem considers both cost and utility, and the relationship (trade-off or correlation) between cost and utility, with the aim of finding low-cost patterns that can lead to a high utility (positive outcome). Moreover, utility is sequence annotations rather than event annotations because utility is viewed as an outcome. Binary utility can be viewed as similar to class labels used in emerging pattern mining. But the semantic is quite different. The former represents positive/negative outcomes, while the latter represent categories of records that may not have a negative or positive meaning. Moreover, in the proposed problem, utility can be numeric unlike in emerging pattern mining where symbolic labels are used. Besides, differently from HUSPM, events are annotated with cost rather than utility, and the average cost is considered.

Because of the many differences with previous work, existing algorithms and techniques for reducing the search space cannot be directly adapted to solve the proposed problem. Generally, designing efficient pattern mining algorithms require to use techniques tailored to the problem. Hence, the next section presents novel properties, appropriate data structures and efficient algorithms to solve the three cases of the proposed problem. Then, Section 5 presents an experimental evaluation to evaluate the algorithms' performance and case studies are described, which shows that interesting patterns are found in real data.

Table 9: Comparison of database and pattern types for the proposed problem and those of previous work

Problem	Event annotations?	Sequence annotations?	Pattern type
SPM	–	–	frequent sequences
HUSPM	Utility (numeric)	–	high utility sequences
Emerging SPM	–	Binary classes	emerging sequences
Low-cost rule mining	Cost (numeric)	–	low-cost rules
Proposed problem - Case 1,2	Cost (numeric)	Binary utility	cost-effective sequences
Proposed problem - Case 3	Cost (numeric)	Numeric utility	cost-effective sequences

4. Proposed Algorithms

This section presents algorithms to efficiently discover Cost-Effective Patterns (CEP) for the three cases introduced in the previous section. The algorithms rely on the basic search procedure of the PrefixSpan algorithm [8] to explore the search space of sequences in a database. This procedure starts by considering patterns containing single events and then recursively grows these patterns by appending items one at a time. To reduce the cost of scanning the database to calculate the measures, a projected database is created for each pattern. In the proposed algorithm, that procedure is adapted to consider the cost and utility, to use a lower-bound on the cost to reduce the search space, and to integrate a memory buffering optimization.

This section is organized as follows. Section 4.1 introduces a lower bound on the cost of patterns, called ASC, and discusses its properties. Then, Section 4.2 presents a lower bound named AMSC, which is tighter than the ASC. Then, Section 4.3 introduces the buffering method to reduce memory usage when creating projected databases. Finally, Section 4.4, 4.5 and 4.6 present the proposed algorithms for the three cases by putting all these ideas together. For each algorithm, pseudocode is given, a detailed example, as well as a discussion of the algorithm's complexity.

4.1. A Lower Bound on the Average Cost

The search space for discovering CEP in a SEL can be very large. If the longest sequence in a SEL has k events, there are up to $2^k - 1$ patterns to be considered. To design efficient CEP mining algorithms, it is thus necessary to avoid exploring all possible patterns to find the desired patterns. In frequent pattern mining, search space reduction is done using the anti-monotonicity property of the support measure. But as shown in Lemma 2, the average cost measure is neither monotonic nor anti-monotonic, and thus cannot be used to reduce the search space as in frequent pattern mining. As a solution, this section introduces a lower bound on the average cost that allows to reduce the search space, named *Average Supported Cost* (ASC), and a corresponding pruning property.

Definition 13. (Average Supported Cost). Consider a pattern p such that $\text{sup}(p) \geq \text{minsup}$. Let $C(p)$ be the set of costs of p in sequences where it appears, i.e. $C(p) = \{c(p, S_i) | p \subseteq S_i \in \text{SEL}\}$. Furthermore, let $c_i(p)$ denotes the i -th smallest cost of p in $C(p)$. The ASC is defined as: $\text{asc}(p) = \frac{1}{\text{sup}(p)} \sum_{i=1,2,\dots,\text{minsup}} c_i(p)$.

For example, consider Table 5, $minsup = 2$ and the pattern $p = \langle a \rangle$. The pattern p appears in the first, third and fourth sequences, and its set of costs is $C(p) = \{20, 25, 40\}$. Hence, $asc(\langle a \rangle) = \frac{1}{3} \times (20 + 25) = 15$. The average cost of $\langle a \rangle$ is $ac(\langle a \rangle) = (20 + 25 + 40)/3 \approx 28.3$

Property 1. (Underestimation of the average cost by the ASC measure) The *ASC* of a pattern p is smaller than or equal to its average cost, that is $asc(p) \leq ac(p)$. In other words, the *ASC* is a lower bound on the average cost.

Proof 2. Since $sup(p) \geq minsup$, $asc(p) = \frac{1}{sup(p)} \sum_{i=1,2,\dots,minsup} c_i(p) \leq ac(p) = \frac{1}{sup(p)} \sum_{i=1,2,\dots,sup(p)} c_i(p)$.

Property 2. (Monotonicity of the *ASC* measure) For any two sequential patterns $p_x \subset p_y$, the relationship $asc(p_x) \leq asc(p_y)$ holds.

Proof 3. Since $p_x \subseteq p_y$, it follows that $sup(p_x) \geq sup(p_y) \geq minsup$ and $c_i(p_x) \leq c_i(p_y)$. Thus, we can infer that $asc(p_x) = \frac{1}{sup(p_x)} \sum_{i=1,2,\dots,minsup} c_i(p_x) \leq asc(p_y) = \frac{1}{sup(p_y)} \sum_{i=1,2,\dots,minsup} c_i(p_y)$.

Property 3. (Pruning using the *ASC* lower bound) For a pattern p , if $asc(p) > maxcost$, then it follows that $ac(p) > maxcost$, and that p is not a CEP. Moreover, any super-pattern $p_y \supset p_x$ is also not a CEP.

Proof 4. This directly follows from Property 1 and 2.

4.2. A Tighter Lower-Bound on the Average Cost

The previous subsection has introduced a lower bound on the average cost that can be used for search space reduction. But can we find a tighter lower bound to more effectively reduce the search space? The answer is yes.

Definition 14. (Average Minimum Supported Cost) Let the *Average Minimum Supported Cost* (*AMSC*) of a pattern p be defined as $amsc(p) = \frac{1}{minsup} \sum_{i=1,2,\dots,minsup} c_i(p)$.

For example, consider Table 5 and that $minsup = 2$. Then, $amsc(\langle a \rangle) = \frac{1}{2} \times (20 + 25) = 22.5 < ac(\langle a \rangle) \approx 28.3$.

Property 4. (Underestimation of the average cost by the *AMSC* measure) The *AMSC* of a pattern p is smaller than or equal to its average cost, that is $amsc(p) \leq ac(p)$.

Proof 5. Let $N = sup(p)$ and $M = minsup$. Without loss of generality, assume that all the cost values in $C(p) = \{c_1, c_2, \dots, c_N\}$ are sorted in ascending order, i.e. $c_1 \leq c_2 \leq \dots \leq c_M \leq \dots \leq c_N$. Then, $\frac{1}{M} \sum_{i=1,2,\dots,M} c_i(p) \leq \frac{1}{N} \sum_{i=1,2,\dots,N} c_i(p)$, because cost values are in ascending order, and thus $(N-M) \sum_{i=1,2,\dots,M} c_i(p) \leq (N-M) \cdot M \cdot c_M(p) \leq M \cdot (N-M) \cdot c_{M+1}(p) \leq M \sum_{i=M+1,M+2,\dots,N} c_i(p)$.

Property 5. (Monotonicity of the *AMSC* measure) Let there be two patterns $p_x \subset p_y$ that are frequent ($sup(p_x) \geq minsup$ and $sup(p_y) \geq minsup$). The relationship $amsc(p_x) \leq amsc(p_y)$ holds.

Proof 6. Since $p_x \subset p_y$, it follows that $sup(p_x) \geq sup(p_y) \geq minsup$. Because $\forall p_x \subseteq p_y$, $c_i(p_x) \leq c_i(p_y)$ and all the cost values $c_i(p_x)$ and $c_i(p_y)$ are sorted in ascending order, $c_i(p_y) > c_{j_i}(p_x)$ where $\{j_1, j_2, \dots, j_{sup(p_y)}\} \subseteq \{1, 2, \dots, sup(p_x)\}$, and therefore $amsc(p_y) = \frac{1}{minsup} \sum_{i=1,2,\dots,minsup} c_i(p_y) \geq amsc(p_x) = \frac{1}{minsup} \sum_{i=1,2,\dots,minsup} c_{j_i}(p_x)$.

Property 6. (Pruning using the *AMSC* lower bound) For a pattern p , if $amsc(p) > maxcost$, then it follows that $ac(p) > maxcost$, and that p is not a CEP. Moreover, any super-pattern $p_y \supset p_x$ is also not a CEP.

Proof 7. This directly follows from Property 1 and 2.

For example, consider Table 5 and that $minsup = 2$. For pattern $\langle a \rangle$, we have $asc(\langle a \rangle) = 15 < amsc(\langle a \rangle) = (20+25)/2 = 22.5 < ac(\langle a \rangle) \approx 28.3$. For pattern b , we have $asc(\langle b \rangle) = (16+20)/5 = 7.2 < amsc(\langle b \rangle) = (16+30)/2 \approx 23 < ac(\langle b \rangle) \approx 28.7$. Furthermore, $asc(\langle a, b \rangle) = ((25 + 30) + (40 + 16))/3 = 37 < amsc(\langle a, b \rangle) = \frac{(25+30)+(40+16)}{2} = 55.5 < ac(\langle a, b \rangle) = 57$. Thus, $\max\{asc(\langle a \rangle), asc(\langle b \rangle)\} < asc(\langle a, b \rangle)$, and $\max\{amsc(\langle a \rangle), amsc(\langle b \rangle)\} < amsc(\langle a, b \rangle)$. In this paper, a key consideration is to find CEP having a low average cost. Using the proposed Property 6, if the *AMSC* of a pattern is larger than *maxcost*, this pattern and all its super-patterns do not need to be considered because they are not CEP.

Table 10: A projected binary SEL

Sid	Sequence (event[cost])	Class
1	$\langle(c[9]),(d[2])\rangle$	+
4	$\langle(d[1])\rangle$	-

4.3. Computing Database Projections using a Projected Database Buffer

The designed algorithms utilize a pattern-growth approach to explore the search space of all patterns. The pattern-growth approach consists of performing a database projection for each considered pattern, and then to scan the projected database to find larger patterns that extend the pattern. Before presenting the designed algorithms, this subsection introduces the concept of database projection. Then, it presents an optimization called Projected Database Buffer to perform database projections more efficiently. Finally, the last paragraph describes how the proposed algorithms explore the search space while using the Projected Database Buffer.

Definition 15. (Database projection) Let there be a pattern p and a sequence S_s from a database SEL. The projection of S_s by p is the part of the sequence S_s that appears after the first occurrence of p in S_s , if p appears in S_s . Otherwise, it is the empty sequence. The projection of SEL by p is the set of all projected sequences by p . It is denoted as PD_p .

For example, consider the binary SEL of Table 4. The projection of this database by pattern $\langle a, b \rangle$ is shown in Table 4. Note that although this example shows the projection of a binary SEL, numeric SEL can be projected in the same way.

When a pattern-growth algorithm considers a pattern p , it projects the database by p . Then, the algorithm uses this reduced database to explore larger patterns that extend p . This reduces the cost of database scans as projected databases become smaller and smaller as larger patterns are considered. However, creating all these projections can require a considerable amount of memory and time. To address this issue, a novel Projected Database Buffer (PDBuf) structure is proposed to reuse the memory for storing projected databases. By reusing memory, runtimes also decrease as the number of memory allocation operations is reduced. The proposed PDBuf structure has two components: (1) a summary list (*sumList*) that stores general information about each projected database currently in memory, and (2) an array structure to store projected databases, called the projected database list (*pdList*). The next paragraphs present these components. Then, the following subsections explain how the designed algorithms utilize that structure and the lower bounds presented in previous subsections to efficiently discover CEP.

Definition 16. (Summary List) The summary list (*sumList*) component of the PDBuf structure is a list of summaries $sumList = \{s_0, s_1, \dots, s_n\}$, containing a summary for each projected database stored in the PDBuf structure. The notation $sumList[x]$ denotes the summary at position x in *sumList*. And the notation $sumList(a_1, a_2]$ denotes the summaries from position a_1 , exclusive, to a_2 , inclusive. A summary is a triple of the form $\{pattern, startIdx, endIdx\}$ indicating a pattern that was used to perform a database projection, and two positive integers *startIdx* and *endIdx* indicating that the projected database is stored from the $startIdx^{th}$ record to the $(endIdx - 1)^{th}$ record of the *pdList* structure.

Definition 17. (Projected Database List) The projected database list (*pdList*) component of the PDBuf structure stores projected databases. It is implemented as an array of elements $pdList = \{e_0, e_1, \dots, e_n\}$, each representing a sequence projected by a pattern. Elements representing projected sequences from the same projected database are stored consecutively in the *pdList*. The notation $pdList[x]$ denotes the event at the position x of *pdList*. And the notation $pdList(a_1, a_2]$ denotes event from position a_1 , exclusive, to a_2 , inclusive. An element is a triple $\{seqId, follIdx, cost\}$ storing the identifier *seqId* of a projected sequence, an integer *startIndex* indicating the position in the original sequence where the projected sequence starts, and the pattern's cost in that sequence. Information about the cost is stored in elements to be able to quickly calculate the cost of patterns.

For example, consider a database containing three sequences: $S_1 = \langle(a[2]), (b[4]), (c[9]), (d[2])\rangle$, $S_2 = \langle(a[5]), (e[4]), (b[8])\rangle$ and $S_3 = \langle(a[2]), (b[3]), (e[4]), (c[2])\rangle$. Figure 2 shows a projected database buffer containing the corresponding projected databases of patterns $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle e \rangle$ (represented with different colors). The first summary

in the *pdList*, denoted as *pdList*[0] is the summary of the projected database of $\langle a \rangle$. This summary contains the tuple $\{pattern = a, startIdx = 0, endIdx = 3\}$ indicating that this projection consists of three projected sequences stored in records 0, 1 and 2 of the *pdList*. The record 0 of the *pdList*, denoted as *pdList*[0], contains the tuple $\{seqId = 1, follIndex = 1, cost = 2\}$. This record represents the projection of sequence 1 with item a . That projected sequence starts from position *follIndex* = 1 of sequence S_1 . In other words, that projected sequence is $\langle (b[4]), (c[9]), (d[2]) \rangle$. The value *cost* = 2 indicates that the cost of a in that sequence is 2. Other records follow the same format.

In the *pdList*, each projected sequence for a pattern p is represented by a triple $\{seqId, follIdx, cost\}$. The reason for storing the cost in each triple is to be able to quickly calculate the cost of an extension of p without having to scan the original database. This is shown with an example. Consider the projected database $PD_{\langle b \rangle}$ of pattern $\langle b \rangle$. The *pdList* contains a triple $\{1, 2, 4\}$ indicating that $\langle b \rangle$ appears at position 2 of sequence 1 with a cost of 4. Moreover, consider the projected database $PD_{\langle c \rangle}$ of pattern $\langle c \rangle$. The *pdList* contains a triple $\{1, 3, 9\}$ indicating that $\langle c \rangle$ appears at position 3 of sequence 1 with a cost of 9. Using this information, it is possible to directly derive the triple of pattern $\langle bc \rangle$ as $\{1, \max(2, 3), 9 + 4\} = \{1, 3, 13\}$. In other words, the triple of that sequence can be obtained without scanning the original database. This technique improves the efficiency of the proposed algorithms.

The proposed algorithms explore the search space using a depth-first search. In this paragraph, an illustrative example is given to explain how the search space is traversed and how the Projected Database Buffer is used during that traversal to reuse memory and thus reduce memory usage. Then, the following subsections describe the proposed algorithms in details. Consider the database of the running example. The search space contains all patterns that can be formed using events a, b, c, d and e . But using the pruning properties, some parts of the search space do not need to be explored. In this example, we will consider that one of the proposed algorithms is applied to explore the part of the search space shown in Fig. 1. The algorithm first scans the database to calculate the support, average cost and upper-bounds of each single events. Then, the algorithm eliminates patterns according to the pruning properties based on cost and support. Assume that the patterns $\langle a \rangle, \langle b \rangle, \langle c \rangle$, and $\langle e \rangle$ satisfy these constraints. Then, the projected databases $PD_{\langle a \rangle}, PD_{\langle b \rangle}, PD_{\langle c \rangle}$ and $PD_{\langle e \rangle}$ of these patterns are created and stored in the Projected Database Buffer (by updating the *sumList* and *pdList* structures). Then, a depth-first search is performed to extends each pattern. The algorithm processes patterns in reverse order and thus starts by $\langle e \rangle$. The projected database of $\langle e \rangle$ is scanned to find larger patterns extending $\langle e \rangle$ but no patterns are found. Thus, the projected database $PD_{\langle e \rangle}$ is not needed anymore in the buffer and that memory can be reused. Then, the algorithm performs the depth-first search to extend pattern $\langle c \rangle$. The algorithm scans the projected database $PD_{\langle c \rangle}$ and no patterns are found extending $\langle c \rangle$. Thus, the projected database $PD_{\langle c \rangle}$ is not needed anymore in the buffer and that memory can be reused. Then, the algorithm performs the depth-first search to find patterns extending the next pattern $\langle b \rangle$. The algorithm scans the projected database $PD_{\langle b \rangle}$ and finds that the pattern $\langle bc \rangle$ is a CEP. The projected database of $PD_{\langle bc \rangle}$ is created and stored in the buffer. It is to be noted that the projected database of $PD_{\langle bc \rangle}$ is stored by overwriting the memory previously used for storing projected databases $PD_{\langle c \rangle}$ and $PD_{\langle e \rangle}$ (depending on how much memory is needed), which thus reduces memory usage because memory is reused. The projected database buffer now contains $PD_{\langle a \rangle}, PD_{\langle b \rangle}$, and $PD_{\langle bc \rangle}$. Then, the algorithm continues the depth-first search to find patterns extending $\langle bc \rangle$. The algorithm scans the projected database $PD_{\langle bc \rangle}$ and no patterns are found extending $\langle bc \rangle$. Thus, the projected database $PD_{\langle bc \rangle}$ is not needed anymore in the buffer and that memory can be reused. Since the depth-first search has finished extending the pattern $\langle b \rangle$, the projected database $PD_{\langle b \rangle}$ is not needed anymore in the buffer and that memory can be reused. Then, the algorithm continues the depth-first search to find patterns extending the next pattern $\langle a \rangle$. The algorithm scans the projected database $PD_{\langle a \rangle}$ and finds that the pattern $\langle ab \rangle, \langle ac \rangle$, and $\langle ae \rangle$ are CEPs. The algorithm thus stores their projected databases $PD_{\langle ab \rangle}, PD_{\langle ac \rangle}$ and $PD_{\langle ae \rangle}$ in the buffer by overwriting the memory used for storing the previously stored projected databases that are not needed anymore. The projected database buffer now contains $PD_{\langle a \rangle}, PD_{\langle ab \rangle}, PD_{\langle ac \rangle}$ and $PD_{\langle ae \rangle}$. Then, the algorithm continues the depth-first search in a similar way to extend the patterns $\langle ab \rangle, \langle ac \rangle$, and $\langle ae \rangle$. Assume that no more patterns are found. The algorithm stops, and all the CEPs have been found.

As can be seen in the above example, by using the buffer while doing the depth-first search, memory is often reused rather than allocating new memory, which reduces memory usage. As it will be shown in the experiment, this can considerably improve performance.

It is to be noted that how the proposed Projected Database Buffer structure is used in the proposed algorithms has some similarity to how the Utility-List Buffer structure is used in itemset mining [52]. The similarity is that both structures are employed to reuse memory during a depth-first search to find patterns. However, there are two important

differences. First, the type of patterns is different and the data stored in the two buffers is different. The latter stores vertical structures for each itemset, while this work store horizontal projected databases for each pattern. Second, another important difference is that this work proposes to use the backward scanning order during the depth-first search. Using this order has the benefit of allowing to reuse more memory compared to how the Utility-List Buffer is used [52]. This is illustrated with the previous example. Consider that the pattern $\langle a \rangle$ is first processed by the depth first search rather than $\langle e \rangle$. Then, the memory of $\langle a \rangle$ cannot be reused after processing $\langle a \rangle$ because other projected databases appearing after $\langle a \rangle$ have not been yet processed and they have been inserted in the buffer after $\langle a \rangle$. On the other hand, if the backward order is used, the memory of $PD_{\langle e \rangle}$ can be reused immediately after processing $\langle e \rangle$ because $PD_{\langle e \rangle}$ is the last inserted projected database in the buffer. A more general observation is that if the proposed backward processing order is not used, the memory used for the projected databases of single events could never be reused. Thus, the backward order is preferable. The next subsections presents the three proposed algorithms.

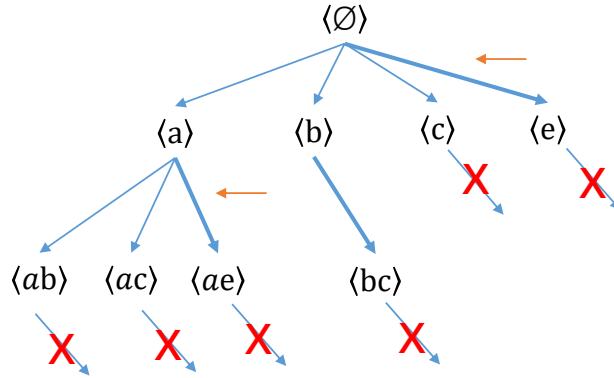


Figure 1: An example of search space traversal using the projected database buffer

pattern:	a	b	c	e						
startIdx:	0	3	6	8						
endIdx:	3	6	8	10						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	4	8	3	9	2	4	4

Figure 2: A Projected Database Buffer containing the projected databases of a , b , c and e

4.4. The CEPB Algorithm

A first algorithm, named CEPB (Cost-Effective Pattern mining in Binary SEL), is designed to find all cost-effective patterns in a binary SEL when considering only positive sequences (the problem of Case 1, Definition 6). The main procedure of CEPB is presented in Algorithm 1. It takes as input a binary SEL, the $minsup$ and $maxcost$ thresholds, and the $sumList$ and $pdList$ components of the projected database buffer, which are initially empty.

The procedure reads the database to calculate the support, average cost, and ASC (or $AMSC$) of each single event (Line 1). For each event a such that $sup(a) \geq minsup$ and $ac(a) \leq maxcost$, the pattern $\langle a \rangle$ is output as a cost-effective pattern (Line 6). Thereafter, the condition $asc(\langle a \rangle) \leq maxcost$ of the search space reduction Property 3 is checked (Line 8). If the condition is false, it means that the algorithm does not need to extend the pattern $\langle a \rangle$ with other events to form larger patterns because all such patterns cannot be cost-effective patterns. Note that the ASC lower bound can be replaced by the $AMSC$ in Line 8 to obtain the more powerful pruning Property 6, since the $AMSC$ is a tighter lower bound on the average cost than the ASC . If the condition $asc(\langle a \rangle) \leq maxcost$ is true, then the Update procedure

<p>input : a SEL D, $minsup$, $maxcost$, the summary list $sumList$, the projected database list $pdList$</p> <p>output: the cost-effective patterns found in positive sequences</p> <pre> 1 Read D once to obtain the support, average cost and ASC of each event; 2 $endPos \leftarrow 0$; 3 foreach event a do 4 if ($sup(a) \geq minsup$) then 5 if ($ac(a) \leq maxcost$) then 6 Save(a); 7 end 8 if ($asc(a) \leq maxcost$) then 9 CreateProjectedDatabase(a, $sumList$, $pdList$, $endPos$); 10 end 11 end 12 else Remove event a from the database D; 13 end 14 Search(0, $endPos - 1$, $sumList$, $pdList$); </pre>

Algorithm 1: The CEPB algorithm

(Algorithm 2) is called with the parameters $\langle a \rangle$, $sumList$, $pdList$ and the variable $endPos$ to create the projected database $PD_{\langle a \rangle}$ and store it in the projected database buffer. The variable $endPos$ indicates the position in the $sumList$ where the summary of $PD_{\langle a \rangle}$ should be stored. If the event a is infrequent, then it is removed from the database or ignored during further processing (Line 13). After the for loop ends, the projected databases of several events have been stored in the buffer. Then, the Search procedure (Algorithm 3) is called to explore the search space of patterns having the empty sequence $\langle \rangle$ as prefix.

The Update procedure (Algorithm 2) takes as input an event a , the summary list $sumList$, the projected database list $pdList$, and the position $sumIndex$ where the summary for the projected database of $\langle a \rangle$ will be stored in the $pdList$. If $sumIndex$ is no less than the current size of $sumList$, the procedure appends a new summary at the end of the $sumList$ (Line 1). Otherwise, the procedure stores the summary of a at the position $sumIndex$, overwriting the summary previously stored at that position. By this mechanism, memory is reused rather than allocating new memory. Information is stored in the summary of $\langle a \rangle$ as follows. First, the field *pattern* is set to $\langle a \rangle$ (Line 2). Then, the start and end positions for storing the projected sequences of $\langle a \rangle$ in the $pdList$ must be determined. If $sumIndex$ is 0, $PD_{\langle a \rangle}$ will be the first projected database in the $pdList$, and thus the fields *startPos* and *endPos* are set to 0 (Line 3 to 6). Otherwise, these fields are set such that the projected sequences of $\langle a \rangle$ will follow those of the previous summary (Line 7 to 10). After that, the Update procedure creates the projected sequences of $PD_{\langle a \rangle}$ and stores them in the $pdList$. This is done by scanning each sequence (Line 11 to 18). For a sequence with an identifier s , the algorithm finds the position i of the first occurrence of $\langle a \rangle$ and its cost. Then, a corresponding projected sequence is stored in the $pdList$ as the tuple $\{s, i, cost\}$ either by appending a new record to the $pdList$ (Line 15) or by reusing a record of the $pdList$ that is not needed anymore (Line 16). Finally, the *endPos* field of the summary of $\langle a \rangle$ is set to remember where the next projected sequence should be inserted in the $pdList$ (Line 17).

The Search procedure (Algorithm 3) takes two positions $preStartPos$ and $preEndPos$ of the summary list as input, and the projected database buffer. Between the $preStartPos$ and $preEndPos$ positions, the database buffer contains summaries for patterns of the form $p = v \cup z$, where v is a common prefix sequence and z is some event. The Search procedure iterates over these summaries in backward order (Line 1 to 14). For each summary corresponding to a pattern p , the procedure scans its projected database PD_p stored in the $pdList$ (which is stored between positions *startIndx* to *endIndx*) (Line 4). During that database scan, the procedure calculates for each event a appearing in PD_p , the support, average cost and ASC (or AMSC) of $\langle p \cup a \rangle$. If $sup(\langle p \cup a \rangle) \geq minsup$ and average cost of $\langle p \cup a \rangle$ is no greater than $maxcost$, the pattern $\langle p \cup a \rangle$ is output as a CEP (Line 7). After, if the pruning condition of Property 3 is passed, the Update procedure is called to calculate the projected database $PD_{\langle p \cup a \rangle}$ and it is stored in the projected database buffer (Line 8 to 10). Then, the Search procedure is recursively called to find each CEP having $\langle p \cup a \rangle$ as prefix (Line 13). When the for loop ends, all CEP patterns having p as prefix have been found.

input : an event a , the summary list $sumList$, the projected database list $pdList$, a position $sumIndex$
output: empty

```

1 if ( $sumIndex \geq |sumList|$ ) then  $sumList[sumIndex].append(\{0,0,0\})$ ;
2  $sumList[sumIndex].pattern \leftarrow a$ ;
3 if ( $sumIndex = 0$ ) then
4    $curSummary.startPos \leftarrow 0$ ;
5    $sumList[sumIndex].endPos \leftarrow 0$ ;
6 end
7 else
8    $sumList[sumIndex].startPos \leftarrow sumList[sumIndex - 1].endPos$ ;
9    $sumList[sumIndex].endPos \leftarrow sumList[sumIndex].startPos$ ;
10 end
11 foreach sequence identifier  $pdList[preEndPos].seqId$  as  $s$ , scan the sequence  $s$  from the position
     $pdList[preEndPos].folInx$  as  $f$  do
12    $i \leftarrow$  position of the first occurrence of  $a$  in sequence  $s$ ;
13    $cost \leftarrow$  cost of the first occurrence of  $a$  in sequence  $s$ ;
14    $insertPosition \leftarrow sumList[sumIndex].endPos$ ;
15   if ( $insertPosition \geq |pdList|$ ) then  $pdList.append(\{s, i, cost\})$ ;
16   else  $pdList[insertPosition] \leftarrow \{s, i, cost\}$ ;
17    $sumList[sumIndex].endPos \leftarrow sumList[sumIndex].endPos + 1$ ;
18 end

```

Algorithm 2: The Update procedure of CEPB

input : two positions $preStartPos$ and $preEndPos$ of the summary list, the summary list $sumList$, the projected database list $pdList$
output: the cost-effective patterns found in positive sequences having one of the pattern p from the buffer as prefix

```

1 for  $i \leftarrow preEndPos$  to  $preStartPos$  do
2    $p \leftarrow sumList[i].pattern$ ;
3    $PD_p \leftarrow$  projected database of  $p$  stored in the  $pdList$ ;
4   Scan  $PD_p$  to calculate the support, average cost and ASC (or AMSC) of  $p \cup a$  for each event  $a$  appearing
     in  $PD_p$ ;
5   foreach event  $a \in PD_p$  do
6     if ( $sup(p \cup a) \geq minsup$ ) then
7       if ( $ac(p \cup a) \leq maxcost$ ) then Save( $p \cup a$ );
8       if ( $asc(p \cup a) \leq maxcost$ ) then
9         CreateProjectedDatabase( $p \cup a, sumList, pdL, endPos$ );
10      end
11    end
12  end
13  Search( $p \cup a, preStartPos + 1, endPos, sumList, pdList$ );
14 end

```

Algorithm 3: The Search procedure of CEPB

The algorithm explores the search space using a depth-first search by recursively growing patterns a single event at a time, which allows to explore the search space of all patterns. To reduce the search space, the CEPB algorithm applies the *minsup* constraint used in traditional sequential pattern mining, as well as a novel search space pruning property based on the cost. Because these pruning techniques only eliminate patterns that are not cost-effective, the algorithm is complete (it finds all the desired patterns). To reduce memory usage, the algorithm uses the projected database buffer. As it will be shown in the experiments, using the novel pruning properties based on the average cost and the proposed buffering technique can greatly reduce runtime and memory usage.

A Detailed Example. Consider the following database: $\langle\langle a[2], (b[4]), (c[9]), (d[2]), + \rangle, \langle\langle a[5], (e[4]), (b[8]), + \rangle, \langle\langle a[2], (b[3]), (e[4]), (c[2]), + \rangle$, and that $minsup = 2$ and $maxcost = 10$. The CEPB algorithm first scans the database to calculate the support values of $\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle$ and $\langle e \rangle$, which are 3, 3, 2, 1 and 2, respectively. Because the pattern $\langle d \rangle$ is infrequent, it is eliminated from the database. Then, CEPB calculates the average costs of $\langle a \rangle, \langle b \rangle, \langle c \rangle$ and $\langle e \rangle$, which are 3.0, 5.0, 5.5, and 4.0, respectively. Because these values are no greater than $maxcost$, all these events are output as cost-effective patterns. Then, CEPB calculates the lower-bound ASC of $\langle a \rangle, \langle b \rangle, \langle c \rangle$, and $\langle e \rangle$, which are 2.0, 3.5, 5.5, and 4.0, respectively. Thereafter, the algorithm creates the projected database buffer. A summary is stored for the patterns $\langle a \rangle, \langle b \rangle, \langle c \rangle$, and $\langle e \rangle$ in position 0 to 3 of the summary list, respectively. Moreover, the projected sequences of each pattern are stored in $pdList[0, 3]$, $pdList[3, 6]$, $pdList[6, 8]$ and $pdList[8, 10]$, respectively. The content of the projected database buffer after this step is shown in Figure 3. Then, the Search procedure is called to find larger patterns having the patterns $\langle a \rangle, \langle b \rangle, \langle c \rangle$ and $\langle e \rangle$ as prefix. The procedure receives as input $sumList[0, 3]$. The procedure do a for loop over $sumList[0, 3]$ to process each pattern in backward order from position $sumList[3]$ to $sumList[0]$. In Figure 3, the green arrow (called *sumPS*) represents the first position visited by the for loop, while the black arrow is the last position (called *sumPE*). Because of the backward processing order, the pattern $\langle e \rangle$ stored in $sumList[3]$ is first processed. The algorithm accesses $pdList[8, 10]$ to scan $PD_{\langle e \rangle}$ but no extensions of $\langle e \rangle$ are found. Then, the *sumPS* pointer of the for loop is moved to index $sumList[2]$ to process the next pattern in the buffer, as shown in Figure 4. This pattern is $\langle c \rangle$. The algorithm accesses $pdList[6, 8]$ to scan $PD_{\langle c \rangle}$ but no extensions of $\langle c \rangle$ are found. Thus, the next pattern is processed by moving *sumPS* to $sumList[1]$, as shown in Figure 5. This pattern is $\langle b \rangle$. The algorithm accesses $pdList[3, 6]$ to scan $PD_{\langle b \rangle}$, which contains the projected sequences $\langle\langle c[9], (d[2]) \rangle\rangle$ and $\langle\langle e[4], (c[2]) \rangle\rangle$. By scanning $PD_{\langle b \rangle}$, the pattern $\langle bc \rangle$ is found to be a CEP because $sup(\langle bc \rangle) = 2$ and $ac(\langle bc \rangle) = 9.0$, and it is output. Because $ASC(\langle bc \rangle) = 9.0 < 10$, the algorithm calls the Update procedure to store the projected database $PD_{\langle bc \rangle}$. For each projected sequence in $PD_{\langle bc \rangle}$, the cost is calculated as the sum of the cost of $\langle b \rangle$ in the corresponding sequence in $PD_{\langle b \rangle}$ and that of $\langle c \rangle$ in the corresponding sequence in $PD_{\langle c \rangle}$. The Search procedure is called to search for patterns having $\langle bc \rangle$ as prefix. For this, the *sumPS* and *sumPE* pointers of the for loop are both set to 2, as illustrated in Figure 6. The algorithm accesses $pdList[6, 8]$ to scan $PD_{\langle bc \rangle}$ but no patterns are found, and thus the process of searching for patterns having $\langle bc \rangle$ as prefix is over.

After the recursive calls to Search returns, *sumPS* is moved to $sumList[0]$ to process the next pattern, as shown in Figure 7. This pattern is $\langle a \rangle$. Next, the algorithm accesses $pdList[0, 3]$ to scan $PD_{\langle a \rangle}$. The support values of patterns $\langle ab \rangle, \langle ac \rangle$ and $\langle ae \rangle$ are 3, 2, 2, respectively. Their average costs are 8.0, 7.5 and 7.5, respectively. Hence, those patterns are output as cost-effective patterns. Because the ASC of these patterns are 5.5, 7.5, 7.5, extensions of these patterns will be considered by the depth-first search. Thus, the Update procedure is called to first store the projected databases of those patterns (see Figure 7). Then, the Search procedure is called with $sumList[1, 3]$. The for loop of Line 1 considers each pattern from $sumList[3]$ to $sumList[1]$, respectively, as shown in Figure 8. The algorithm begins to search from $sumList[3]$ and ends at $sumList[1]$. However, in this example, no more CEP are found. The for loop is completed, the algorithm stops and all cost-effective patterns have been output. It can be seen in these figures that the size of the projected database buffer does not increase much during search space exploration. This shows that the proposed projected database buffer is useful to reduce memory usage.

Complexity. The complexity of the CEPB algorithm is analyzed as follows. The algorithm first scans the database to calculate the support, average cost and ASC of each event in the database. The complexity of this step is $O(n \times w)$, where n and w are the number of sequences in the database and the average sequence length, respectively. After that, the algorithm removes infrequent events from the original database, which has a time cost of $O(n \times w)$. The algorithm then outputs all cost-effective patterns containing a single event. Then, the Search procedure is called for each pattern to recursively search for larger patterns using a depth-first search. For each recursive call, the algorithm takes $O(n \times w)$ time to create the projected database of the prefix pattern, and then it takes $O(n)$ time to calculate the pattern's average cost. Then, the Update procedure takes $O(n) + O(1)$ time to maintain the projected database buffer.

	↓	←	↓							
pattern:	a	b	c	e						
startIdx:	0	3	6	8						
endIdx:	3	6	8	10						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	4	8	3	9	2	4	4

Figure 3: Projected Database Buffer containing the projected databases of $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle e \rangle$

	↓	←	↓							
pattern:	a	b	c	e						
startIdx:	0	3	6	8						
endIdx:	3	6	8	10						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	4	8	3	9	2	4	4

Figure 4: Accessing the next pattern $\langle c \rangle$ in $sumList[2]$

	↓	↓								
pattern:	a	b	bc	e						
startIdx:	0	3	6	8						
endIdx:	3	6	8	10						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	4	8	3	13	5	4	4

Figure 5: Accessing the next pattern $\langle b \rangle$ in $sumList[1]$, and then inserting $PD_{\langle bc \rangle}$ in $pdList[6, 8]$

		↓	↓							
pattern:	a	b	bc	e						
startIdx:	0	3	6	8						
endIdx:	3	6	8	10						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	4	8	3	13	5	4	4

Figure 6: Accessing the next pattern $\langle bc \rangle$ in $sumList[2]$

	↓	↓								
pattern:	a	ab	ac	ae						
startIdx:	0	3	6	8						
endIdx:	3	6	8	10						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	6	13	5	11	4	9	6

Figure 7: Accessing the next pattern $\langle a \rangle$ in $sumList[0]$, and then inserting $PD_{\langle ab \rangle}$, $PD_{\langle ac \rangle}$ and $PD_{\langle ae \rangle}$ in $pdList[3, 6]$, $pdList[6, 8]$, $pdList[8, 10]$, respectively

	↓		↓							
pattern:	a	ab	ac	ae						
startIdx:	0	3	5	7						
endIdx:	3	5	7	9						
seqId:	1	2	3	1	2	3	1	3	2	3
folIdx:	1	1	1	2	3	2	3	4	2	3
cost:	2	5	2	6	18	5	11	4	9	6

Figure 8: Accessing patterns $\langle ae \rangle$, $\langle ac \rangle$ and $\langle ab \rangle$ from $sumList[3]$ to $sumList[1]$

Therefore, the complexity of the CEPB algorithm is $O(n \times w \times x)$, where x is the number of considered patterns (which depends on how the parameters are set). It is to be noted that although several database scans are performed, projected databases become smaller as the algorithm considers larger patterns. Moreover, each projected database is relatively small because database projections are created using pointers on the original database rather than by copying the database content.

4.5. The Correlated CEPB Algorithm

The second proposed algorithm, named corCEPB (correlated CEPB), is designed to find all cost-effective patterns in a binary SEL when considering not only positive but also negative sequences (the problem of Case 2, Definition 9). The main difference between CEPB and corCEPB is that the latter calculates the correlation of patterns, which requires to consider both positive and negative sequences. Information about the correlation is useful for the user to learn about the most efficient patterns.

The main procedure of corCEPB is presented in Algorithm 4. It takes as input a binary SEL D , the *minsup* and *maxcost* thresholds, and the *sumList* and *pdList* components of the projected database buffer, which are initially empty. The procedure reads the database to calculate the support, average cost, and *ASC* (or *AMSC*) of each single event (Line 1). Then, a loop is performed to consider each event a of D . If $\text{sup}(\langle a \rangle, D_a^+)$ is equal to 0, then the pattern $\langle a \rangle$ and its extensions cannot be cost-effective patterns, and thus $\langle a \rangle$ is removed from the database or ignored from further processing (Line 3). Otherwise, if $\text{sup}(\langle a \rangle)$ is larger than *minsup* and $\text{ac}(\langle a \rangle) \leq \text{maxcost}$, then $\text{cor}(\langle a \rangle)$ is calculated and $\langle a \rangle$ is output as a cost-effective pattern (Line 5 to 9). In the case where $\text{sup}(\langle a \rangle) < \text{minsup}$, event a is removed from the database or ignored from further processing. Thereafter, the condition $\text{asc}(\langle a \rangle) \leq \text{maxcost}$ of the search space reduction Property 3 is checked (Line 10). If the condition is false, the algorithm does not need to extend the pattern $\langle a \rangle$ with other events to form larger patterns because all such patterns cannot be cost-effective patterns. Note that the lower bound *ASC* can be replaced by *AMSC* in Line 10 to obtain the more powerful pruning Property 6, since the *AMSC* is a tighter lower bound on the average cost than the *ASC*. If the condition $\text{asc}(\langle a \rangle) \leq \text{maxcost}$ is true, then the Update procedure (Algorithm 2) is called with the parameters $\langle a \rangle$, *sumList*, *pdList* and the variable *endPos* to create the projected database $PD_{\langle a \rangle}$ and store it in the projected database buffer. The variable *endPos* indicates the position in the *sumList* where the summary of $PD_{\langle a \rangle}$ should be stored and is initialized to 0 for each event a . After the for loop ends, the projected databases of several events have been stored in the buffer. Then, the Search procedure (Algorithm 5) is called to explore the search space of patterns having the empty sequence $\langle \rangle$ as prefix (Line 18).

The Search procedure (Algorithm 5) takes as input a pattern p , two positions *preStartPos* and *preEndPos* in the summary list, and the projected database buffer. Between the *preStartPos* and *preEndPos* positions, the database buffer contains summaries for patterns of the form $p = v \cup z$, where v is a common prefix sequence and z is some event. The Search procedure iterates over these summaries in backward order (Line 1 to 19). For each summary corresponding to a pattern p , the procedure scans its projected database PD_p stored in the *pdList* (which is stored between positions *startIndx* to *endIndx*) (Line 2 to 4). During that database scan, the procedure calculates for each event a appearing in PD_p , the support, average cost and *ASC* (or *AMSC*) of $\langle p \cup a \rangle$. If the $\text{sup}(\langle a \rangle, PD_{\langle a \rangle}^+)$ is equal to 0, then the pattern $\langle p \cup a \rangle$ and its extensions will be ignored (Line 3). Otherwise, if $\text{sup}(\langle a \rangle) \geq \text{minsup}$ and $\text{ac}(\langle p \cup a \rangle) \leq \text{maxcost}$, then $\text{cor}(\langle p \cup a \rangle)$ is calculated and pattern $\langle p \cup a \rangle$ is output as a CEP (Line 5 to 9). After, if the pruning Property 3 is passed, the Update procedure is called to calculate the projected database $PD_{\langle p \cup a \rangle}$ and store it in the projected database buffer (Line 10 to 12). Then, the Search procedure is recursively called to find each CEP having $\langle p \cup a \rangle$ as prefix (Line 18). When the for loop ends, all CEP patterns having p as prefix have been found.

A Detailed Example. Consider a database D containing the sequences $\langle (a[2]), (b[4]), (c[9]), (d[2]), + \rangle$, $\langle (a[5]), (e[4]), (b[8]), - \rangle$, $\langle (a[2]), (b[3]), (e[4]), (c[2]), - \rangle$, that *minsup* = 2 and *maxcost* = 10. The algorithm scans the database to calculate the support, average cost and *ASC* (or *AMSC*) of each event. The support values of $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, and $\langle e \rangle$ in positive sequences are 1, 1, 1, 1, 0, respectively. The pattern $\langle e \rangle$ is eliminated because it is infrequent. The support values of $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, and $\langle e \rangle$ in all sequences are 2, 3, 2, 1, 2, respectively. The pattern $\langle d \rangle$ is eliminated because $\text{sup}(\langle d \rangle) < \text{minsup}$. The algorithm loops over single events, and it is found that the average cost values of $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$ are 3.0, 5.0, and 5.5, respectively. Moreover, their correlation are -0.5 , -0.33 , and 1.0 , respectively. Therefore, $\langle a \rangle$, $\langle b \rangle$ and $\langle c \rangle$ are output as CEPs. The algorithm calculates the *ASC* of $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$, which are 2.0, 3.5, 5.5, respectively. Because these values are no greater than *maxcost*, the projected databases $PD_{\langle a \rangle}$, $PD_{\langle b \rangle}$, and $PD_{\langle c \rangle}$ are stored in the projected database buffer by updating the *sumList* and *pdList* (see Figure 9). More precisely, the projected databases of these patterns are stored in *pdList*[0, 3), *pdList*[3, 6) and *pdList*[6, 8), respectively. Then,

input : a binary SEL D , $minsup$, $maxcost$, the summary list $sumList$, the projected database list $pdList$
output: the cost-effective patterns

```

1 Read  $D$  once to calculate the support, average cost and  $ASC$  (or  $ASMC$ ) of each event;
2  $endPos \leftarrow 0$ ;
3 foreach event  $a$  do
4   if  $sup(a, D_a^+) \neq 0$  then
5     if  $(sup(a) \geq minsup)$  then
6       if  $(ac(a) \leq maxcost)$  then
7         if  $(sup(a, D_a^+) = sup(a))$  then  $cor(a) \leftarrow 1$ ;
8         else Calculate  $cor(a)$ ;
9         Save( $a$ );
10        if  $asc(a) \leq maxcost$  then
11          CreateProjectedDatabase( $a, sumList, pdL, endPos$ );
12        end
13      end
14    end
15    else Remove event  $a$  from the database;
16  end
17 end
18 Search( $0, endPos - 1, sumList, pdList$ );

```

Algorithm 4: The corCEPB algorithm

input : two positions $preStartPos$ and $preEndPos$ of the summary list, the summary list $sumList$, the projected database list $pdList$

output: the cost-effective patterns having p as prefix

```

1 for  $i \leftarrow preEndPos$  to  $preStartPos$  do
2    $p \leftarrow sumList[i].pattern$ ;
3    $PD_p \leftarrow$  projected database of  $p$  stored in the  $pdList$ ;
4   Scan  $PD_p$  to calculate the support, average cost and  $ASC$  (or  $AMSC$ ) of  $p \cup a$  for each event  $a$  appearing
   in  $PD_p$ ;
5   foreach event  $a \in PD_p$  do
6     if  $sup(a, D_a^+) \neq 0$  then
7        $endPos \leftarrow 0$ ;
8       if  $(sup(a) \geq minsup)$  then
9         if  $(ac(p \cup a) \leq maxcost)$  then
10          if  $(sup(a, D_a^+) = sup(a))$  then  $cor(p \cup a) \leftarrow 1$ ;
11          else calculate  $cor(p \cup a)$ ;
12          Save( $p \cup a$ );
13          if  $asc(p \cup a) \leq maxcost$  then Update( $p \cup a, sumList, pdL, endPos$ );
14        end
15      end
16    end
17  end
18  Search( $preStartPos + 1, endPos, sumList, pdList$ );
19 end

```

Algorithm 5: The Search procedure of corCEPB

the Search procedure is called to find larger patterns having the patterns $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$ as prefix. The procedure receives as input $sumList[0, 2]$. The procedure do a for loop over $sumList[0, 2]$ to process each pattern in backward order from position $sumList[2]$ to $sumList[0]$. In Figure 9, the green arrow (called $sumPS$) represents the first position visited by the for loop, while the black arrow is the last position (called $sumPE$). Because of the backward processing order, the pattern $\langle c \rangle$ stored in $sumList[2]$ is first processed. The projected database $PD_{\langle c \rangle}$ stored in $pdList[6, 8]$ is scanned, and no patterns extending $\langle c \rangle$ are found. Then, the $sumPE$ pointer is moved to $sumList[1]$, which contains the pattern $\langle b \rangle$, as shown in Figure 10. The projected database $PD_{\langle b \rangle}$ stored in $pdList[3, 6]$ is scanned, which contains the sequences $\langle (c[9]), (d[2]), + \rangle$ and $\langle (e[4]), (c[2]), - \rangle$. It is found that in that projected database, $sup(\langle bc \rangle) = 2 \geq 2$ and $ac(\langle bc \rangle) = 9, 0 \leq 10$. Hence, $\langle bc \rangle$ is output as a CEP. Moreover, since $ASC(\langle bc \rangle) = 9 < maxcost$, extensions of $\langle bc \rangle$ will be considered. Thus, its projected database is inserted in the buffer, as shown in Figure 10. Then the $sumPS$ and $sumPE$ pointers are both moved to $sumList[2]$ (as shown in Figure 11) and the algorithm searches for extensions of $\langle bc \rangle$. No patterns are found, and the recursive calls return and $sumPS$ is moved to $sumList[0]$ (as shown in Figure 12) to consider extensions of the pattern $\langle a \rangle$. The algorithm accesses $pdList[0, 3]$ to scan the projected database $PD_{\langle a \rangle}$, which contains the projected sequences $\langle (b[4]), (c[9]), (d[2]), + \rangle$ and $\langle (e[4]), (b[8]), - \rangle$. It is found that $sup(\langle ab \rangle) = 2$, $sup(\langle ac \rangle) = 2$, $ac(\langle ab \rangle) = 8.0$ and $ac(\langle ac \rangle) = 7.5$. Thus, these patterns satisfy the constraints and their correlation is calculated as $cor(\langle ab \rangle) = -0.4$ and $cor(\langle ac \rangle) = 1.0$. Moreover, $\langle ab \rangle$ and $\langle ac \rangle$ are output as CEPs. Then, the ASC values are calculated as $ASC(\langle ab \rangle) = 5.5 \leq 10$ and $ASC(\langle ac \rangle) = 7.5 \leq 10$. Thus, the projected databases $PD_{\langle ab \rangle}$ and $PD_{\langle ac \rangle}$ are inserted in the buffer. Then, the Search procedure is called with $sumList[1, 2]$ to extend these patterns. The algorithm first processes the pattern at $sumList[2]$, which is $\langle ac \rangle$ (as shown in Figure 13). The algorithm accesses $pdList[6, 8]$ to scan $PD_{\langle ac \rangle}$ but finds that no patterns are satisfying the support threshold. Then, the algorithm considers the next pattern at $sumList[1]$, which is $\langle ab \rangle$. The algorithm accesses $pdList[3, 6]$ to scan $PD_{\langle ab \rangle}$ but finds that no patterns are satisfying the constraints. Then, the for loop is completed, and the search process ends. All CEPs have been output.

Complexity. The complexity of the corCEPB algorithm is similar to that of the CEPB algorithm. The main difference between these two algorithms is that the correlation can be calculated for patterns in addition to the support, average cost and lower bound on the cost. This calculation takes $O(n)$ time where n is the number of sequences in the database. But on overall, the complexity of the coCEPB algorithm remains $o(n \times w \times x)$ where x is the number of considered patterns and w is the average sequence length.

4.6. The CEPN Algorithm

The third proposed algorithm, named CEPN (Cost-Effective high utility Pattern mining in Numeric SEL) is designed to find all cost-effective patterns in a numeric SEL (the problem of Case 3, Definition 12). The main procedure of CEPN is presented in Algorithm 6. It takes as input a numeric SEL D , the $minsup$ and $maxcost$ thresholds, and the $sumList$ and $pdList$ components of the projected database buffer, which are initially empty.

The procedure first reads the database to calculate the support, average cost, and ASC (or $AMSC$) of each single event (Line 1). Then, a loop is performed to consider each event a . If $sup(\langle a \rangle) \geq minsup$ and the average cost of $\langle a \rangle$ is no greater than $maxcost$, the algorithm calculates the trade-off of $\langle a \rangle$ (Line 6) and output $\langle a \rangle$ as a cost-effective pattern (Line 7). Thereafter, the condition $asc(\langle a \rangle) \leq maxcost$ of the search space reduction Property 3 is checked (Line 9). If the condition is false, it means that the algorithm does not need to consider extensions of the pattern $\langle a \rangle$ because all such patterns cannot be cost-effective patterns. Note that the ASC lower bound can be replaced by the $AMSC$ in Line 9 to obtain the more powerful pruning Property 6. If the condition $asc(\langle a \rangle) \leq maxcost$ is true, then the CreateProjectedDatabase procedure (Algorithm 2) is called with the parameters a , $sumList$, $pdList$ and the variable $endPos$ to create the projected database $PD_{\langle a \rangle}$ and store it in the buffer. The variable $endPos$ indicates the position in the $sumList$ where the summary of $PD_{\langle a \rangle}$ should be stored and is initialized to 0. If the event a is infrequent, then it is removed from the database or ignored from further processing (Line 14). After the for loop ends, the projected databases of several events have been stored in the buffer. Then, the Search procedure (Algorithm 7) is called to explore the search space of these patterns having the empty sequence $\langle \rangle$ as prefix.

The Search procedure (Algorithm 7) takes as input two positions $preStartPos$ and $preEndPos$ of the summary list as input, and the projected database buffer. Between the $preStartPos$ and $preEndPos$ positions, the database buffer contains summaries for patterns of the form $p = v \cup z$, where v is a common prefix sequence and z is some event. The Search procedure iterates over these summaries in backward order (Line 1 to 17). For each summary corresponding to a pattern p , the procedure scans its projected database PD_p stored in the $pdList$ (which is stored between positions

Diagram illustrating the merging process. The left array contains elements **a**, **b**, and **c** with **startInx: 0** and **endInx: 3**. The right array contains elements **a**, **b**, and **bc** with **startInx: 0** and **endInx: 6**. The result array shows the sequence of elements and their associated costs:

	1	2	3	1	2	3	1	3
seqId:	1	2	3	1	2	3	1	3
follInx:	1	1	1	2	3	2	3	4
cost:	2	5	2	4	8	3	9	2

Figure 9: Projected Database Buffer containing the projected databases of Figure 10: Accessing the next pattern $\langle b \rangle$ in $sumList[1]$, and then inserting $\langle a \rangle$, $\langle b \rangle$ and $\langle c \rangle$ $PD_{\langle bc \rangle}$ in $pdList[6,8]$

pattern:	a	b	bc					
startIdx:	0	3	6					
endIdx:	3	6	8					
seqId:	1	2	3	1	2	3	1	3
folIdx:	1	1	1	2	3	1	3	4
cost:	2	5	2	4	8	3	13	5

Figure 11: Accessing the next pattern $\langle bc \rangle$ in *sumList*[2]



								
pattern:	a	ab	ac					
startIdx:	0	3	6					
endIdx:	3	6	8					
seqId:	1	2	3	1	2	3	1	3
folIdx:	1	1	1	2	3	1	3	4
cost:	2	5	2	6	13	5	11	4

Figure 13: Accessing the patterns $\langle ac \rangle$ and $\langle ab \rangle$ from $sumList[2]$ to $sumList[1]$

<p>input : a numeric SEL D, $minsup$, $maxcost$, the summary list $sumList$, the projected database list $pdList$</p> <p>output: the cost-effective patterns</p> <pre> 1 Read the SEL once to calculate the support, average cost and ASC of each event; 2 $endPos \leftarrow 0$; 3 foreach event a do 4 if ($sup(a) \geq minsup$) then 5 if ($ac(a) \leq maxcost$) then 6 Calculate $tf(a)$; 7 Save(a); 8 end 9 if ($asc(a) \leq maxcost$) then 10 CreateProjectedDatabase($a, sumList, pdL, endPos$); 11 end 12 end 13 else 14 Remove event a from the database 15 end 16 end 17 Search($0, endPos - 1, sumList, pdList$); </pre>

Algorithm 6: The CEPN algorithm

$startIdx$ to $endIdx$) (Line 2 to 4). During that database scan, the procedure calculates for each event a appearing in PD_p , the support, average cost and ASC (or AMSC) of $\langle p \cup a \rangle$. If $sup(\langle p \cup a \rangle) \geq minsup$ and average cost of $\langle p \cup a \rangle$ is no greater than $maxcost$, the trade-off of $\langle p \cup a \rangle$ is calculated and the pattern is output as a CEP (Line 6 to 10). After, if the pruning condition of Property 3 is passed, the CreatedProjectedDatabase procedure is called to calculate the projected database $PD_{\langle p \cup a \rangle}$ and store it in the buffer (Line 8 to 10). Then, the Search procedure is recursively called to find each CEP having $\langle p \cup a \rangle$ as prefix (Line 13). When the for loop ends, all CEP patterns having p as prefix have been found.

When the algorithm terminates, all the required patterns have been found. Before presenting the patterns to the user, they can then be sorted by average utility and/or trade-off. The main originality of the CEPN algorithm is to use the trade-off measure to quantify the efficiency of patterns for the case of a numeric SEL.

A Detailed Example. Consider a database D containing the sequences $\langle (a[2]), (b[4]), (c[9]), (d[2]), 5, \langle (a[5]), (e[4]), (b[8]), 20 \rangle$ and $\langle (a[2]), (b[3]), (e[4]), (c[2]), 10 \rangle$, that $minsup = 2$ and $maxcost = 10$. The algorithm first scans the database to calculate the support, average cost and ASC (or AMSC) of each event. The support values of $\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle$ and $\langle e \rangle$ are 3, 3, 2, 1, and 2, respectively. Because $sup(\langle d \rangle) < minsup$, the pattern $\langle d \rangle$ is eliminated from the database. The average cost values of $\langle a \rangle, \langle b \rangle, \langle c \rangle$, and $\langle e \rangle$ are 3.0, 5.0, 5.5 and 4.0, respectively. Hence, those patterns are output as CEPs with a trade-off of 0.26, 0.43, 0.73 and 0.27, respectively. The ASC values of $\langle a \rangle, \langle b \rangle, \langle c \rangle$, and $\langle e \rangle$ are 2.0, 3.5, 5.5, and 4.0, respectively. Because those values are no greater than $maxcost$, the algorithm stores the summaries of $\langle a \rangle, \langle b \rangle, \langle c \rangle$, and $\langle e \rangle$ in the buffer, at positions $sumList[0, 3]$. The projected databases $PD_{\langle a \rangle}, PD_{\langle b \rangle}, PD_{\langle c \rangle}$, and $PD_{\langle e \rangle}$, are stored in $pdList[0, 3], pdList[3, 6], pdList[6, 8]$ and $pdList[8, 10]$, respectively, as depicted in Figure 3. Then, the Search procedure is called with $sumList[0, 3]$ to consider extensions of each of those patterns. The procedure does a for loop over $sumList[0, 3]$ to process each pattern in backward order from position $sumList[3]$ to $sumList[0]$. In Figure 3, the green arrow (called $sumPS$) represents the first position visited by the for loop, while the black arrow is the last position (called $sumPE$). Because of the backward processing order, the pattern $\langle e \rangle$ stored in $sumList[3]$ is first processed. The projected database $PD_{\langle e \rangle}$ stored in $pdList[8, 10]$ is scanned, and no patterns extending $\langle e \rangle$ are found. Then, the pointer $sumPS$ of the for loop is moved to $sumList[2]$ to consider the next pattern $\langle c \rangle$, as shown in Figure 4. The algorithm accesses $pdList[6, 8]$ to scan $PD_{\langle c \rangle}$ but no patterns extending $\langle c \rangle$ satisfy the constraints. Then, $sumPS$ is moved to $sumList[1]$ (as depicted in Figure 5), and the algorithm accesses $pdList[3, 6]$ to scan $PD_{\langle b \rangle}$. This projected database contains sequences $\langle (c[9]), (d[2]), 5 \rangle$ and $\langle (e[4]), (c[2]), 10 \rangle$. It

input : two positions $preEndPos$ and $preStartPos$ of the summary list, the summary list $sumList$, the projected database list $pdList$

output: the cost-effective patterns having p as prefix

```

1 for  $i \leftarrow preEndPos$  to  $preStartPos$  do
2    $p \leftarrow sumList[i].pattern$ ;
3    $PD_p \leftarrow$  projected database of  $p$  stored in the  $pdList$ ;
4   Scan  $PD_p$  to calculate the support, average cost and ASC (or AMSC) of  $p \cup a$  for each event  $a$  appearing
   in  $PD_p$ ;
5   foreach event  $a \in PD_p$  do
6     if ( $sup(a) \geq minsup$ ) then
7       if ( $ac(p \cup a) \leq maxcost$ ) then
8         Calculate  $tf(p \cup a)$ ;
9         Save( $p \cup a$ );
10      end
11      if  $asc(p \cup a) \leq maxcost$  then
12        CreateProjectedDatabase( $p \cup a, sumList, pdL, endPos$ );
13      end
14    end
15  end
16  Search( $preStartPos + 1, endPos, sumList, pdList$ );
17 end

```

Algorithm 7: The Search procedure of CEPN

is found that the pattern $\langle bc \rangle$ is a CEP because $sup(\langle bc \rangle) = 2$ and $ac(\langle bc \rangle) = 9.0$. The pattern $\langle bc \rangle$ is thus output with its trade-off of 1.20. Because $ASC(\langle bc \rangle) = 9.0 < 10$, the algorithm will consider extensions of $\langle bc \rangle$, and thus the CreateProjectedDatabase procedure is called to create and store $PD_{\langle bc \rangle}$ in the buffer. The Search procedure is then called to search for patterns extending $\langle bc \rangle$. In the for loop of Line 1, $sumPS$ and $sumPE$ are both set to 2, as shown in Figure 6. The algorithm accesses $pdList[6, 8]$ to scan $PD_{\langle bc \rangle}$ but no patterns extending $\langle bc \rangle$ are found to satisfy the constraints. The recursive call to Search returns and the algorithms consider extending the next pattern in $sumList[0, 3]$, which is $\langle a \rangle$ at position $sumList[0]$ (as shown in Figure 7). The algorithm accesses $pdList[0, 3]$ to scan $PD_{\langle a \rangle}$. It is found that the support values of patterns $\langle ab \rangle$, $\langle ac \rangle$ and $\langle ae \rangle$ are 3, 2, and 2, respectively. Moreover, their average cost values are 8.0, 7.5 and 7.5, respectively. Hence, those are output as CEPs with a trade-off of 0.69, 1.00, 0.50, respectively. Because the ASC values of those patterns are 5.5, 7.5 and 7.5, respectively, the algorithm will consider extending these patterns. Hence, the CreateProjectedDatabase procedure is called to create and store the projected databases $PD_{\langle ab \rangle}$, $PD_{\langle ac \rangle}$, and $PD_{\langle ae \rangle}$ in the buffer. Then, the Search procedure is called to consider extensions of $\langle ab \rangle$, $\langle ac \rangle$ and $\langle ae \rangle$. The for loop of Line 1 is performed with $sumPS$ and $sumPE$ initially set to $sumList[3]$ and $sumList[1]$, respectively (as shown in Figure 8). The algorithm thus searches from $sumList[3]$ to $sumList[1]$. However, no patterns satisfy the constraints. The algorithm terminates and all CEPs have been output.

Complexity. The complexity of the CEPN algorithm is similar to that of the corCEPB algorithm. The main difference between these two algorithms is that the trade-off is calculated for patterns instead of the correlation. These two calculations take $O(n)$ time where n is the number of sequences in the database. Thus, on overall, the complexity of the CEPN algorithm remains $o(n \times w \times x)$ where x is the number of considered patterns and w is the average sequence length.

5. Experimental evaluation

This section first reports results of experiments to assess the performance of the proposed CEPB, corCEPB and CEPN algorithms. Then, a case study is presented, where the proposed algorithms have been applied on e-learning data to identify interesting cost-effective patterns. All algorithms were implemented in Java and experiments were

carried out on a computer having a 64 bit Xeon E3-1270 3.6 Ghz CPU, running the Windows 10 operating system and equipped with 64 GB of RAM.

5.1. Performance Evaluation

The performance of the algorithms was evaluated using four standard benchmark datasets commonly used to evaluate sequential pattern mining algorithms, namely Bible, BMS, SIGN and FIFA. These datasets were obtained from the SPMF software website [53], and were chosen because they have different characteristics such as dense, sparse, long and short sequences. Since these datasets do not contain utility and cost information, those values were randomly generated using a normal distribution with cost values in the $[0, 5]$ interval and utility values in the $[0, 100]$ interval, similarly to several previous studies on high utility pattern mining [54]. The Bible dataset contains 36,369 sequences with 13,905 event types and an average sequence length of 44.3 events. The BMS dataset contains 59,601 sequences with 497 event types and an average sequence length of 6.02 events. The SIGN dataset contains 730 sequences with 267 event types and an average sequence length of 104.1 events. The FIFA dataset contains 20,450 sequences with 2,990 event types and an average sequence length of 34.74 events. Each algorithm was run on each dataset while the *maxcost* and *minsup* threshold were increased to evaluate how they influence performance. On each dataset, we recorded the execution time, memory, number of candidate patterns and number of patterns found.

Three versions of the algorithm were compared (ASC: when the ASC lower bound is used for search space pruning), (AMSC: when the AMSC lower bound is used instead of the ASC), and (AMSC+Buf: which indicates that the AMSC was used and that the projected database buffer optimization is used). Results of each experiment are presented next.

5.1.1. Influence of maxcost on runtime, number of patterns and number of candidates

In the first experiment, execution times of the CEPB, corCEPB and CEPN algorithms were compared for various *maxcost* threshold values, when the *minsup* threshold is fixed. For the Bible, BMS, SIGN and FIFA datasets, the *minsup* thresholds were set to 0.1%, 0.5%, 2%, and 5%, respectively. These values were chosen empirically on each dataset to filter patterns that have a very low frequency, while ensuring that cost-effective patterns can be found. The *maxcost* threshold was decreased until the trend regarding its influence on runtime could be clearly observed. Runtime results are shown in Fig. 16, Fig. 17 and Fig. 18, respectively. Moreover, the number of patterns found and the number of candidate patterns considered are shown in Fig. 14 and Fig. 15, respectively.

It is first observed that generally as the *maxcost* threshold is decreased (and the *maxcost* constraint becomes more strict), the number of patterns decreases as well as runtimes. This shows that reducing the search space using the proposed ASC or AMSC lower bounds on the average cost measure is useful. Second, it is observed that using the AMSC instead of the ASC improves runtimes by up to 10 times on all datasets (even though these datasets have quite different characteristics). This shows that the AMSC is really better than the ASC lower bound, as it allows to eliminate considerably more candidate patterns from the search space (see Fig. 15).

A third observation is that using the projected database buffer improves runtime. This is interesting since the main purpose of this technique is to reduce memory consumption by reusing memory rather than reducing runtimes. The reason for the runtime reduction is that less operations are performed to allocate and free memory. This also means that the Java garbage collector must perform less work.

5.1.2. Influence of minsup on runtime

In a second experiment, execution times of the algorithms were compared for various *minsup* threshold values, when the *maxcost* threshold is set to a fixed value. For the Bible, BMS, SIGN and FIFA datasets, the *maxcost* thresholds were set to 50, 100, 30 and 100, respectively. These values were chosen empirically to ensure that cost-effective patterns can be found and that the trend regarding the influence of *minsup* can be clearly observed. Results are shown in Fig. 19 for the CEPN algorithm. It was observed that the support has the same influence on the two other proposed algorithms for the four datasets. This is because the support has no influence on how the average cost is used in other calculations. Thus results are not shown for CEPB and corCEPB. Generally, as *minsup* is increased, execution time decreases. This is because more infrequent patterns can be eliminated from the search space using Lemma 1 for high support values. This is in accordance with results in sequential pattern mining that have shown that the support measure is very effective for reducing the search space [3, 4, 5].

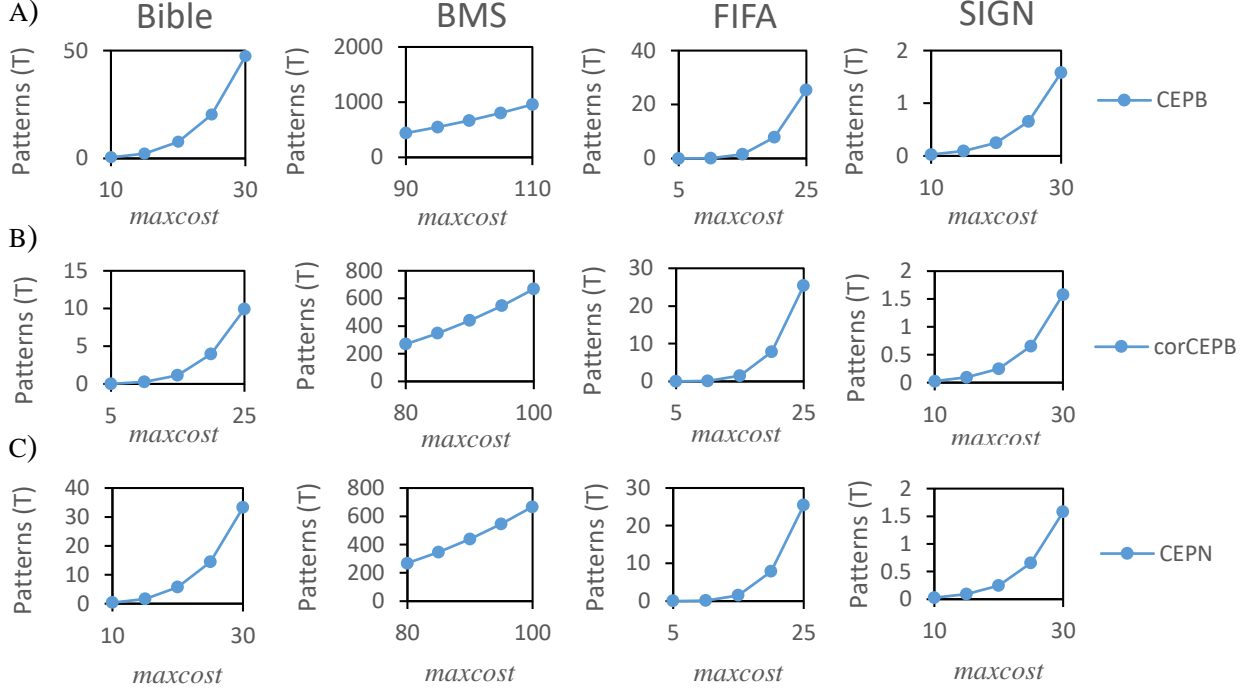


Figure 14: The number of mined patterns

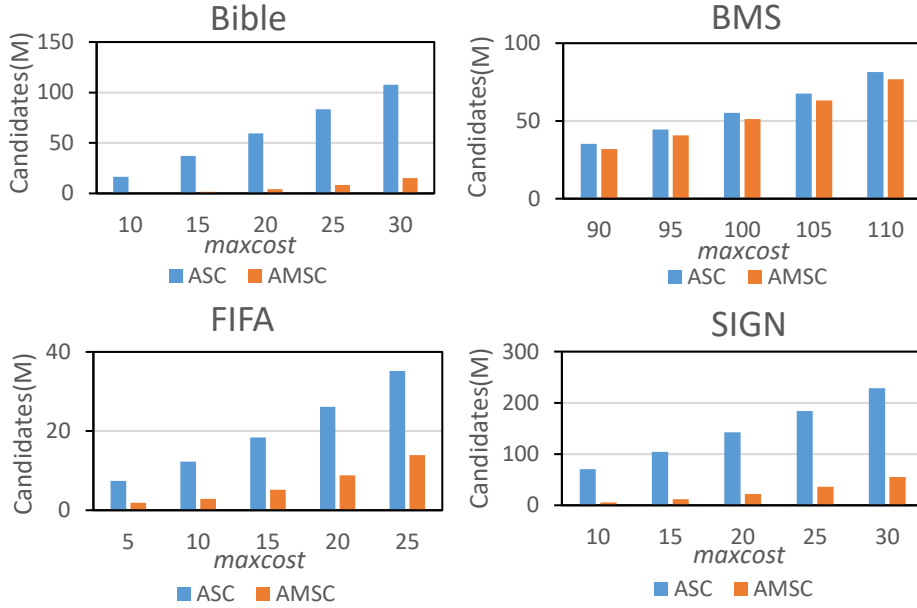


Figure 15: The number of candidate patterns

Besides, it is again observed in this experiment that using the AMSC lower bound and the buffering technique (AMSC+Buf) provides the best performance on all datasets compared to the ASC and AMSC versions of the algorithms. As *minsup* is increased, the execution time difference between the different versions of the algorithm become smaller (lines are getting closer on the chart). This is because few large patterns satisfy the *minsup* constraint, and the

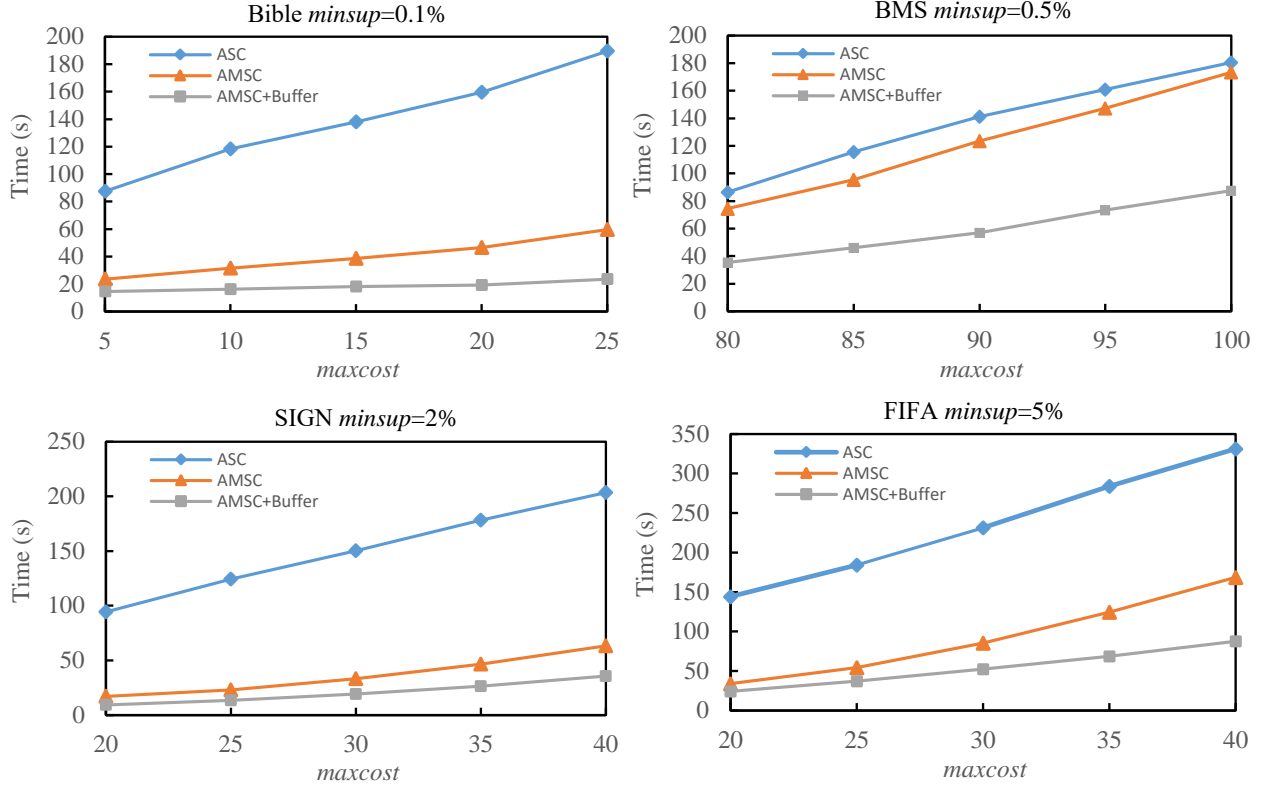


Figure 16: Runtime of CEPB when increasing *maxcost*

average cost of these patterns is generally below the *maxcost* threshold.

5.1.3. Influence of *minsup* and *maxCost* on memory usage

Table 11, 12, 13, and 14 compare the memory usage of the algorithms for various *maxcost* values for the four datasets. The *minsup* values were set to the same fixed values as in Section 5.1.1. It is first observed that when *maxcost* is increased, the three algorithms require more space to store candidate patterns. For example, when *maxcost* is increased from 30 to 35 in Table 11, the memory usage increases from 2139 to 4179 and 2102 to 4024, respectively. Moreover, it is observed that using the proposed AMSC lower bound and buffer optimization (AMSC+Buf) reduces memory usage since memory is reused. For example, in Table 14, it can be seen that using the AMSC with the projected database buffer optimization can reduce memory consumption by hundreds or even thousands of megabytes. Thus, the designed lower bound and buffering optimization not only reduce runtime but are also quite effective at reducing memory usage.

5.2. Case study in E-learning

The previous experiments were performed on standard benchmark datasets commonly used to evaluate the runtime and memory usage of pattern mining algorithms when parameters are varied. However, a limitation of these datasets is that cost and utility values are synthetic. Thus, patterns found in these datasets are not meaningful.

This section addresses this issue by analyzing patterns found in data having real cost and utility values. The goal is to verify if interesting patterns are found by the proposed algorithms. The real data is an e-learning dataset collected from the Deeds e-learning environment was used. The data was made available online at <https://archive.ics.uci.edu/ml/datasets/Data+for+Software+Engineering+Teamwork+Assessment+in+Education+Setting> by Vahdat et. al [55]. Students use Deeds to learn digital electronics by browsing learning materials, and assess their knowledge by solving problems of different difficulty levels. The dataset provides data of 115 students. The data of

Table 11: Memory comparison of CEPB, corCEPB and CEPN on the Bible dataset

<i>maxcost</i>	30		35		40		45		50	
Algorithm	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf
CEPB	2139	2102	4179	3005	4189	3019	4194	3030	4193	3038
corCEPB	596	174	847	307	1123	943	2150	1061	3052	2109
CEPN	1964	592	4152	1122	4151	1188	5377	2156	5384	3889

Table 12: Memory comparison of CEPB, corCEPB and CEPN on the BMS dataset

<i>maxcost</i>	90		95		100		105		110	
Algorithm	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf
CEPB	4228	3289	4223	3289	5616	4112	5604	4112	6028	4112
corCEPB	4170	3291	4203	3292	4171	3292	4211	3289	4216	3289
CEPN	4226	3288	4227	3291	4235	3292	4238	3292	4241	3292

Table 13: Memory comparison of CEPB, corCEPB and CEPN on the FIFA dataset

<i>maxcost</i>	5		10		15		20		25	
Algorithm	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf
CEPB	194	190	262	191	262	191	263	191	262	191
corCEPB	194	190	262	193	281	193	263	193	264	193
CEPN	194	190	261	193	281	193	263	193	264	193

Table 14: Memory comparison of CEPB, corCEPB and CEPN on the SIGN dataset

<i>maxcost</i>	10		15		20		25		30	
Algorithm	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf	ASC	AMSC+Buf
CEPB	2139	857	4185	2034	4188	2170	4203	2161	4205	3387
corCEPB	605	132	1122	293	2142	1116	3071	2124	4227	3357
CEPN	599	217	1074	445	1204	557	2174	855	4244	3243

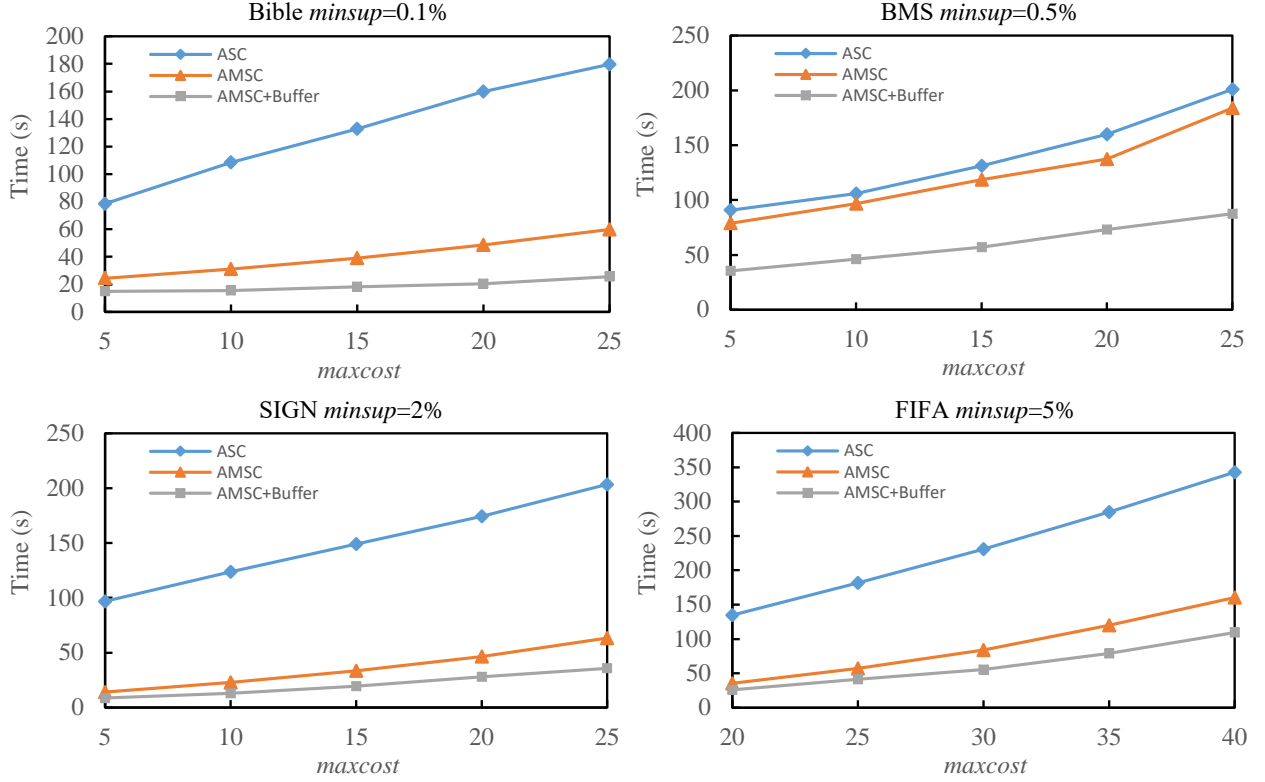


Figure 17: Runtime of corCEPB when increasing the *maxcost* threshold

each student is a sequence of learning modules, called sessions, where each session consists of learning events, each having a time duration. In addition, a score is given to each student at the end of a session based on its results at the session's exam. Moreover, a final score is given to each student based on a final exam that assesses the overall knowledge of learners about digital electronics. There are totally six learning sessions that are offered to learners but students were allowed to not do all sessions, and there was also no strict order for doing sessions. The number of event types is fifteen. The goal of the case study is to find cost-effective patterns indicating time-effective ways of studying that allows to pass the course or obtain a high score.

5.2.1. Patterns found by corCEPB

The corCEPB algorithm was first applied on the e-learning data to find sequences of learning sessions that have a low cost in terms of time but allow to pass the final exam. The data was preprocessed to obtain a binary SEL, where each sequence indicates the learning sessions completed by a student, the cost is the time spend for each learning session, and the utility is whether the student has passed or failed the final exam (a PASS or FAIL value). For the purpose of this study, the minimum passing score was assumed to be 60%.

The corCEPB algorithm was applied on the prepared data with *minsup* = 0.5 and *maxcost* = 600. Some patterns found are shown in Table 15. In that table, learning sessions are denoted as e_1, e_2, e_3, e_4, e_5 and e_6 . Furthermore, patterns are organized into three groups, separated by black bold lines. The top, center and bottom groups contain patterns having a high positive correlation, no significant correlation, and having a high negative correlation to the utility, respectively. Some interesting patterns are found. For example, the patterns $\langle e_1, e_6 \rangle$, $\langle e_1, e_2, e_5, e_6 \rangle$, $\langle e_2, e_6 \rangle$ and $\langle e_1, e_2, e_6 \rangle$ have a positive correlation with success (0.21, 0.209, 0.208 and 0.204, respectively). It was also found that some patterns such as $\langle e_4, e_5 \rangle$ and $\langle e_5 \rangle$ have a negative correlation with success (-0.109 and -0.147 , respectively). Moreover, it was found that some patterns such as $\langle e_2, e_3 \rangle$ and $\langle e_3, e_4, e_5, e_6 \rangle$ are barely correlated with the final exam result (their correlation are both 0.001).

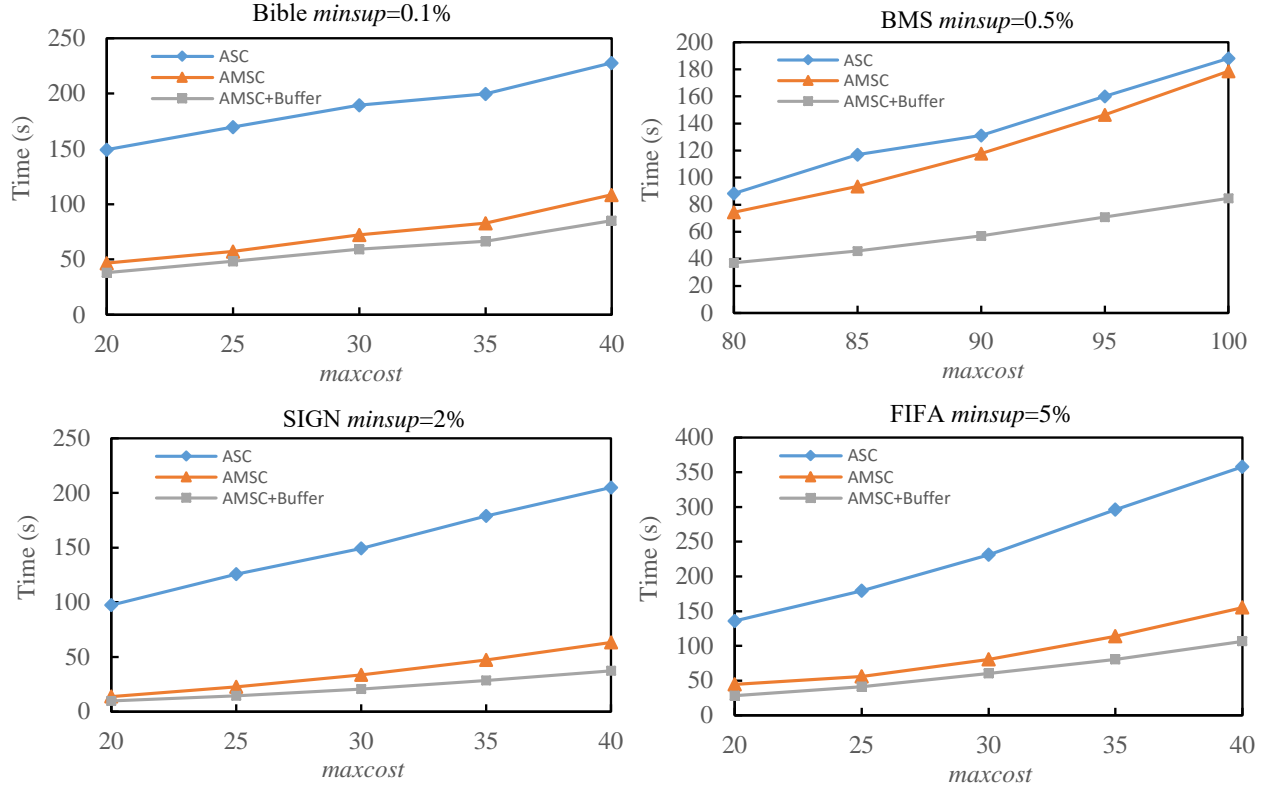


Figure 18: Runtime of CEPN when increasing the $maxcost$ threshold

Based on these patterns, it is observed that learners who did Session 1 and Session 6 are more likely to pass the final exam. On the other hand, if a student only studies Session 5, or Session 4 and Session 5, he is likely to fail the exam. Besides, if a student spends time on other unrelated sessions, it may consumes more time but may not increases much his chances of passing the final exam. Another observation is that positive correlation values are rather small in this dataset. This is because the dataset is small (only 62 students took the final exam), and that few students passed the exam when using a 60 point passing threshold. If that threshold is reduced, correlation will increase.

Note that the corCEPN algorithm was applied instead of CEPB because the latter assumes that sequences with a negative utility are not available. If CEPB was applied, the same patterns would be obtained except that correlation values would not be calculated.

5.2.2. Patterns found by CEPN

We then applied the proposed CEPN algorithm to analyze learning events for specific learning sessions of the Deeds dataset. This was done to study the relationships between events within a learning session with respect to utility and cost. For each session, a student has a sequence of events, where each event has a time duration (cost), and where the utility of a sequence is the learner's score at the session's exam. The database can thus contain multiple sequences for the same session (one for each student). Unlike in the previous subsection, the utility is here represented as a numeric value. The goal of this experiment is to obtain insights about time-efficient ways of using learning materials to obtain high scores in each session.

In this experiment, parameters were set as $minsup = 0.1$ and $maxcost = 100$ to obtain a small set of patterns. For Session 6, the average score is 14. The most efficient pattern to obtain this score is $\langle DeedsEs.6.2 \rangle$, which has a trade-off of 0.63. For Session 5, to obtain an average score of 6 the most efficient pattern is $\langle StudyEs.5.2 \rangle$, having a trade-off of 1.35. For Session 4, the average score of 14 is obtained with the pattern $\langle StudyEs.4.2 \rangle$, having a trade-off of 0.71.

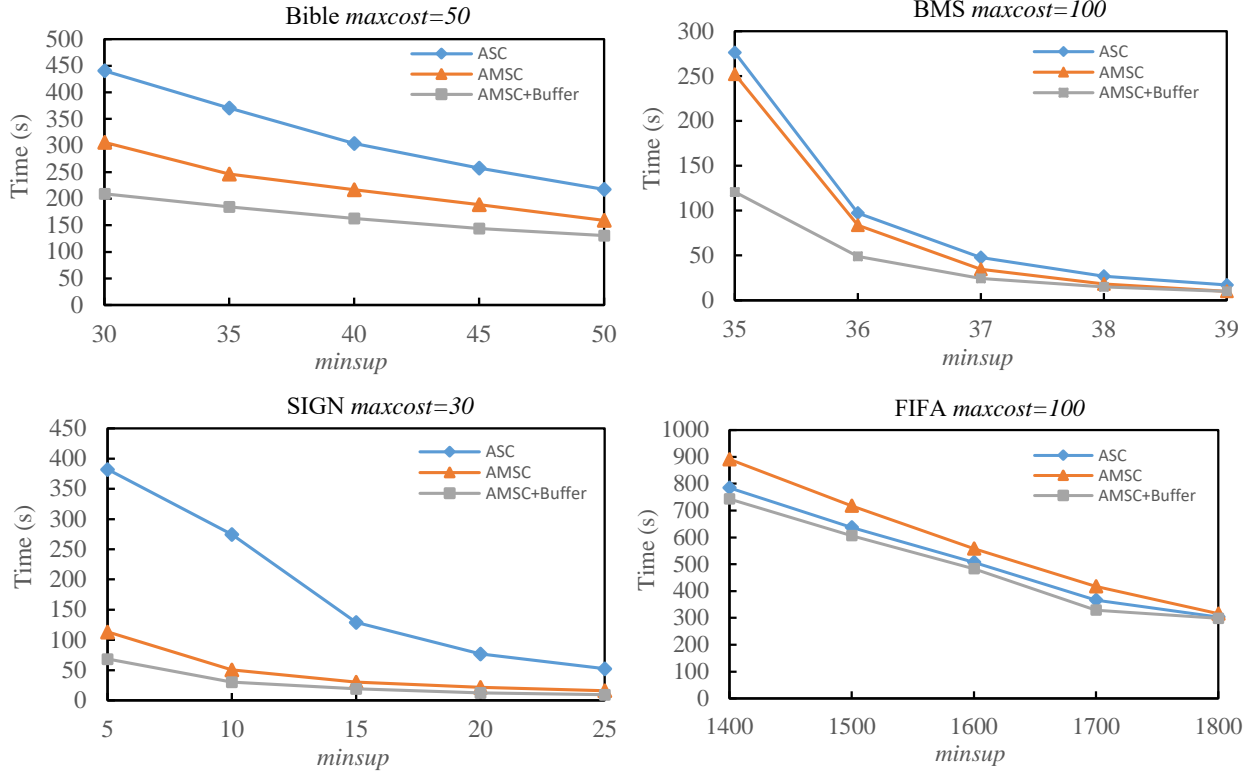


Figure 19: Runtime of CEPN when increasing the *minsup* threshold

These patterns can be interpreted as follows. To obtain a given score, students do not need to do all learning activities (events), and sometimes a single event is enough. It was also observed that some patterns having a small trade-off also have a low utility. For this reason, we suggest to consider not only the cost of patterns but also their utility. Table 16 shows the most cost-effective pattern for each utility value. It can be seen that to obtain a high score of 28(/40), the pattern having the smallest trade-off (1.35) is $\langle Deeds_Es_6_1, Deeds_Es_6_2, Study_Es_6_3, Study_Es_6_3 \rangle$. Some other patterns are also correlated with this utility but have smaller trade-off values (not shown in Table 16). Moreover, it is observed that even though some patterns have a much smaller trade-off value than the above pattern, they allow to obtain a low utility (e.g pattern $\langle Study_Es_6_1 \rangle$ has a trade-off 1.2 and utility of 9). From an application perspective, it is thus interesting to show the most cost-effective patterns for each utility range so that users can choose patterns in a specific range to obtain that utility. This is more meaningful than just showing an unsorted set of patterns.

On overall, this case study has shown that cost-efficient patterns are discovered for different ranges of utility values. A user could thus select specific patterns and apply them to attain a given utility value. To further analyze patterns, we have also checked the questions in the final exam of each session to compare them with the materials appearing in each pattern. It was found that exam questions are indeed strongly correlated with patterns. This shows that the patterns found are interesting.

5.2.3. Comparison with related work

This paper does not compare the performance (time and memory) and number of patterns found by the proposed algorithms with that of previous work because such comparison would be unfair. The reason is that the problem addressed in this paper is quite different from those of previous work, and thus the search space is not the same. A summary of the key differences between the proposed problem and those addressed in previous work was presented in Table 9. In particular, this paper does not compare the proposed algorithms with Twinkle for the following reasons:

- In terms of pattern type, Twinkle discovers sequential rules each having an antecedent and a consequent, and

Table 15: Cost-effective patterns found in sequences of learning sessions

Pattern	Correlation	Average Cost	Support
$\langle e_1, e_6 \rangle$	0.210	250.2	39
$\langle e_1, e_2, e_5, e_6 \rangle$	0.209	485.7	34
$\langle e_2, e_6 \rangle$	0.208	298.4	41
$\langle e_1, e_2, e_6 \rangle$	0.204	391.9	36
$\langle e_1, e_5, e_6 \rangle$	0.194	344.3	37
$\langle e_6 \rangle$	0.193	157.2	50
$\langle e_1, e_4 \rangle$	-0.004	169.1	41
$\langle e_1, e_5 \rangle$	0.002	186.0	41
$\langle e_2, e_3 \rangle$	0.001	284.1	40
$\langle e_3, e_4, e_5, e_6 \rangle$	0.001	469.5	40
$\langle e_1, e_4, e_5 \rangle$	0.003	263.2	38
$\langle e_1, e_2, e_4 \rangle$	-0.003	311.5	36
$\langle e_2, e_3, e_4 \rangle$	-0.005	358.2	38
$\langle e_5 \rangle$	-0.147	96.3	53
$\langle e_4, e_5 \rangle$	-0.109	171.0	49
$\langle e_1, e_3 \rangle$	-0.099	234.6	37
$\langle e_1, e_3, e_4 \rangle$	-0.081	311.2	35

where the order between items from sequences is ignored in the antecedent and consequent. Moreover, a strict constraint is imposed that an item cannot appear twice in a rule and that the antecedent and consequent must be disjoint. In this paper, the proposed algorithm discovers sequences of items that are totally ordered, and where an item can appear multiple times in the same pattern. Because of these differences, the size of the search space (the set of all possible patterns) is very different for these two problems. If there are d distinct items, the number of rules found by Twinkle is in the worst case $3^d - 2^d + 1$. In this paper, if the longest sequence has m items, the number of possible patterns is in the worst case $2^m - 1$.

- In terms of input data, Twinkle is designed for processing sequences of events, where it is not allowed for an event to appear more than once per sequence, where each event has a cost value, and where there is no utility information. On the other hand, the proposed algorithm processes sequences of events where an event may appear multiple times per sequence, where each event has a cost and where there is also utility information.
- In terms of measures used to evaluate patterns, the two algorithms are also very different and do not have the same parameters. This paper relies on measures such as the average cost, utility, correlation and trade-off, while Twinkle calculates support, confidence, and uses a sliding window to find patterns.

In terms of patterns found, this paper proposes to find cost-efficient patterns, providing a good trade-off or correlation between cost and utility. Because previous algorithms either do not consider the cost and/or utility, they can output many patterns that have a low utility and/or have a high cost. This is illustrated by comparing patterns found by the proposed CEPN algorithm with those of the state-of-the-art CM-SPADE [9] algorithm for frequent sequential pattern mining on the *Deeds* e-learning dataset. Table 17 shows some patterns found by CM-SPADE (gray color) and some patterns found by the proposed CEPN algorithm (white color), previously presented in Table 16. It is observed that for the same utility, the patterns found by CM-SPADE have a much higher average cost than those found by CEPN. For example, the two first lines of Table 17 show patterns both having a utility of 10 but having average costs of 21 and 101, respectively. The reason why CM-SPADE extracts the first pattern is that frequency is the only criterion used by CM-SPADE to select patterns. On overall, this table indicates that patterns having a high support are not necessarily cost-efficient. This shows the importance of explicitly considering cost and utility in a pattern mining

Table 16: Cost-effective patterns found in event sequences of learning session 6

Utility	Pattern	trade-off	Average Cost	Support
1	$\langle Study_Es_6_1, Study_Es_6_1, Study_Es_6_1 \rangle$	48.0	57.6	5
2	$\langle Study_Es_6_1, Study_Es_6_1, Study_Es_6_3 \rangle$	15.0	33.0	5
4	$\langle Study_Es_6_1, Study_Es_6_2, Study_Es_6_2 \rangle$	7.0	32.8	6
5	$\langle Study_Es_6_1, Study_Es_6_1 \rangle$	5.1	27.6	9
6	$\langle Study_Es_6_1, Study_Es_6_1, Deeds_Es_6_1 \rangle$	6.0	40.5	6
7	$\langle Study_Es_6_2, Study_Es_6_2 \rangle$	2.9	20.7	11
8	$\langle Study_Es_6_2, Study_Es_6_2, Deeds_Es_6_2 \rangle$	3.6	31.3	6
9	$\langle Study_Es_6_1 \rangle$	1.2	11.0	20
10	$\langle Study_Es_6_1, Deeds_Es_6_2 \rangle$	2.1	21	13
11	$\langle Study_Es_6_2, Study_Es_6_3 \rangle$	1.56	18.2	16
12	$\langle Study_Es_6_2 \rangle$	0.69	8.9	25
13	$\langle Study_Es_6_3 \rangle$	0.64	8.52	25
14	$\langle Deeds_Es_6_2 \rangle$	0.62	9.1	28
15	$\langle Study_Es_6_2, Deeds_Es_6_2, Study_Es_6_3 \rangle$	1.7	27.0	10
16	$\langle FSM_Es_6_1, FSM_Es_6_1, Deeds_Es_6_2, Study_Es_6_3 \rangle$	3.9	64.2	5
17	$\langle Deeds_Es_6_2, Study_Es_6_3 \rangle$	0.89	15.6	16
18	$\langle Study_Es_6_3, Study_Es_6_3 \rangle$	1.0	18.8	9
20	$\langle Deeds_Es_6_1, Study_Es_6_3, Study_Es_6_3 \rangle$	1.6	32.7	7
21	$\langle FSM_Es_6_3, Study_Es_6_3, Study_Es_6_3 \rangle$	4.5	94.8	6
23	$\langle Deeds_Es_6_2, Study_Es_6_3, Study_Es_6_3 \rangle$	1.2	27.0	6
24	$\langle FSM_Es_6_1, Deeds_Es_6_1, Study_Es_6_3, Study_Es_6_3 \rangle$	3.6	86.3	6
28	$\langle Deeds_Es_6_1, Deeds_Es_6_2, Study_Es_6_3, Study_Es_6_3 \rangle$	1.35	38.0	5

Table 17: Comparison of patterns found with frequent sequential pattern mining

Pattern	Utility	Average Cost	Support
$\langle Study_Es_6.1, Deeds_Es_6.2 \rangle$	10	21	13
$\langle FSM_Es_6.1, Deeds_Es_6.2, FSM_Es_6.2 \rangle$	10	101	6
$\langle Study_Es_6.2, Deeds_Es_6.2, Study_Es_6.3 \rangle$	15	27.0	10
$\langle Study_Es_6.1, FSM_Es_6.2, FSM_Es_6.3 \rangle$	15	103.5	12
$\langle FSM_Es_6.1, FSM_Es_6.1, Deeds_Es_6.2, Study_Es_6.3 \rangle$	16	64.2	5
$\langle FSM_Es_6.1, FSM_Es_6.2, Study_Es_6.2, FSM_Es_6.3, FSM_Es_6.3 \rangle$	16	295.5	8
$\langle Deeds_Es_6.1, Study_Es_6.3, Study_Es_6.3 \rangle$	20	32.7	7
$\langle FSM_Es_6.1, Deeds_Es_6.1, FSM_Es_6.2 \rangle$	20	137.7	19
$\langle FSM_Es_6.3, Study_Es_6.3, Study_Es_6.3 \rangle$	21	94.8	6
$\langle FSM_Es_6.1, FSM_Es_6.2, FSM_Es_6.3 \rangle$	21	178.3	12
$\langle Deeds_Es_6.2, Study_Es_6.3, Study_Es_6.3 \rangle$	23	27.0	6
$\langle FSM_Es_6.1, FSM_Es_6.2, Deeds_Es_6.2, Study_Es_6.3, FSM_Es_6.3 \rangle$	23	197	12

model and to measure the trade-off or correlation between cost and utility, as proposed in this paper. Algorithm for other pattern mining problems would also face the same problem as CM-SPADE as none of them considers both cost and utility and their correlation/trade-off.

We would also like to clarify that it would be hard to compare patterns found by the proposed algorithms with high utility sequential pattern mining because these problems have different input. In fact, datasets for high utility pattern mining do not contain cost information. Moreover, each item in high utility pattern mining datasets is annotated with a utility value. However, in this paper, events (items) are each annotated with a cost value, and a single utility value annotates each sequence. Thus, if one would like to apply high utility sequential pattern mining on the e-learning data used in this paper, it would require to replace cost values of items by utility values. But how to compute these utility values is a major problem. A simple approach to obtain utility values would be to divide the sequence utility equally between its items (events). In other words, the final exam score for a sequence of learning activities would be divided equally between all the activities. But this approach would assume that all activities are equally important, which may not be true. Then, one may want to subtract cost values from utility values. However, as explained in the introduction this approach does not allow to measure the correlation between cost and utility. Moreover, since cost and utility are measured using different units in the e-learning data, combining them in a single value would not only result in information loss but the resulting values would be meaningless for the user.

6. Conclusion

This article presented a novel problem of discovering cost-effective patterns in event sequences by considering both a utility and a cost model. Three versions of the problem have been defined, to be applied in different real-life scenarios. A performance evaluation performed on four real-life datasets has shown that search space pruning using the average cost measure is effective and that the projected database buffer technique improves performance. A case study on data from an e-learning system has shown that useful cost-effective patterns are discovered. Those patterns can provide insights to students and teachers about how to use learning materials more efficiently.

There are several opportunities for future work. From an algorithmic perspective, we are interested in exploring other optimizations that could improve performance. Moreover, we intend to integrate the concept of cost in other pattern discovery tasks such as itemsets and sequential rules, and add other constraints to the proposed model. A more detailed study on how cost-efficient patterns can be used in e-learning and other applications can also be done.

References

- [1] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, *Data mining and knowledge discovery* 8 (1) (2004) 53–87.
- [2] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, B. Le, A survey of itemset mining, *WIREs Data Mining and Knowledge Discovery* (2017).
- [3] P. Fournier-Viger, J. C.-W. Lin, U. R. Kiran, Y.-S. Koh, A survey of sequential pattern mining, *Data Science and Pattern Recognition* 1 (1) (2017) 54–77.
- [4] R. Agrawal, R. Srikant, Mining sequential patterns, in: *Proc. of the 11th Intern. Conf. on Data Engineering*, IEEE, 1995, pp. 3–14.
- [5] R. Srikant, R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, in: *Proc. of the 5th Intern. Conf. on Extending Database Technology*, Springer, 1996, pp. 1–17.
- [6] M. J. Zaki, Spade: An efficient algorithm for mining frequent sequences, *Machine learning* 42 (1-2) (2001) 31–60.
- [7] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu, Freespan: frequent pattern-projected sequential pattern mining, in: *Proc. of the 6th ACM SIGKDD Intern. Conf. on Knowledge discovery and data mining*, ACM, 2000, pp. 355–359.
- [8] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: The prefixspan approach, *IEEE Transactions on Knowledge and Data Engineering* 16 (11) (2004) 1424–1440.
- [9] P. Fournier-Viger, A. Gomariz, M. Campos, R. Thomas, Fast vertical mining of sequential patterns using co-occurrence information, in: *Proc. of the 18th Pacific-Asia Conference on Knowledge Discovery in Data, Part I*, 2014, pp. 40–52.
- [10] P. Fournier-Viger, C. Wu, A. Gomariz, V. S. Tseng, VMSP: efficient vertical mining of maximal sequential patterns, in: *Proc. of the 27th Canadian conference on Artificial Intelligence*, 2014, pp. 83–94.
- [11] B. Le, H. V. Duong, T. C. Truong, P. Fournier-Viger, Fclossm, fgensm: two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy, *Knowl. Inf. Syst.* 53 (1) (2017) 71–107.
- [12] J. Yin, Z. Zheng, L. Cao, Uspan: an efficient algorithm for mining high utility sequential patterns, in: *Proc. of the 18th ACM SIGKDD Intern. Conf. on Knowledge discovery and data mining*, ACM, 2012, pp. 660–668.
- [13] M. Zihayat, H. Davoudi, A. An, Mining significant high utility gene regulation sequential patterns, *BMC Systems Biology* 11 (6) (2017) 109:1–109:14.
- [14] O. K. Alkan, P. Karagoz, Crom and huspect: Improving efficiency of high utility sequential pattern extraction, in: *Proc. of the 32nd IEEE Intern. Conf. on Data Engineering*, 2016, pp. 1472–1473.
- [15] T. Truong-Chi, P. Fournier-Viger, A survey of high utility sequential pattern mining, in: *High-Utility Pattern Mining*, Springer, 2019, pp. 97–129.
- [16] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, et al., Knowledge discovery and data mining: Towards a unifying framework., in: *Proc. of the 2nd ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data mining*, Vol. 96, 1996, pp. 82–88.
- [17] C. C. Aggarwal, *Data mining: the textbook*, Springer, 2015.
- [18] O. Maimon, L. Rokach, Introduction to knowledge discovery and data mining, in: *Data Mining and Knowledge Discovery Handbook*, Springer, 2009, pp. 1–15.
- [19] P. K. Novak, N. Lavrač, G. I. Webb, Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining, *Journal of Machine Learning Research* 10 (2009) 377–403.
- [20] C. Zhang, C. Liu, X. Zhang, G. Almpandis, An up-to-date comparison of state-of-the-art classification algorithms, *Expert Systems with Applications* 82 (1) (2017) 128–150.
- [21] F. Herrera, C. J. Carmona, P. González, M. J. Del Jesus, An overview on subgroup discovery: foundations and applications, *Knowledge and information systems* 29 (3) (2011) 495–525.
- [22] A. Zimmermann, S. Nijssen, Supervised pattern mining and applications to classification, in: *Frequent pattern mining*, Springer, 2014, pp. 425–442.
- [23] N. Anwar, H. L. H. S. Warnars, H. E. P. Sanchez, Survey of emerging patterns, in: *Proc. of the 2017 IEEE Intern. Conf. on Cybernetics and Computational Intelligence*, IEEE, 2017, pp. 11–18.
- [24] L. Venturini, P. Garza, D. Apiletti, Bac: A bagged associative classifier for big data frameworks, in: M. Ivanović, B. Thalheim, B. Catania, K.-D. Schewe, M. Kirikova, P. Šaloun, A. Dahanayake, T. Cerquitelli, E. Baralis, P. Michiardi (Eds.), *Proc. of the 20th East-European Conf. on Advances in Databases and Information Systems*, Springer International Publishing, Cham, 2016, pp. 137–146.
- [25] J. M. Luna, Pattern mining: current status and emerging topics, *Progress in Artificial Intelligence* 5 (2016) 165–170.
- [26] P. Fournier-Viger, J. C.-W. Lin, T. Truong-Chi, R. Nkambou, A survey of high utility itemset mining, in: *High-Utility Pattern Mining*, Springer, 2019, pp. 1–45.
- [27] P. Fournier-Viger, Y. Zhang, J. C.-W. Lin, H. Fujita, Y. S. Koh, Mining local and peak high utility itemsets, *Information Sciences* 481 (2019) 344–367.
- [28] C.-W. Wu, Y.-F. Lin, P. S. Yu, V. S. Tseng, Mining high utility episodes in complex event sequences, in: *Proc. of the 19th ACM SIGKDD Intern. Conf. on Knowledge discovery and data mining*, 2013.
- [29] P. Fournier-Viger, Z. Li, C.-W. Lin, R. U. Kiran, H. Fujita, Efficient algorithms to identify periodic patterns in multiple sequences, *Information Sciences* 489 (2019) 205–226.
- [30] C. Jiang, F. Coenen, M. Zito, A survey of frequent subgraph mining algorithms, *Knowledge Eng. Review* 28 (2013) 75–105.
- [31] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, J. C. S. Lui, Diversified temporal subgraph pattern mining, in: *Proc. of the 22nd ACM SIGKDD Intern. Conf. on Knowledge discovery and data mining*, 2016.
- [32] P. Fournier-Viger, C.-W. Wu, V. S. Tseng, Mining maximal sequential patterns without candidate maintenance, in: H. Motoda, Z. Wu, L. Cao, O. Zaiane, M. Yao, W. Wang (Eds.), *Proc. of the 9th Intern. Conf. on Advanced Data Mining and Applications*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 169–180.
- [33] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, A novel approach for mining high-utility sequential patterns in sequence databases, *ETRI journal* 32 (5) (2010) 676–686.

- [34] G.-C. Lan, T.-P. Hong, V. S. Tseng, S.-L. Wang, Applying the maximum utility measure in high utility sequential pattern mining, *Expert Systems with Applications* 41 (11) (2014) 5071–5081.
- [35] H. Ryang, U. Yun, High utility pattern mining over data streams with sliding window technique, *Expert Syst. Appl.* 57 (2016) 214–231.
- [36] U. Yun, H. Ryang, G. Lee, H. Fujita, An efficient algorithm for mining high utility patterns from incremental databases with one database scan, *Knowl.-Based Syst.* 124 (2017) 188–206.
- [37] W. Gan, C.-W. Lin, H. Chao, T.-P. Hong, P. S. Yu, Coupmm: Correlated utility-based pattern mining, 2018 IEEE International Conference on Big Data (2018) 2607–2616.
- [38] O. K. Alkan, P. Senkul, Crom and huspext: Improving efficiency of high utility sequential pattern extraction, *IEEE Transactions on Knowledge and Data Engineering* 27 (2015) 2645–2657.
- [39] J.-Z. Wang, J.-L. Huang, Y.-C. Chen, On efficiently mining high utility sequential patterns, *Knowledge and Information Systems* 49 (2015) 597–627.
- [40] W. M. Van der Aalst, A. Weijters, Process mining: a research agenda, *Computers in Industry* 55 (3) (2004) 231–244.
- [41] W. M. Van Der Aalst, M. La Rosa, F. M. Santoro, *Business process management*, Springer, 2016.
- [42] A. Bogarín, R. Cerezo, C. Romero, A survey on educational process mining, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8 (1) (2018).
- [43] R. S. Mans, W. M. van der Aalst, R. J. Vanwersch, A. J. Moleman, Process mining in healthcare: Data challenges when answering frequently posed questions, in: *Proc. of the 2012 Joint Workshop on Process-Oriented Information Systems and Knowledge Representation in Health Care*, Springer, 2013, pp. 140–153.
- [44] J. Garcia-Algarra, Subgroup discovery in process mining, in: *Proc. of the 20th Intern. Conf. on Business Information Systems*, Vol. 288, Springer, 2017, p. 237.
- [45] J. Ranjan, K. Malik, Effective educational process: a data-mining approach, *Vine* 37 (4) (2007) 502–515.
- [46] F. Mannhardt, D. Blinde, Analyzing the trajectories of patients with sepsis using process mining, *CEUR*, 2017, pp. 72–80.
- [47] B. Dalmas, P. Fournier-Viger, S. Norre, Twinkle: A constrained sequential rule mining algorithm for event logs, in: *Proc. 9th Intern. KES Conf. on Intelligent Decision Technologies*, Elsevier, 2017, pp. 205–214.
- [48] L. K. Poon, S.-C. Kong, M. Y. Wong, T. S. Yau, Mining sequential patterns of students’ access on learning management system, in: *Proc. 2nd Intern. Conf. on Data Mining and Big Data*, Springer, 2017, pp. 191–198.
- [49] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, R. Thomas, A survey of sequential pattern mining, *Data Science and Pattern Recognition* 1 (1) (2017) 54–77.
- [50] J. Pei, J. Han, W. Wang, Mining sequential patterns with constraints in large databases, in: *Proc. of the 11th Intern. Conf. on Information and knowledge management*, ACM, 2002, pp. 18–25.
- [51] G. V. Glass, K. D. Hopkins, *Statistical methods in education and psychology*, Pearson, 1996.
- [52] Q.-H. Duong, P. Fournier-Viger, H. Ramampiaro, K. Nørnvåg, T.-L. Dam, Efficient high utility itemset mining using buffered utility-lists, *Applied Intelligence* 48 (7) (2018) 1859–1877.
- [53] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, V. S. Tseng, Spmf: a java open-source pattern mining library, *Journal of Machine Learning Research* 15 (1) (2014) 3389–3393.
- [54] S. Zida, P. Fournier-Viger, C.-W. Wu, J. C.-W. Lin, V. S. Tseng, Efficient mining of high-utility sequential rules, in: *Proc. of the 11th Intern. Conf. on Machine Learning and Data Mining in Pattern Recognition*, Springer, 2015, pp. 157–171.
- [55] M. Vahdat, L. Oneto, D. Anguita, M. Funk, M. Rauterberg, A learning analytics approach to correlate the academic achievements of students with interaction data from an educational simulator, in: *Design for Teaching and Learning in a Networked World*, Springer, 2015, pp. 352–366.