Front. Comput. Sci.
https://doi.org/10.1007/s11704-016-6245-4

RESEARCH ARTICLE

CLS-Miner: efficient and effective closed high-utility itemset mining

Thu-Lan DAM[1,2], Kenli LI (✉)[1,3,4], Philippe FOURNIER-VIGER[5], Quang-Huy DUONG[6]

1 College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
2 Faculty of Information Technology, Hanoi University of Industry, Hanoi, Vietnam
3 CIC of HPC, National University of Defense Technology, Changsha 410073, China
4 National Supercomputing Center in Changsha, Changsha 410082, China
5 School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China
6 Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway

Abstract High-utility itemset mining (HUIM) is a popular data mining task with applications in numerous domains. However, traditional HUIM algorithms often produce a very large set of high-utility itemsets (HUIs). As a result, analyzing HUIs can be very time consuming for users. Moreover, a large set of HUIs also makes HUIM algorithms less efficient in terms of execution time and memory consumption. To address this problem, closed high-utility itemsets (CHUIs), concise and lossless representations of all HUIs, were proposed recently. Although mining CHUIs is useful and desirable, it remains a computationally expensive task. This is because current algorithms often generate a huge number of candidate itemsets and are unable to prune the search space effectively. In this paper, we address these issues by proposing a novel algorithm called CLS-Miner. The proposed algorithm utilizes the utility-list structure to directly compute the utilities of itemsets without producing candidates. It also introduces three novel strategies to reduce the search space, namely chain-estimated utility co-occurrence pruning, lower branch pruning, and pruning by coverage. Moreover, an effective method for checking whether an itemset is a subset of another itemset is introduced to further reduce the time required for discovering CHUIs. To evaluate the performance of the proposed algorithm and its novel strategies, extensive experiments have been conducted on six benchmark datasets having various characteristics. Results show that the proposed strategies are highly efficient and effective, that the proposed CLS-Miner algorithm outperforms the current state-of-the-art CHUD and CHUI-Miner algorithms, and that CLS-Miner scales linearly.

Keywords utility mining, high-utility itemset mining, closed itemset mining, closed high-utility itemset mining

1 Introduction

Frequent itemset mining (FIM) [1] is a popular knowledge discovery task. The goal of FIM is to find all frequent itemsets in a transaction database. A frequent itemset is a set of items that appears in at least *minsup* transactions, where *minsup* is a parameter set by the user. FIM has been the subject of many studies, and remains to this day a very active research area [2–7]. However, it assumes that all items in a database are equally important (e.g., in terms of profit, importance, or weight) and that items may not appear more than once in each transaction. However, these assumptions often do not hold in reality. In practice, retailers may be more interested in find-

Received April 28, 2016; accepted November 29, 2016

E-mail: lkl@hnu.edu.cn

ing the itemsets that yield a high profit than in discovering the itemsets that appear frequently. To address these limitations, the task of FIM has been redefined as the task of high-utility itemset mining (HUIM). In HUIM, items can appear more than once in each transaction and a weight is assigned to each item to indicate its relative importance to the user (e.g., in terms of unit profit). The goal of HUIM is to discover all high-utility itemsets (HUIs) in a transaction database. A set of items is said to be a HUI if its utility (e.g., the profit that it yields in a database) is not less than a minimum utility threshold *minutil* set by the user.

In recent years, HUIM [8–14] has become a popular research area, because it has many applications in several domains such as cross-marketing, click stream analysis, and biomedicine [9, 10, 15–17]. HUIM is widely recognized as more difficult than FIM because the powerful downward closure property used to prune the search space in FIM does not hold in HUIM. This latter property states that the support of an itemset is anti-monotonic, i.e., supersets of an infrequent itemset are infrequent and subsets of a frequent itemset are frequent. However, in HUIM, a HUI may have supersets and subsets having lower, equal, or higher utilities. Thus, techniques to prune the search space developed in FIM cannot be used in HUIM directly.

Many efficient HUIM algorithms have been proposed [8–13, 18, 19]. To prune the search space, these algorithms utilize measures that overestimate the utility of itemsets and are monotonic [8, 10]. Even though these algorithms have many applications, a major drawback is that the set of HUIs can be very large, depending on how the minimum utility parameter is set by the user. In general, when a HUIM algorithm generates more HUIs, its execution time and memory consumption also increase greatly. Furthermore, analyzing a large number of HUIs produced by a HUIM algorithm is a difficult and time-consuming task for users.

To address this issue, a compact and lossless representation of HUIs, called closed high-utility itemsets (CHUIs) [20], was proposed. This representation is inspired by the concept of closed patterns [5, 21–23], which was introduced in FIM. An itemset is said to be a CHUI if (1) its utility is not less than a minimum utility threshold specified by the user, and if (2) it has no supersets appearing in the same transactions [20]. The set of CHUIs is interesting because it can be several orders of magnitude smaller than the set of all HUIs, and it allows all HUIs to be derived without scanning the database (it is lossless). Moreover, CHUIs also provide meaningful information to decision makers because they are the largest HUIs that are common to groups of customers [20].

Although mining CHUIs is desirable for the above reasons, it remains a computationally difficult task, and it is challenging to design a closed high-utility itemset mining (CHUIM) algorithm that is both efficient in terms of runtime and memory usage [20]. To design an efficient CHUIM algorithm, one must correctly incorporate techniques from closed FIM with techniques used in HUIM, to reduce the search space effectively, ensuring that no CHUIs are missed. CHUD [20] is the first CHUIM algorithm. However, a major limitation of CHUD is that it is a two-phase algorithm, which can generate a large number of candidates[1)] and repeatedly scans the database.

To address this limitation of two-phase algorithms, one-phase algorithms have been proposed recently and have been shown to outperform previous two-phase HUIM algorithms [9, 11, 24, 25]. These algorithms employ a novel structure called a utility-list [9, 11]. Using this structure, the utility of an itemset can be quickly calculated without scanning the original database by making join operations with utility-lists of smaller itemsets. Inspired by these algorithms, a one-phase algorithm called CHUI-Miner (Closed$^+$ High Utility Itemset Miner) [26] has been recently proposed for CHUIM. It employs a structure called (extended utility-list (EU-List) to maintain information about the utilities of itemsets in transactions. Furthermore, it adopts a divide-and-conquer methodology to efficiently mine the complete set of CHUIs. However, even with the improvements introduced in CHUI-Miner, the task of CHUIM remains computationally expensive for the following reasons.

(1) First, although CHUI-Miner [26] discovers CHUIs using a single phase and does not generate candidates as per the definition of the two-phase model, CHUI-Miner explores the search space by generating itemsets, and a costly join operation has to be repeatedly performed to evaluate the utility of each itemset. Thus, it is desirable to design more efficient pruning strategies that could reduce the number of utility-list joins that are performed.

(2) Second, CHUI-Miner [26] prunes the search space using two upper-bounds on the utilities of itemsets called the transaction-weighted utilization (TWU) [8] and the remaining utility [9]. An important limitation of using

[1)] It is important to note that the term "candidate" has different meanings in HUIM and FIM. In FIM, an algorithm is said to not generate candidates if it only considers patterns that exist in a database [3, 4]. In HUIM, an algorithm is said to not generate candidates if it does not mine HUIs in two phases, i.e., by first identifying a set of potential HUIs using overestimations, and then calculating their exact utilities by scanning the database

the TWU upper-bound is that it is loose, and thus numerous candidates need to be considered to find the final set of CHUIs.

(3) Third, a drawback of the search space pruning strategy based on the remaining utility upper-bound used by CHUI-Miner is that it can only be applied to an itemset after its utility-list has been fully constructed. However, constructing a utility-list is an expensive operation [9, 11]. To improve the efficiency of CHUIM, it is thus desirable to design pruning strategies that can be applied during or before utility-list construction.

(4) Fourth, efficient techniques must be designed to avoid considering as many non-closed itemsets as possible. In particular, two key operations for closed pattern mining are subsumption checks and closure computations [5, 22, 23]. Those operations need to be implemented efficiently.

To address the need for a more efficient CHUIM algorithm, this paper proposes a novel algorithm called CLS-Miner. This algorithm introduces several novel ideas to discover CHUIs efficiently by addressing the four above limitations. The main contributions of this work are summarized as follows.

(1) Three efficient strategies for pruning the search space are proposed. The first strategy is called chain-estimated utility co-occurrence pruning (Chain-EUCP). It uses the estimated utility of item co-occurrences to determine whether an itemset and its extensions should be pruned. The second strategy is called lower branch pruning (LBP). This strategy reduces the search space using a novel upper-bound on the utilities of transitive extensions of an itemset. These two strategies allow candidates to be eliminated without fully constructing their utility-lists. The last strategy is based on a new concept called coverage that is inspired by the definition of a closed itemset. The coverage can be used to prune low-utility itemsets and also to quickly calculate the closure of itemsets. The three proposed pruning strategies can greatly reduce the number of join operations for constructing utility-lists and, thus, prune a large part of the search space.

(2) An efficient pre-check containing method for checking whether an itemset X contains another itemset Y is also proposed. This method is used to perform subsumption checks and closure computations. These two operations are essential in closed pattern mining. As shown in the experimental evaluation section of this paper, the novel pre-check method is effective and can greatly reduce the runtime of these two operations.

(3) A novel algorithm called CLS-Miner is proposed to efficiently mine CHUIs. The CLS-Miner algorithm utilizes the utility-list structure and integrates the three proposed pruning strategies and the fast pre-check method.

(4) An extensive performance evaluation of the proposed algorithm and its strategies is performed on both real-life and synthetic datasets. The performance of CLS-Miner is compared with the performance of the state-of-the-art CHUD and CHUI-Miner algorithms for CHUIM. Results show that the proposed strategies are effective for reducing the search space, and that the CLS-Miner algorithm is highly efficient. The proposed algorithm also has excellent scalability.

The rest of this paper is organized as follows. Section 2 reviews studies related to HUIM and CHUIM. Preliminaries and the problem definition are introduced in Section 3. Section 4 presents the three novel strategies and pre-check method. Section 5 presents the CLS-Miner algorithm, and gives a detailed example illustrating how the algorithm is applied. Experimental results are reported in Section 6. Finally, a conclusion is drawn, and future work is discussed in Section 7.

2 Related work

This paper presents a new algorithm for CHUIM. Thus, this section briefly reviews studies related to HUIM and CHUIM.

2.1 HUIM

In real life, customer transactions generally contain information about the purchase quantities of items, and items may have different unit profits. Traditional algorithms for FIM ignore this information. To address this limitation, the task of HUIM was proposed [27]. In HUIM, the utility of an itemset may be equal, less than, or greater than the utility of any of its subsets. Thus, if an algorithm prunes supersets of low-utility itemsets to reduce the search space, it may produce an incomplete set of HUIs. To address this issue, the TWU measure [8] was introduced. It is an upper-bound on the utility of itemsets, and restores the downward closure property. According to the TWU model, all supersets of an itemset having a TWU lower than the minimum utility threshold also have a TWU lower than that threshold. However, a disadvantage of using the TWU is that it is a loose upper-bound on the utility

of itemsets and, thus, a huge number of candidates still need to be considered to find the final set of HUIs.

Numerous HUIM algorithms discover HUIs in two phases using the TWU model such as Two-Phase [8], UP-Growth and UP-Growth+ [10], PB [28], and BAHUI [12]. These algorithms first generate a set of candidate HUIs by overestimating their utilities using the TWU measure (phase 1). Then, these algorithms perform additional database scans to calculate the exact utilities of candidates and keep only those that are HUIs (phase 2). Despite having introduced new techniques to discover HUIs, an important drawback of two-phase algorithms is that they can generate a huge number of candidates because the TWU is a loose upper-bound on the utility of itemsets.

Recently, algorithms have been proposed to mine HUIs using a single phase to avoid maintaining a large number of candidates in memory before calculating their exact utility. The HUI-Miner algorithm [9] introduced a new data structure called utility-list to maintain information about the utilities of itemsets in transactions. Once HUI-Miner has constructed the utility-list of each single item, it can create the utility-list of any larger itemset by joining the utility-lists of some of its subsets, without scanning the database. HUI-Miner can find all HUIs and their exact utilities using the utility-list structure. HUI-Miner [9] was shown to outperform previous algorithms. However, HUI-Miner still has to perform a costly join operation to obtain the utility-list of each itemset generated by its search procedure. To reduce the number of join operations, the FHM algorithm [11] was introduced. It integrates a novel strategy for pruning the search space using information about itemset co-occurrences. It was shown that this strategy can greatly reduce the number of join operations, and that FHM [11] can be up to six times faster than HUI-Miner.

2.2 CHUIM

Although the discovery of HUIs is useful and has many applications, a drawback of traditional HUIM algorithms is that they can produce a huge number of patterns. For users, analyzing a huge number of patterns can be difficult and time consuming. Moreover, as more patterns are discovered, the runtimes of HUIM algorithms become longer and more memory is consumed [20].

In FIM, a popular solution to this problem is to discover concise representations of patterns that are small sets of patterns that summarize all other patterns by eliminating redundant itemsets (itemsets that can be derived from the other itemsets). For example, several concise representations of frequent itemsets have been proposed such as closed itemsets [21, 22], maximal itemsets [29, 30], and generator itemsets [31]. It was demonstrated that these representations can be orders of magnitude smaller than the set of all frequent patterns, thus providing more concise information to users. Moreover, it was shown that mining these representations is often faster than discovering all frequent itemsets.

To obtain similar benefits in HUIM, researchers have developed concise representations of HUIs [20, 26, 32, 33]. Designing a concise representation of HUIs is more challenging than developing a concise representation of frequent itemsets as the utility measure is neither monotonic nor anti-monotonic [20]. Thus, concise representations in FIM need to be adapted to the context of HUIM, and novel techniques need to be designed to efficiently discover these representations in databases. Three main concise representations of HUIs have been proposed, which have different characteristics and provide different information to users.

- A maximal HUI [33] is a HUI that has no supersets that are HUIs. The representation of maximal HUIs provides the benefit of being a very compact representation of all HUIs. However, it has the drawback of being lossy, i.e., a large amount of information about HUIs can be lost.
- A generator of a HUI [32] is the smallest set of items common to a group of customers that have bought a HUI. The representation of generators of HUIs is useful to discover small sets of items that characterize groups of customers. However, it is also lossy.
- A closed+ HUI [20, 26] is a HUI that has no supersets having the same support, and annotated with its utility unit array. The advantages of CHUIs are that they are very compact and lossless (unlike the two previous representations). Thus, information about every other HUI can be recovered from this representation without scanning the database.

In this paper, we are interested in closed+ HUIs because they are the most popular concise representation of HUIs and are lossless. The first proposed algorithm for mining CHUIs was CHUD (Closed$^+$ High Utility itemset Discovery) [20]. It is a depth-first search algorithm that extends the DCI-Closed [22] algorithm, which is one of the fastest algorithms for mining frequent closed itemsets. DCI-Closed [22] utilizes a bitwise vertical representation of the database. Moreover, it adopts a divide-and-conquer approach and visits itemsets in the search space in a specific order by traversing generator

itemsets to reach closed itemsets. Furthermore, DCI-Closed integrates an efficient mechanism to detect duplicate closed itemsets without maintaining previously found closed itemsets in main memory.

CHUD [20] is a two-phase algorithm. In the first phase, it applies a modified DCI-Closed procedure to discover candidate CHUIs. During this process, CHUD applies several effective strategies for reducing the number of candidates generated based on the properties of the utility measure. Similar to the DCI-Closed algorithm, CHUD utilizes the itemset–transaction id (Tid) set pair tree (IT-Tree) structure [22] to explore the search space of itemsets. In an IT-Tree, each node $N(X)$ contains an itemset X, its Tid set $TidSet(X)$, and two lists of items called $PreSet(X)$ and $PostSet(X)$. The CHUD algorithm recursively explores the IT-Tree and stops when all candidate CHUIs have been found. A key difference between DCI-Closed and CHUD, is that this latter stores an estimated utility value $EstU(X)$ in each node $N(X)$ of the IT-Tree. This value represents an upper-bound on the utility of the itemset X and its supersets, and is used for reducing the search space. A data structure called transaction utility table (TU-Table) [20] is also adopted, which stores the transaction utilities of all transactions. The TU-Table allows the estimated utility of any itemset X to be calculated efficiently using its Tid set and without scanning the database.

In the second phase, CHUD calculates the exact utility of each candidate itemset X found in phase 1 to filter those that are not HUIs. At the same time, the CHUD algorithm creates a structure called utility unit array for each CHUI. This structure ensures that the set of CHUIs is a lossless representation of all HUIs (that it can be used to derive all other HUIs and their exact utilities). The CHUD algorithm utilizes four strategies to reduce the search space. The first is called discarding global unpromising items (DGU). It consists of only considering promising items for generating candidates and to remove the utilities of unpromising items from the global TU-Table. The three other strategies are called removing the exact utilities of items from the global TU-Table (REG), removing the minimum itemset utilities from the local TU-Tables (RML), and discarding candidates having a maximum item utility that is less than the minimum utility threshold (DCM). Although CHUD introduces a novel tree structure and several candidate pruning strategies, it inherits the limitations of two-phase algorithms. This considerably degrades its performance both in terms of runtime and memory consumption.

Recently, a one-phase algorithm called CHUI-Miner [26] was proposed for mining CHUIs more efficiently. CHUI-Miner integrates techniques from closed itemset mining [21, 22] to only discover closed patterns. It utilizes an EU-List structure to store information about the utility of itemsets and adopts a divide-and-conquer approach to mine CHUIs without producing candidates. CHUI-Miner is the first one-phase algorithm for mining CHUIs. However, CHUI-Miner only prunes the search space using the TWU [8] and the remaining utility upper-bounds [9]. As a result, CHUI-Miner can still consider a huge number of itemsets from the search space. Moreover, CHUI-Miner repeatedly applies a EU-list construction operation that is very costly.

Although CHUIM has many applications, discovering CHUIs remains a computationally expensive task. Therefore, there is a need for more efficient algorithms [26]. To address this issue, this paper proposes a novel algorithm called CLS-Miner to mine CHUIs efficiently. It is a one-phase algorithm (unlike CHUD), which relies on the utility-list structure to discover CHUIs (similarly to CHUI-Miner). However, there are some key differences between CLS-Miner and CHUI-Miner. The proposed algorithm integrates three novel search space pruning strategies: (1) Chain-EUCP, (2) LBP, and (3) pruning by coverage. These novel strategies can prune itemsets in the search space before their utility-lists are constructed, and thus greatly reduce the cost of mining CHUIs. Furthermore, an efficient pre-check containing method is also proposed to quickly determine whether an itemset is a subset of another itemset. This method is used to optimize the operations of closure computations and subsumption checks typically performed by closed pattern mining algorithms. It will be shown in the experimental evaluation of this paper that the proposed pre-check method considerably reduces the time for discovering CHUIs.

3 Preliminaries and problem definition

Before presenting the proposed algorithms and its novel strategies, it is necessary to introduce preliminaries related to HUIM, closed itemset mining, and to formally define the problem of CHUIM. This section introduces these definitions.

3.1 Problem definition

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. A positive number $p(i_j)$ is associated to each item $i_j \in I$, called the external utility of i_j. It represents the relative importance of item i_j to the user (e.g., the unit profit of i_j). Let D be a transaction database containing a set of n transactions $D = \{T_1, T_2, \ldots, T_n\}$ such that $T_d \subseteq I$ ($1 \leqslant d \leqslant n$), and each transaction T_d has a unique

identifier d called its Tid. Moreover, given a transaction T_d and an item i_j, let $q(i, T_d)$ be the internal utility (e.g., the purchase quantity) of item i in T_d. An itemset X is a set of l distinct items $X = \{i_1, i_2, \ldots, i_l\}$ such that $X \subseteq I$. An itemset containing l items is said to be of length l, and to be an l-itemset. In the rest of this paper, for the sake of brevity, each itemset will be denoted by the concatenation of its items. For example, the itemset $\{x, y, z\}$ will be denoted as xyz. The union of two itemsets X and Y will be denoted as XY or $X \cup Y$.

Example 1 Table 1 presents a transaction database D containing five transactions (T_1, T_2, T_3, T_4, and T_5), which will be used as a running example. In this database, the set of items is $I = \{a, b, c, d, e, f, g\}$. The external utilities (e.g., unit profits) of these items are presented in Table 2. The items a, b, c, d, e, f, and g have external utilities of 5, 2, 1, 2, 3, 1, and 1, respectively. The itemset ac appears in transactions T_1, T_2, and T_3. The items a, c, e, and g have internal utilities (purchase quantities) of 2, 6, 2, and 5, respectively, in transaction T_2.

Table 1 Example transaction database

Tid	Transaction	TU
T_1	$(a, 1), (c, 1), (d, 1)$	8
T_2	$(a, 2), (c, 6), (e, 2), (g, 5)$	27
T_3	$(a, 1), (b, 2), (c, 1), (d, 6), (e, 1), (f, 5)$	30
T_4	$(b, 4), (c, 3), (d, 3), (e, 1)$	20
T_5	$(b, 2), (c, 2), (e, 1), (g, 2)$	11

Table 2 External utility values

Item	a	b	c	d	e	f	g
External utility	5	2	1	2	3	1	1

Definition 1 (Utility of an item in a transaction) The utility of an item i in a transaction T_d is defined as $u(i, T_d) = q(i, T_d) \times p(i)$, where $q(i, T_d)$ is the internal utility (purchase quantity) of item i in transaction T_d, and $p(i)$ is the external utility (unit profit) of item i.

For example, in Table 1, $u(a, T_1) = 1 \times 5 = 5$ and $u(c, T_1) = 1 \times 1 = 1$.

Definition 2 (Utility of an itemset in a transaction) The utility of an itemset X in a transaction T_d is denoted as $u(X, T_d)$ and defined as $u(X, T_d) = \sum_{i \in X} u(i, T_d)$.

For example, in Table 1, $u(ac, T_1) = 1 \times 5 + 1 \times 1 = 6$ and $u(ac, T_2) = 2 \times 5 + 6 \times 1 = 16$.

Definition 3 (Transaction utility and total utility) The utility of a transaction T_d is denoted as $TU(T_d)$, and is calculated as $TU(T_d) = u(T_d, T_d)$. The total utility of a database D is denoted as $TUD(D)$, and is calculated as $TUD(D) = \sum_{T_d \in D} TU(T_d, T_d)$.

For example, in Table 1, $TU(T_1) = 8$, $TU(T_2) = 27$, $TU(T_3) = 30$, $TU(T_4) = 20$, and $TU(T_5) = 11$. The total utility of database D is $TUD(D) = (TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) + TU(T_5)) = 8 + 27 + 30 + 20 + 11 = 96$.

Definition 4 (Utility and relative utility of an itemset) The utility of an itemset X in a database D is defined as $u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$. The relative utility of an itemset X in a database D is denoted as $ru(X)$ and is defined as $ru(X) = u(X)/TUD(D)$.

For example, in Table 1, $u(ac) = u(ac, T_1) + u(ac, T_2) + u(ac, T_3) = 6 + 16 + 6 = 28$. The relative utility of itemset ac is, thus, $ru(ac) = 28/96 = 0.29$.

Definition 5 (HUI) Let there be a user-specified minimum utility threshold $minutil$ ($0 < minutil < TUD(D)$). If the utility of an itemset X is no less than $minutil$, then X is said to be a HUI. Otherwise, X is said to be a low-utility itemset. The problem of HUIM consists of discovering all HUIs in a given transaction database.

An equivalent definition of the problem of HUIM is the following. Given a relative minimum utility threshold $r\_minutil = minutil/TUD(D)$, an itemset X is a HUI if and only if $ru(X) \geqslant r\_minutil$.

Having formally defined the problem of HUIM, the following paragraphs define the problem of CHUIM.

Definition 6 (Support and Tid set of an itemset) The Tid set of an itemset X is the set of Tids of transactions containing X, and is denoted by $TidSet(X)$. The *support* of an itemset X is denoted as $sup(X)$ and defined as $sup(X) = |TidSet(X)|$. In other words, the support of an itemset is the number of transactions where the itemset appears.

Definition 7 (Closed itemset) An itemset X is called a closed itemset if there exists no proper superset $Y \supset X$ in D such that $sup(X) = sup(Y)$ [21,22].

Definition 8 (CHUI) An itemset X is a *CHUI* if X is closed and its utility $u(X) \geqslant minutil$ [20,26].

Problem statement Let there be a database D where internal and external utilities are provided for all items. Given a user-specified minimum utility threshold $minutil$, the problem of CHUIM is to discover all closed itemsets in D having

utilities that are no less than the *minutil* threshold [20, 26].

Example 2 Consider the database of Table 1. Suppose that the parameter *minutil* is set to 30 (or, equivalently, that *r_minutil* = 31%). The set of HUIs for the running example is H = {*bd*:30, *ace*:31, *bcd*:34, *bce*:31, *bde*:36, *bcde*:40, and *abcdef*:30}, where the number beside each itemset indicates its utility. Among those itemsets, the CHUIs are CH = {*ace*:31, *bce*:31, *bcde*:40, and *abcdef*:30}.

In FIM, the downward closure property is used to reduce the search space. However, this property does not hold in HUIM, when considering the utility measure. To restore this property, the TWU measure was introduced and it is an upper-bound on the utility. The TWU measure is defined as follows [8].

Definition 9 The TWU [8] of an itemset X in a database D is denoted as $TWU(X)$ and defined as $TWU(X) = \sum_{T_d \in D \wedge X \subseteq T_d} TU(T_d)$.

Property 1 (Overestimation [8]) The TWU of an itemset X is no less than its utility, that is $TWU(X) \geqslant u(X)$.

For example, the transaction utilities of transactions T_1, T_2, and T_3 of the running example are 8, 27, and 30, respectively. The TWU of item a is $TWU(a) = TU(T_1)+TU(T_2)+TU(T_3)$ = 8 + 27 + 30 = 65. The following property of the TWU is commonly used to prune the search space in HUIM.

Property 2 (Pruning using the TWU [8]) Let there be an itemset X. If $TWU(X) < minutil$, then X and its supersets are low-utility itemsets.

For example, the TWU of item f is $TWU(f) = TU(T_3)$ = 5 + 4 + 1 + 12 + 3 + 5 = 30. Tables 3 and 4 show the transaction utilities of all transactions in D and the TWU values of all items.

Table 3 Transaction utilities for the running example

Tid	T_1	T_2	T_3	T_4	T_5
TU	8	27	30	20	11

Table 4 TWU values of items for the running example

Item name	*a*	*b*	*c*	*d*	*e*	*f*	*g*
TWU	65	61	96	58	88	30	38

3.2 Utility-list structure

The proposed algorithm relies on the utility-list structure of the HUI-Miner algorithm [9] to mine HUIs in a single phase. Therefore, this subsection introduces important definitions related to this structure and its key properties. The utility-list structure was proposed in HUI-Miner to avoid the drawbacks of the previous two-phase algorithms: maintaining a large amount of candidates in memory and repeatedly scanning the database to calculate the utilities of itemsets.

Definition 10 (Utility-list [9]) Without loss of generality, let $\succ$ be a total order on items from I. The *utility-list* of an itemset X in a database D is denoted by $ul(X)$. It contains a tuple of the form $(tid, iutil, rutil)$ for each transaction T_{tid} containing X ($X \subseteq T_{tid}$). The *iutil* element of a tuple corresponding to a transaction T_{tid} stores the utility of X in T_{tid}, i.e., $u(X, T_{tid})$. The *rutil* element of a tuple stores the value $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$, called the remaining utility [9].

Example 3 The utility-list of item a is {(T_1, 5, 3)(T_2, 10, 17)(T_3, 5, 25)}. The utility-list of item e is {(T_2, 6, 5)(T_3, 3, 5)(T_4, 3, 0)}. The utility-list of itemset *ae* is {(T_2, 16, 5),(T_3, 8, 5)}.

Utility-lists have the two following important properties, which are used to calculate the utility of any itemset and to prune the search space.

Property 3 (Sum of *iutil* values [9]) Let there be an itemset X. The utility of X (denoted as $u(X)$) is equal to the sum of the *iutil* values in its utility-list $ul(X)$. If the sum of the *iutil* values in $ul(X)$ is higher than or equal to the *minutil* threshold, it follows that X is a HUI. Otherwise, it is a low-utility itemset [9].

Property 4 (Pruning using the sum of *iutil* and *rutil* values [9]) Let there be an itemset X. An itemset Y is said to be an *extension* of X if and only if Y can be obtained by appending an item y to X such that $y \succ i$, $\forall i \in X$. It can be demonstrated that any transitive extension of X can only have a utility that is less than or equal to the sum of *iutil* and *rutil* values in $ul(X)$. If the sum of *iutil* and *rutil* values in the utility-list of X is less than *minutil*, all transitive extensions of X are low-utility itemsets.

As proposed in the HUI-Miner algorithm [9], the utility-list of any itemset can be obtained by intersecting the utility-lists of some of its subsets. For example, let P, Px and Py be itemsets, such that Px and Py are extensions of P with items x and y, respectively. The utility-list of itemset Pxy is obtained by applying Algorithm 1 [9]. For each element in $ul(x)$, the algorithm checks whether there is an element corresponding to the same transaction in $ul(y)$. If there is one, a binary search is then performed in the utility-list of P to de-

termine whether there is also an element corresponding to the same transaction. Therefore, the complexity of this procedure is $O(m \log nz)$, where m, n, and z are the number of entries in $ul(x)$, $ul(y)$, and $ul(P)$, respectively.

Algorithm 1 The utility-list construction procedure

Input:
ul(P) : the utility-list of itemset *P*;
ul(Px): the utility-list of itemset *Px*;
ul(Py): the utility-list of itemset *Py*;
Output:
ul(Pxy): the utility-list of itemset *Pxy*;

```
1: ul(Pxy) = NULL;
2: for each (tuple ex ∈ ul(Px)) do
3:   if (∃ey ∈ ul(Py) and ex.tid==ey.tid) then
4:     if (ul(P) is not empty) then
5:       Search element e ∈ ul(P) such that e.tid = ex.tid;
6:       exy ⟵ (ex.tid; ex.iutil + ey.iutil − e.iutil; ey.rutil);
7:     else
8:       exy ⟵ (ex.tid; ex.iutil + ey.iutil; ey.rutil);
9:     end if
10:    ul(Pxy) ⟵ ul(Pxy) ∪ exy;
11:  end if
12: end for
13: return ul(Pxy);
```

3.3 Frequent closed itemset mining

As the proposed algorithm mines closed patterns, this subsection introduces the concept of a closed itemset and its key properties. These properties will be used in the proposed algorithm.

The task of mining frequent closed itemsets consists of discovering all closed itemsets having support values that are no less than a user-specified minimum support threshold *minsup* [22, 23, 30, 31, 34]. It has been shown in previous studies that the number of frequent closed itemsets is often much lower than the number of frequent itemsets. Thus, it is desirable to mine closed itemsets [20].

Most frequent closed itemset mining algorithms [22, 23, 30, 31, 34] discover closed itemsets by performing two steps: search space browsing and closure computation. Among these algorithms, DCI-Closed [22] is one of the fastest algorithms. A typical closed itemset mining algorithm browses the search space of itemsets by going from an equivalence class to another by computing the closure of the visited frequent itemsets. This process allows to quickly identify the maximal element (the closed itemset) in each equivalence class. Two itemsets are said to belong to the same equivalence class if they appear in exactly the same set of transactions. Some key definitions related to closed itemsets are the following [22].

Definition 11 (Closure of an itemset [22]) The closure of an itemset X, denoted as $C(X)$, is the largest set Y such that $X \subseteq Y$ and $sup(X) = sup(Y)$. An alternative and equivalent definition is $C(X) = \bigcap_{d \in TidSet(X)} T_d$. An itemset X is called a closed itemset, if $X = C(X)$.

Definition 12 (Generator [22]) A generator is an itemset of the form $X = Y \cup i$, where Y is a closed itemset and $i \notin Y$. A generator X is said to be order preserving if $C(X) = Y \cup i$ or $i \prec C(Y \cup i) \backslash Y \cup i$.

To browse the search space of itemsets and find the closed itemsets, several algorithms start from the minimal elements of each equivalence class (the generators) [22]. As soon as a generator is found, its closure is computed. Then, new generators are built from that closed itemset by appending single items to that itemset, and the process is repeated with each of these generators to find other closed itemsets. As a closed itemset is the maximal element of its equivalence classes, this strategy guarantees that jumping from an equivalence class to another will not miss any closed itemsets. However, several generators have the same closure. Thus, it is necessary to utilize a mechanism to avoid generating the same closed itemset more than once. A solution to this problem is to use the following property [22].

Property 5 ([22]) Given two itemsets Y and X, if $Y \subset X$ and $supp(X) = supp(Y)$, then $C(X) = C(Y)$.

Therefore, if a generator Y is found, having the same support as an already-discovered closed itemset X and $Y \subset X$, it can be concluded that the closure of Y is X. In this case, it is said that X subsumes Y, and it becomes unnecessary to compute the closure of Y. Otherwise, the closure of Y should be computed to obtain a new closed itemset.

To compute the closure of a generator X, the following property can be employed [22].

Property 6 ([22]) Given an itemset X and an item $i \in I$, $TidSet(X) \subseteq TidSet(i) \Leftrightarrow i \in C(X)$.

By the above property, if $TidSet(X) \subseteq TidSet(i)$, it follows that i belongs to the closure of X. Therefore, by performing this inclusion check for all the items in I not included in X, the closure $C(X)$ of any itemset X can be computed.

Having presented the problem of HUIM, CHUIM, and important properties related to closed itemset mining, the following section introduces the proposed techniques.

4 Proposed strategies and pre-check method

To design an efficient CHUIM algorithm, three key challenges must be addressed.

- First, it is necessary to design effective search space pruning techniques because the search space in itemset mining tasks is very large. For example, if there are 1000 distinct items in a database, there are $2^{1000}-1$ possible itemsets (by excluding the empty set). To design a HUIM algorithm, it is necessary to design search space pruning techniques that rely on properties of the utility measure [8–13].
- Second, efficient techniques must be designed to avoid considering as many non-closed itemsets as possible. As mentioned in the previous section, most closed pattern mining algorithms perform two key operations: subsumption checks and closure computations [5, 22, 23].
  These two operations are repeatedly performed by most closed itemset mining algorithms to prune the search space of non-closed itemsets. For this reason, it is crucial to implement them efficiently.
- Third, an efficient CHUIM algorithm should calculate the utility of itemsets efficiently. To avoid generating a huge number of candidates and directly calculate the utility of itemsets, many recent state-of-the-art HUIM algorithms utilize the utility-list structure. However, a costly join operation needs to be performed to build utility-lists of l itemsets ($l > 1$). Therefore, an important question is how to reduce the cost of join operations, or avoid performing it when possible [11].

This section describes how these challenges are addressed in the proposed CLS-Miner algorithm. The designed algorithm introduces three novel search space pruning strategies to prune low-utility itemsets without fully constructing their utility-lists. These strategies are presented in Sections 4.1, 4.2, and 4.3, respectively. Moreover, an efficient pre-check method is introduced in Section 4.4 to optimize the operations of subsumption checking and closure computation to identify closed itemsets efficiently. Then, in Section 5, we present the proposed algorithm that combines all the proposed strategies and the pre-check method.

4.1 Chain-EUCP strategy

The first proposed pruning strategy is called Chain-EUCP. It relies on a structure called the Estimated Utility Co-occurrence Structure (EUCS) [11] defined as follows.

Definition 13 (EUCS [11]) Let I^* be the set of all items having a TWU no less than *minutil* in a database D. The EUCS of database D is denoted as $EUCS_D$ and defined as $EUCS_D = \{(a; b; TWU(ab)) \in I^* \times I^* \times \mathbb{R}^+\}$.

As other efficient one-phase algorithms for HUIM [9, 11, 24–26], the proposed CLS-Miner algorithm scans the database twice to compute the TU and TWU values of all items and 2-itemsets, as well as to construct the EUCS. During the first database scan, the TWU values and the utilities of all items are calculated. The TWU values of items are then used to establish a total order $\succ$ on items appearing in the transaction database, which is the order of ascending TWU values as suggested in the previous works [9–11]. The second database scan constructs the EUCS. During this database scan, items in transaction are reordered using the ascending order of TWU values.

Example 4 Consider database D of the running example as presented in Table 1. During the first database scan, the TWU values of all items are calculated (Fig. 1(a)). Then, during the second database scan, transactions are reordered by ascending order of TWU values (Fig. 1(b)), and the EUCS structure is built (Fig. 1(c)). In this example, the EUCS is represented as a triangular matrix, where each row/column represents an item, and each cell indicates the TWU of the itemset formed by combining the items corresponding to the column and row. An interesting observation is that some pairs of items never co-occur in the database, and hence have a TWU value of zero. For example, the item g does not appear in any of the transactions containing f or d.

In previous work [11], the following property was proposed to reduce the search space using the EUCS. This property is known as the Estimated Utility Co-occurrence Pruning (EUCP) property.

Property 7 (EUCP property [11]) Let there be a non-empty itemset $X = \{x_1, x_2, \ldots, x_z\}$. Furthermore, let there be an item y, such that Xy is an extension of X (that is, $y \succ x_z$). If $TWU(x_z y) < minutil$, then Xy and its transitive extensions are low-utility itemsets.

Proof Let Z be a transitive extension of Xy. As $Xy \subseteq Z$ and $x_z y \subseteq Xy$, it follows that $x_z y \subseteq Z$. Because $TWU(x_z y) < minutil$, Z and its supersets are low-utility itemsets (by Property 2). □

Item	*f*	*g*	*d*	*b*	*a*	*e*	*c*
TWU	30	38	58	61	65	88	96

(a)

Tid	Ordered transaction
T_1	(*d*,1), (*a*,1), (*c*,1)
T_2	(*g*,5), (*a*,2), (*e*,2), (*c*,6)
T_3	(*f*,5), (*d*,6), (*b*,2), (*a*,1), (*e*,1), (*c*,1)
T_4	(*d*,3), (*b*,4), (*e*,1), (*c*,3)
T_5	(*g*,2), (*b*,2), (*e*,1), (*c*,2)

(b)

Item	*f*	*g*	*d*	*b*	*a*	*e*
g	0					
d	30	0				
b	30	11	50			
a	30	27	38	30		
e	30	38	50	61	57	
c	30	38	58	61	65	88

(c)

Fig. 1 (a) Items are reordered by ascending order of TWU values; (b) transactions are sorted according to this order; (c) the EUCS matrix

In previous work [11], Property 7 was used to reduce the search space. In this paper, a more general version of this property is proposed to prune a larger part of the search space.

Property 8 (Generalized EUCP property) Let there be a non-empty itemset X and an item $x \in X$. Furthermore, let there be an item y, such that Xy is an extension of X. If $TWU(xy) < minutil$, then Xy and its transitive extensions are low-utility itemsets.

Proof Let Z be any extension of Xy. As $Xy \subseteq Z$ and $xy \subseteq Xy$, it follows that $xy \subseteq Z$. Because $TWU(xy) < minutil$, Z and its supersets are low-utility itemsets (by Property 2). □

The novel Chain-EUCP strategy applies Property 8 to directly prune an itemset Pxy and its transitive extensions without constructing their utility-lists. By Property 2, if there is a tuple $(x; y; TWU(xy))$ in the EUCS such that $TWU(xy) < minutil$, then any superset Pxy of xy has a utility lower than $minutil$. Thus, if the above condition is met, Pxy and its transitive extensions do not need to be considered by the proposed algorithm. The proposed Chain-EUCP strategy is different from the EUCP strategy [11] used in previous work. For an itemset Pxy, the EUCP only checks the pruning condition for the two last items x and y appended to Pxy, whereas the Chain-EUCP strategy checks the pruning condition $TWU(ab) < minutil$ for all pairs of items $a, b \in Pxy$. If the condition is met for any such pair of items, it follows that Pxy and its supersets are low-utility itemsets and, hence, that they do not need to be considered by the proposed algorithm (by Property 8). Therefore, the proposed Chain-EUCP strategy is more general than the EUCP strategy used in previous work [11], and can prune a larger part of the search space. The proposed algorithm, which relies on this strategy, is presented in Section 5.

4.2 Coverage concept

The second novel strategy proposed in this paper is based on a novel concept of coverage. This concept is inspired by the definition of frequent closed itemsets in FIM [4]. In this subsection, we first present the concept of coverage, then explain how this concept is used to calculate the utility of itemsets, and propose a new strategy for reducing the search space when mining CHUIs.

Definition 14 (Coverage) Let there be two items $i, j \in I$. Item j is said to cover i if $TidSet(i) \subseteq TidSet(j)$. The coverage of item i is denoted by $Cov(i)$ and defined as $Cov(i) = \{z | TidSet(i) \subseteq TidSet(z) \wedge z \in I\}$.

Property 9 (Relationship between the TWU and coverage) Let there be two items $i, j \in I$ in a database D. We have that $j \in Cov(i)$ if and only if $TWU(i) = TWU(ij)$.

Proof Let the notation $TidSet(x\overline{y})$ denote the set of Tids of transactions containing x but not y. The TWU of the itemset ij is defined by the following equation:

$$\begin{aligned} TWU(ij) &= \sum_{T_d \in D \wedge ij \subseteq T_d} TU(T_d) = \sum_{T_d \in TidSet(ij)} TU(T_d) \\ &= \sum_{T_d \in TidSet(ij)} TU(T_d) + \sum_{T_d \in TidSet(i\overline{j})} TU(T_d) \\ &\quad - \sum_{T_d \in TidSet(i\overline{j})} TU(T_d) \\ &= \sum_{T_d \in TidSet(i)} TU(T_d) - \sum_{T_d \in TidSet(i\overline{j})} TU(T_d) \\ &= TWU(i) - \sum_{T_d \in TidSet(i\overline{j})} TU(T_d). \end{aligned}$$

Therefore,

$$\begin{aligned} & TWU(ij) = TWU(i) \\ \Leftrightarrow & \sum_{T_d \in TidSet(i\overline{j})} TU(T_d) = 0 \Leftrightarrow TidSet(i\overline{j}) = \varnothing \\ \Leftrightarrow & TidSet(i) \subseteq TidSet(j) \Leftrightarrow j \in (i). \end{aligned}$$

□

This property provides a simple way of calculating the coverage of any single item $i \in I$ using the TWU of single items and the EUCS structure presented in the previous subsection.

Example 5 Consider the example database presented in Table 1. Figure 1 (top) indicates the TWU values of all items

and presents the corresponding EUCS (bottom). Consider the item b, we have $TWU(b) = 61$. In the EUCS, there are two tuples containing b, which are $(b, c, 61)$ and $(b, e, 61)$. The TWU values stored in these tuples are equal to $TWU(b) = 61$. Hence, the coverage of item b is $Cov(b) = \{c, e\}$.

In the following, an itemset obtained by performing the union of an item and its coverage is called a *coverage itemset*. For example, the itemset bce is a coverage itemset since it can be obtained by performing the union of the item b with its coverage $Cov(b) = \{c, e\}$.

To calculate the utility of any coverage itemset X such as bce without performing utility-list intersections or additional database scans, the following property is introduced.

Property 10 (Utility of a coverage itemset) Let there be a coverage itemset $X = pp_1p_2...p_r$. The utility of X can be efficiently calculated using the following equation:

$$u(X) = \sum_{1 \leqslant i \leqslant r} u(pp_i) - (r-1) \times u(p).$$

Proof Because $p_i \in Cov(p)$, it follows that $TidSet(p) \subseteq TidSet(p_i)$. Therefore, $TidSet(pp_i) = TidSet(p)$, and $TidSet(pp_1p_2...p_r) = TidSet(p)$., thus

$$\begin{aligned}
u(X) = u(pp_1p_2...p_r) &= \sum_{T_d \in TidSet(p)} u(pp_1p_2 \cdots p_r, T_d) \\
&= \sum_{T_d \in TidSet(p)} [u(p, T_d) + u(p_1, T_d) + \cdots + u(p_r, T_d)] \\
&= \sum_{T_d \in TidSet(p)} [[(u(p, T_d) + u(p_1, T_d)) + \cdots + (u(p, T_d) \\
&\quad + u(p_r, T_d))] - (r-1) \times u(p, T_d)] \\
&= \sum_{T_d \in TidSet(p)} [[(u(p, T_d) + u(p_1, T_d)) + \cdots + (u(p, T_d) \\
&\quad + u(p_r, T_d))] - (r-1) \times \sum_{T_d \in TidSet(p)} u(p, T_d)] \\
&= \sum_{1 \leqslant i \leqslant r} \sum_{T_d \in TidSet(p)} [u(p, T_d) + u(p_i, T_d)] - (r-1) \times u(p) \\
&= \sum_{1 \leqslant i \leqslant r} u(pp_i) - (r-1) \times u(p). \qquad \square
\end{aligned}$$

Example 6 Consider the example database given in Table 1. The coverage of item b is $Cov(b) = \{e, c\}$, as explained in Example 5. By applying Property 10, the utility of the coverage itemset bec can be calculated directly without constructing its utility-list as follows:

$$u(bec) = u(be) + u(bc) - u(b) = 25 + 22 - 16 = 31.$$

Using the above property, the utility of any coverage itemset X can be obtained directly by adding the utilities of 2-itemsets and subtracting the utility of a 1-itemset. Thus, the utility of X can be obtained without performing utility-list intersections or additional database scans. The only requirement for calculating the utility of an itemset using the above property is to precalculate the coverage of each item, and the utilities of 1-itemsets and 2-itemsets. The coverage of all items can be calculated efficiently using the EUCS structure by applying the proposed Algorithm 2.

Algorithm 2 The CoverageConstruct procedure

Input: The set of items I;
Output: The coverage of each item in I;

```
1: for each item x ∈ I do
2:     Cov(x) ⟵ ∅;
3:     for each item y ∈ I and y ≻ x do
4:         if (EUCS(x, y) = TWU(x)) then
5:             Cov(x) ⟵ Cov(x) ∪ y;
6:         end if
7:     end for
8: end for
```

Moreover, this paper also introduces a second property for pruning the search space of itemsets using the novel concept of coverage. This pruning strategy is integrated into the proposed CLS-Miner algorithm.

Property 11 (Pruning using the coverage) Let there be a single item p, and $Cov(p)$ be its coverage. Furthermore, consider an itemset $Y \subseteq Cov(p)$ and an itemset X such that $X \cap Y = \varnothing$ and $p \notin X$. If $p \cup X$ is a HUI, then $p \cup X \cup Y$ is also a HUI.

Proof Because Y is a subset of $Cov(p)$, it follows that:

$$TidSet(p \cup X \cup Y) = TidSet(p \cup X).$$

The utility of the itemset $p \cup X \cup Y$ is calculated as

$$\begin{aligned}
u(p \cup X \cup Y) &= \sum_{T_d \in D} u(p \cup X \cup Y, T_d) \\
&= \sum_{T_d \in TidSet(p \cup X)} [u(p \cup X, T_d) + u(Y, T_d)] \\
&= \sum_{T_d \in TidSet(p \cup X)} u(p \cup X, T_d) \\
&\quad + \sum_{T_d \in TidSet(p \cup X)} u(Y, T_d) \\
&= u(p \cup X) + \sum_{T_d \in TidSet(p \cup X)} u(Y, T_d) > u(p \cup X).
\end{aligned}$$

Thus, if $p \cup X$ is a HUI, then $u(p \cup X) \geqslant minutil$. Hence, $u(p \cup X \cup Y) > u(p \cup X) \geqslant minutil$. Therefore, $p \cup X \cup Y$ is a HUI. $\square$

Furthermore, it can be easily seen by Definition 14 (related to the coverage) and Property 6 (related to closure computa-

tions) that if an item j belongs to the coverage of an item i, it follows that j belongs to the closure of all itemsets containing the item i. Hence, the concept of coverage is also integrated into the process of closure computation.

The proposed algorithm integrating this strategy and the other strategies presented throughout Section 4 is presented in Section 5.

4.3 LBP strategy

A third novel pruning strategy is introduced in this section. It is motivated by the following observation. HUI-Miner [9] is a one-phase algorithm that introduced a novel pruning strategy to reduce the search space using the sum of *iutil* and *rutil* values in the utility-list of an itemset X (Property 4). If this sum is less than *minutil*, then the itemset X and its transitive extensions are low-utility itemsets and do not need to be considered. Although this strategy was shown to be effective for reducing the search space, a drawback is that it can only be applied to an itemset X after its utility-list has been fully constructed. However, constructing the utility-list of an itemset is very costly. It is, thus, desirable to design pruning strategies that can prune an itemset X and its transitive extensions before its utility-list is fully constructed. The proposed Chain-EUCP strategy is one such strategy and relies on information about item co-occurrences to reduce the search space. In this subsection, Property 4 is adapted to introduce a novel strategy for pruning an itemset X and its transitive extensions (called lower branches) before its utility-list is constructed. This strategy is called LBP and it relies on a new upper-bound on the utility of itemsets and their extensions, called the conjunction upper-bound utility. The LBP strategy is based on the following definitions and properties.

Definition 15 (Tid set difference) Let there be an itemset X and an itemset Y such that their Tid sets are $TidSet(X)$ and $TidSet(Y)$, respectively. The size difference of the Tid sets of X and Y is denoted as $diff(X, Y)$, and is calculated as $diff(X, Y) = abs(|TidSet(X)| - |TidSet(Y)|)$, where abs is the absolute value function.

Property 12 Let there be an itemset X and an itemset Y such that $X \cap Y = \emptyset$. The relation $diff(X, XY) \geqslant |TidSet(X)| - \min(|TidSet(X)|, |TidSet(Y)|)$ holds. For the sake of brevity, the value $|TidSet(X)| - \min(|TidSet(X)|, |TidSet(Y)|)$ will be denoted as $cdiff(X, Y)$.

Proof Since $TidSet(XY) \subseteq TidSet(X)$ and $TidSet(XY) \subseteq TidSet(Y)$, then $|TidSet(XY)| \leqslant |TidSet(X)|$ and

$$
\begin{aligned}
&|TidSet(XY)| \leqslant |TidSet(Y)| \\
&\quad \Longrightarrow |TidSet(XY)| \leqslant \min(|TidSet(X)|, |TidSet(Y)|) \\
&\quad \Longrightarrow diff(X, XY) = |TidSet(X)| - |TidSet(XY)| \\
&\qquad\qquad \geqslant |TidSet(X)| - \min(|TidSet(X)|, |TidSet(Y)|). \quad \square
\end{aligned}
$$

Definition 16 (The smallest sum of *iutil* and *rutil* values of an itemset) The smallest sum of the *iutil* and *rutil* values of an itemset X is denoted as $ul_{\min}(X)$. It is defined as the sum $ulm.iutils + ulm.rutils$ for the tuple $ulm \in ul(X)$ such that $\nexists ulk \in ul(X)$ where $(ulk.iutils + ulk.rutils) < (ulm.iutils + ulm.rutils)$.

Definition 17 (Conjunction upper-bound utility) Let there be an itemset X and an item $y \succ i, \forall i \in X$. The *conjunction upper-bound utility* of itemset X with respect to item y is denoted by $con(X, y)$ and defined as $con(X, y) = \sum_{ul \in UL(X)}(ul.iutil + ul.rutil) - cdiff(X, y) \times ul_{\min}(X)$.

This upper-bound is calculated very efficiently in the proposed CLS-Miner algorithm, and is used for pruning the search space based on the following properties.

Property 13 Let there be an itemset X and an item y, such that $y \succ i, \forall i \in X$. The value $con(X, y)$ is no less than the sum of *iutil* and *rutil* values in the utility-list of Xy, i.e., $con(X, y) \geqslant \sum_{ul \in UL(Xy)}(ul.iutil + ul.rutil)$.

Proof Let $UL(x\bar{y})$ denote the subset of the utility-list of x corresponding to transactions where y does not appear. In other words, $UL(x\bar{y}) = \{ul | ul \in UL(x) \wedge y \notin T_{ul.tid}\}$. Hence,

$$
\begin{aligned}
con(X, y) &= \sum_{ul \in UL(X)} (ul.iutil + ul.rutil) - cdiff(X, y) \times ul_{\min}(X); \\
con(X, y) &= \sum_{ul \in UL(Xy)} (ul.iutil + ul.rutil) \\
&\quad + \sum_{ul \in UL(X\bar{y})} (ul.iutil + ul.rutil) \\
&\quad - cdiff(X, y) \times ul_{\min}(X).
\end{aligned}
$$

Let $LH = con(X, y) - \sum_{ul \in UL(Xy)}(ul.iutil + ul.rutil)$. Then,

$$
\begin{aligned}
LH &= \sum_{ul \in UL(X\bar{y})} (ul.iutil + ul.rutil) - cdiff(X, y) \times ul_{\min}(X); \\
LH &\geqslant \sum_{ul \in UL(X\bar{y})} ul_{\min}(X) - cdiff(X, y) \times ul_{\min}(X); \\
LH &\geqslant diff(X, Xy) \times ul_{\min}(X) - cdiff(X, y) \times ul_{\min}(X); \\
LH &\geqslant (diff(X, Xy) - cdiff(X, y)) \times ul_{\min}(X) \geqslant 0.
\end{aligned}
$$

Therefore, $con(X, y) \geqslant \sum_{ul \in UL(Xy)}(ul.iutil + ul.rutil)$. $\square$

Property 14 (LBP) Let there be an itemset X and an item $y \succ i, \forall i \in X$. If $con(X, y) < minutil$, then Xy and all its transitive extensions are low-utility itemsets.

Proof As $con(X, y) \geqslant \sum_{ul \in UL(Xy)}(ul.iutil + ul.rutil)$ (by Property 13), and $con(X, y) < minutil$, it follows that $\sum_{ul \in UL(Xy)}(ul.iutil + ul.rutil) < minutil$. Therefore, by Property 4, the itemset Xy and its transitive extensions are low-utility itemsets. □

The above property is used to prune the search space. For a given itemset X, if $con(X, y)$ is less than *minutil*, then Xy and all its transitive extensions are low-utility itemsets, and can thus be pruned. As will be shown in the experimental evaluation of this paper, the LBP strategy can greatly reduce the number of extensions considered by the proposed CLS-Miner algorithm and, thus, also reduces its running time since the construction of numerous utility-lists can be avoided.

The proposed CLS-Miner algorithm integrating the novel LBP strategy and the other strategies presented throughout Section 4 is presented in Section 5.

4.4 Pre-check method for fast subsumption checks and closure computations

This section introduces a fourth technique that will be used to improve the performance of the proposed algorithm (described in the next section). This technique reduces the cost of subsumption checks and closure computations. These two operations are performed by most closed pattern mining algorithms to determine whether an itemset is closed and otherwise to directly obtain its closure (and, thus, prune the search space). Because both subsumption checks and closure computations are typically repeatedly performed by closed pattern mining algorithms, it is crucial to implement these operations efficiently in a CHUIM algorithm. In this section, we propose an efficient pre-check method to optimize both operations. Recall that closure computation consists of calculating the smallest closed itemset that is a superset of a given itemset. The subsumption relation is defined as follows.

Definition 18 (Subsume [20, 22]) Let there be two itemsets X and Y. The itemset X is said to subsume Y if and only if $Y \subset X$ and $sup(Y) = sup(X)$.

For example, *bce* subsumes *bc* because $bc \subset bce$ and $sup(bc) = sup(bce) = 3$.

Definition 19 (PreSet and PostSet [22]) Let there be an itemset $Px = P \cup x$, where P is a closed itemset and x is an item such that $x \notin P$. The *PreSet* of Px is defined as $PreSet(Px) = \{y \in I | y \notin Px \text{ and } y \prec x\}$. The *PostSet* of Px is defined as $PostSet(Px) = \{y \in I | y \notin Px \text{ and } y \succ x\}$.

When an itemset Y is generated by the algorithm, a subsumption check is performed to determine whether Y is a subset of a previously found closed itemset. If yes, it follows that Y is non-closed and that the supersets of Y do not need to be explored as their closure has already been discovered [22]. To implement subsumption checks efficiently, a challenge is that the set of previously found closed itemsets becomes larger as more CHUIs are found by the algorithm. Thus, each new itemset needs to be compared with an ever-increasing set of closed itemsets. To avoid the performance degradation that would result from such an implementation of subsumption checking, it is necessary to implement this operation more efficiently.

In the CHUD [20] and CHUI-Miner [26] algorithms, subsumption checks are implemented by the $IsSubsumedCheck$ procedure presented in Algorithm 3. To further optimize this procedure, the following paragraphs introduce a novel property establishing a relationship between the Tid sets of a pair of itemsets and the subsumption relation between these itemsets. This property is the basis for the efficient fast subsumption checking procedure proposed in this paper.

Algorithm 3 IsSubsumedCheck procedure

Input: An itemset Y, $PreSet(Y)$;

Output: Return *true* if Y is subsumed by an already mined CHUI. Otherwise, return *false*.

```
1: for each (item J ∈ PreSet(Y)) do
2:     if (TidSet(Y) ⊆ TidSet(J)) then
3:         Return true;
4:     end if
5: end for
6: Return false;
```

Property 15 (Subsume relation) Let there be two itemsets J and Y. Furthermore, assume that the Tid sets $TidSet(Y)$ and $TidSet(J)$ are sorted in ascending order of Tids. Moreover, let $TidSet_i(X)$ denote the Tid value stored at position i in the Tid set of an itemset X. If Y is subsumed by J, then the three following inequalities must hold:

(1) $|TidSet(Y)| \leqslant |TidSet(J)|$.

(2) $TidSet_i(Y) \geqslant TidSet_i(J), \forall i, 1 \leqslant i \leqslant |TidSet(Y)|$.

(3) $TidSet_{i-diff(J,Y)}(Y) \leqslant TidSet_i(J), \forall i, |TidSet(J)| - |TidSet(Y)| < i \leqslant |TidSet(J)|$.

Proof As Y is subsumed by J, it follows that $TidSet(Y) \subseteq TidSet(J)$ and, thus, that $|TidSet(Y)| \leqslant |TidSet(J)|$. We prove that inequality (2) holds by showing that $\nexists i, 1 \leqslant i \leqslant |TidSet(Y)|$ if $TidSet_i(Y) < TidSet_i(J)$. Assume that $\exists i, 1 \leqslant i \leqslant |TidSet(Y)|$ and $TidSet_i(Y) < TidSet_i(J)$. The integer m is

the smallest position that satisfies $TidSet_m(Y) < TidSet_m(J)$.

- If $m = 1$, then $TidSet_1(Y) < TidSet_1(J)$, and therefore $TidSet_1(Y) \notin TidSet(J)$. This contradicts the statement that Y is subsumed by J.
- If $m > 1$, as $TidSet_m(Y) < TidSet_m(J)$ and $TidSet_m(Y) > TidSet_{(m-1)}(Y) \geqslant TidSet_{(m-1)}(J)$, it follows that $TidSet_{(m-1)}(J) < TidSet_m(Y) < TidSet_m(J) \Rightarrow TidSet_m(Y) \notin TidSet(J)$, which contradicts the statement that Y is subsumed by J.

Therefore, $\nexists i, 1 \leqslant i \leqslant |TidSet(Y)|$ such that $TidSet_i(Y) < TidSet_i(J)$, inequality (2) holds. The inequality (3) can be proven in a similar way. □

Both the proposed CLS-Miner and the CHUI-Miner [26] algorithms are utility-list-based algorithms. Therefore, the subsumption check operation can be costly. To perform subsumption check efficiently, we introduce some improvements based on Property 15. These improvements consist of checking three conditions before performing a subsumption check.

- First, the length of the utility lists of Y and J is compared. If the length of the utility-list of Y is greater than that of J, Y is not subsumed by J.
- Second, if there exists a Tid in $TidSet(Y)$ that is smaller than that at the same position in $TidSet(J)$, then Y is also not subsumed by J.
- Third, if there exists a Tid in $TidSet(Y)$ that is greater than that at the same position in $TidSet(J)$ when reading the Tid sets from right to left, then Y is also not subsumed by J.

If an itemset Y passes these three conditions, then the proposed algorithm performs the subsumption check as in CHUI-Miner. Otherwise, the subsumption check is not performed as it follows by Property 15 that Y is not subsumed by X. The three proposed pre-check conditions are especially effective for sparse or short transaction datasets, as will be shown in the experimental evaluation. The detailed pseudo-code of the modified subsumption checking method is presented in Algorithm 4. It relies on a procedure called *precheckContain*, which checks the three conditions. This procedure is also used in the main search procedure of the proposed algorithm for calculating the closure of itemsets. The proposed algorithm is described in the next section.

Algorithm 4 The improved IsSubsumedCheck procedure

Input: An itemset Y and $PreSet(Y)$;

Output: Return *true* if Y is subsumed by an already mined CHUI. Otherwise, return $false$;

```
1: for each (item J ∈ PreSet(Y)) do
2:   if preCheckContain(J, Y) then
3:     if (TidSet(Y) ⊆ TidSet(J)) then
4:       Return true;
5:     else
6:       Return false;
7:     end if
8:   else
9:     Return false;
10:  end if
11: end for
```

Procedure preCheckContain(X,Y)

Input : X: an itemset, Y: an itemset;

Output : Return *true* if X potentially contains Y. Otherwise, return $false$;

```
1: lenX = |TidSet(X)|;
2: lenY = |TidSet(Y)|;
3: if (lenX < lenY) then
4:   Return false;
5: end if
6: for (int i = 0; i < lenX; i++) do
7:   if (TidSet(X).getTid(i) > TidSet(Y).getTid(i)) then
8:     Return false;
9:   end if
10:  if (TidSet(X).getTid(lenX − i) < TidSet(Y).getTid(lenY − i)) then
11:    Return false;
12:  end if
13: end for
14: Return true;
```

5 Proposed CLS-Miner algorithm

This section first presents the CLS-Miner algorithm and discusses its complexity. Then, a detailed example of how the algorithm works on the running example database is presented.

5.1 Designed algorithm

The proposed CLS-Miner algorithm finds closed itemsets efficiently by employing the search space browsing and closure computation techniques of the DCI_CLOSED algorithm [22], and by integrating the four novel techniques presented in Section 4. CLS-Miner applies the efficient Chain-EUCP strategy to prune the search space using the EUCS structure. This strategy can greatly reduce the number of join operations that are performed for constructing utility-lists, similarly to the less-general EUCP strategy used in previous work [11]. CLS-Miner also applies the designed LBP strategy to reduce the search space. Moreover, the concept of coverage and pruning using the coverage is integrated into closure computation to find potential CHUIs. In addition, CLS-Miner uti-

lizes the proposed fast subsumption checking method. The main procedure of CLS-Miner is given in Algorithm 5.

Algorithm 5 CLS-Miner

Input: a transaction database D and the *minutil* threshold;
Output: the complete set of CHUIs;
1: Scan D to calculate the TWU of each single item;
2: Let I^* be the list of items having TWU values no less than *minutil*;
3: Let $\succ$ be the ascending order of TWU values on items in I^*;
4: Scan D to build the utility-list of each item $i \in I^*$ and the EUCS_D structure;
5: Call **CoverageConstruct**();
6: **Search-CHUI** ($\emptyset$, $\emptyset$, the utility-list of items $\in I^*$, $EUCS_D$);

Procedure Search-CHUI()
Input: P: an itemset, $PreSet(P)$: a set of pre-extensions of P, $PostSet(P)$: a set of post-extensions of P, and the EUCS structure;
Output: the complete set of CHUIs;
1: **for each** itemset $x \in PostSet(P)$ **do**
2: $Px \longleftarrow P \cup x$;
3: Construct the utility list of Px, $UL(Px)$;
4: **if** $(SUM(UL(Px).iutils + SUM(UL(Px).rutils) \geqslant minutil)$ **then**
5: **if** $IsSubsumedCheck(Px, PreSet(P)) = false$ **then**
6: $postSetNew \longleftarrow \emptyset$;
7: Declare variable $passed = true$;
8: **for each** itemset $y \in PostSet(P)$ and $y \succ x$ **do**
9: **if** $con(Px, y) < minutil$ **then**
10: continue;
11: **end if**
12: **if** $(\exists i \in Px$ such that $EUCS(i, y) < minutil)$ **then**
13: continue;
14: **end if**
15: **if** $(y \in Cov(x))$ OR (preCheckContain(Px,y) AND $TidSet(Px) \subseteq TidSet(y)$) **then**
16: $Px \longleftarrow Px \cup y$;
17: Construct the utility list of Pxy, $UL(Pxy)$;
18: **if** $(SUM(UL(Pxy).iutils + SUM(UL(Pxy).rutils) < minutil)$ **then**
19: $passed = false$;
20: break;
21: **end if**
22: **else**
23: $postSetNew \longleftarrow postSetNew \cup y$;
24: **end if**
25: **end for**
26: **if** $passed = true$ **then**
27: **if** $SUM(UL(Pxy).iutils > minutil$ **then**
28: **Output** Pxy;
29: **end if**
30: $PreSet(Pxy) \longleftarrow PreSet(P)$;
31: $PostSet(Pxy) \longleftarrow postSetNew$;
32: **Search-CHUI**(Pxy, $PreSet(Pxy)$, $PostSet(Pxy)$, $EUCS_D$);
33: **end if**
34: $PreSet(P) \longleftarrow PreSet(P) \cup x$;
35: **end if**
36: **end if**
37: **end for**

The proposed algorithm first scans the database D to calculate the TWU values of all items (line 1). Then, the algorithm identifies the set I^* of all items having a TWU value no less than *minutil* (line 2). Items not in this set are henceforth ignored because they cannot be part of a HUI according to Property 2. The TWU values of these items are then used to establish a total order $\succ$ on items, which is the order of ascending TWU values (line 3). A second database scan is then performed. During this database scan, items in transactions are reordered according to the total order $\succ$, the utility-list of each item $i \in I^*$ is built and the EUCS structure is built (line 4). The coverage relation is then constructed by calling the *CoverageConstruct* procedure (line 5). Then, the recursive *Search-CHUI* procedure is called to recursively search for CHUIs using a depth-first search similar to the DCI_CLOSED [22], CHUD [20], and CHUI-Miner [26] algorithms. The procedure takes as input parameters the current itemset to be extended P, the two sets of items $PreSet(P)$ and $PostSet(P)$, and the $EUCS$ structure. The procedure outputs all the CHUIs that strictly contain P by analyzing all the valid closed itemsets that are obtained by extending P with the items in its $PostSet$.

The main steps of the *Search-CHUI* procedure are as follows. For each itemset x of $PostSet(P)$, an itemset $Px = P \cup x$ is created, denoted by Px, and its utility-list $UL(Px)$ is built using Algorithm 1 (lines 2 and 3). If the sum of the *iutil* and *rutil* values in the utility-list of Px is no less than *minutil*, then extensions of Px will be explored (line 4). Before exploring these extensions, the procedure $IsSubsumedCheck(Px, PreSet(P))$ is called to check whether Px is included in previously found closed itemsets. If yes, then supersets of X do not need to be explored (line 5). If no, the search procedure tries to merge Px with each item $y \in PostSet(P)$ such that $y \succ x$ to form a larger itemset Pxy. The variable *postSetNew* stores a set of items called postset items, which can be appended to Pxy to generate potential CHUIs. This variable is initialized to the empty set. A variable name *passed* is also used to mark an extension Pxy as a potential CHUI (lines 6 and 7). Before constructing the utility-list of Pxy, the algorithm checks whether the two following conditions are satisfied: (*i*) $con(Px, y)$ is no less than *minutil* (lines 9–11), and (*ii*) x and y co-occur and pass the pruning condition of the Chain-EUCP strategy (lines 12–14). If these conditions are satisfied, the search procedure checks whether the coverage relation is respected and if y belongs to the closure of Px (line 15). Then, the utility-list of Pxy is constructed (line 17). If the sum of the *iutil* and *rutil* values in the utility-list of Pxy is less than *minutil*, this means

that *Pxy* and its transitive extensions are low-utility itemsets. Hence, the search procedure stops adding items to *Pxy* to not generate its extensions (lines 18–20). Otherwise, *y* is added to *postSetNew* (line 22). If *Pxy* is a potential CHUI and its utility is no less than *minutil*, it is output (lines 26–29). *The Search-CHUI* procedure is then recursively called to continue exploring the search space (line 32). Finally, the item *x* is added to *PreSet*(*P*) (line 34). When the proposed algorithm terminates, all the CHUIs in the database have been obtained. We next prove that the algorithm is correct and complete.

Theorem 1 The proposed CLS-Miner algorithm is correct and complete at mining all CHUIs.

Proof To demonstrate that the algorithm finds all and only the CHUIs, we must first show that the algorithm can find all closed itemsets. This is clear since the proposed algorithm adopts the search procedure of the DCI_CLOSED algorithm [22] for closed itemset mining. This procedure is also employed by the CHUD [20] and CHUI-Miner [26] algorithms. Its search space browsing technique enumerates all possible itemsets. Using the *PostSet* set guarantees that the complete set of potential closed itemsets will be obtained, whereas the *PreSet* set guarantees that all duplicate closed itemsets will be pruned by the procedure *IsSubsumedCheck()*. Hence, the search procedure is correct and complete for finding the closed itemsets and eliminating all non-closed itemsets [22].

Then, we must show that the algorithm finds all itemsets that are HUIs. To calculate the utilities of itemsets, the CLS-Miner algorithm employs the utility-list structure and Property 3 [9]. It has been shown previously that this property is correct for calculating the utility of itemsets [9]. Thus, CLS-Miner can correctly calculate the utility of itemsets and identify only the HUIs.

Having established that the search procedure of the algorithm can find all itemsets that are closed and high utility, we must lastly show that the various pruning techniques used in CLS-Miner to enhance its performance do not prune HUIs. The basic pruning strategy used in CLS-Miner is Property 4. This property was demonstrated to prune only low-utility

itemsets in previous work [9]. Moreover, the CLS-Miner also integrates three novel pruning strategies presented in Section 4. These strategies are based on Properties 8, 11, and 13, respectively. These properties have been proven to be correct in Sections 4.1, 4.2, and 4.3, respectively.

Finally, it should be noted that the novel pre-check method for fast subsumption checking and closure computation, proposed in Section 4.4, is correct and has no influence on the output (by Property 15).

Hence, based on the above discussion, it can be concluded that the proposed CLS-Miner algorithm is correct and complete for mining all CHUIs. □

5.2 Complexity of CLS-Miner

In this subsection, we analyze the complexity of the proposed CLS-Miner algorithm by first considering each main step of the algorithm. Then, the global worst-case complexity of CLS-Miner is discussed and is compared with the complexity of the state-of-the-art CHUD and CHUI-Miner algorithms.

1) **Calculating the TWU of single items** The CLS-Miner algorithm first calculates the TWU values of all items. This requires a single database scan, and thus takes $O(n \times w)$ time, where n is the number of transactions and w is the average transaction length.
2) **Generating I^*** Then, CLS-Miner stores all items having TWU values no less than the *minutil* threshold in a list called I^*. This list is then sorted in ascending order of TWU values. This operation takes in the worst case $O(m \log m)$ time, where m is the number of distinct items in the database.
3) **Constructing the utility-lists of items in I^*** Thereafter, CLS-Miner builds the utility-list of each item in I^*. The utility-list of an item contains an entry with three fields for each transaction where the item appears. In the worst case, each item appears in all transactions. Thus, the worst-case space complexity for the initial utility-lists is $O(|I^*| \times m)$. Building these initial utility-lists requires a single database scan, and thus takes $O(n \times w)$ time. Constructing the utility-lists of each k-itemset ($k > 1$) using the utility-list construction procedure (Algorithm 1) can be performed in linear time [9].
4) **Constructing the EUCS structure** Another important step of the CLS-Miner algorithm is to build the EUCS. This operation is very fast as it also requires a single database scan. Furthermore, this structure occupies a small amount of memory bounded by $|I^*| \times |I^*|$. However, in practice, this structure is much smaller because usually limited number of pairs of items co-occur in real-life transaction databases (see Section 6). Thus, this structure can be implemented as a sparse matrix rather than a full triangular matrix. In our implementation, the EUCS is implemented as a hash map, where

each entry has an item x as key, and another hashmap as value, which maps each item y that co-occurs with x to the value $TWU(xy)$.

5) **Constructing the coverage relation** Another operation performed by CLS-Miner is calculating the coverage of items in I^* using the EUCS structure. In the worst case, each item is compared with each item, and each item appears in all transactions. Thus, the worst-case time complexity of this operation is $O(|I^*|^2 \times m^2)$. However, in practice there is a limited number of pairs of items that co-occur in transactions having the same TWU. Thus, calculating the coverage is very fast.

6) **Discovering all CHUIs** The last major operation performed by CLS-Miner is the discovery of all CHUIs by recursively applying the *Search-CHUI()* procedure. The complexity of this procedure is proportional to the number of times that the *IsSubsumedCheck()* procedure is called, which is proportional to the number of itemsets in the search space that are not pruned by the algorithm. In the worst case, no itemsets are pruned by the pruning strategies and, thus, all the subsets of $|I^*|$ must be considered by the algorithm. There are $2^{I^*} - 1$ subsets. Thus, the worst-case complexity is $O(2^{|I^*|} - 1)$.

Therefore, the worst-case time complexity of the CLS-Miner algorithm (Algorithm 5) is $O(2^{|I^*|})$, which is roughly $O(2^{|I|})$. It can, thus, be said that CLS-Miner is a pseudo-polynomial algorithm as its time complexity is roughly linear with the number of patterns that it visits in the search space. The number of patterns in the search space is determined by the effectiveness of the pruning strategies employed by CLS-Miner. Among all the operations performed by CLS-Miner, the discovery of CHUIs using the *Search-CHUI()* procedure is by far the most costliest in terms of execution time. Other operations such as calculating the TWU of items, generating I^*, creating the initial utility-lists, and constructing the coverage relation is generally not costly and has a negligible impact on the overall runtime of CLS-Miner as these operations are only performed once.

Having discussed the complexity of the proposed algorithm, in this paragraph we briefly discuss the complexity of the state-of-the-art CHUD [20] and CHUI-Miner [26] algorithms. CHUD is a two-phase algorithm. In the first phase, CHUD finds candidate CHUIs using a depth-first search. In the worst case, no itemsets are pruned and CHUD visits all itemsets in the search space, which has a time complexity of roughly $O(2^{|I|})$. Then, CHUD calculates the exact utility of each candidate in the second phase by scanning the database to eliminate low-utility itemsets. This process has a worst-case time complexity of $O(|I| \times n \times w \times v)$, where v is the average itemset length, as each itemset must be compared with each transaction. Thus, the worst-case time complexity of CHUD is roughly $O(2^{|I|})$. Similarly, it can also be established that the worst-case time complexity of CHUI-Miner [26] is also roughly $O(2^{|I|})$.

Although a worst-case time complexity of $O(2^{|I|})$ may seem high, it is typical of the complexity of efficient algorithms in the field of itemset mining, as the number of operations usually depends on the number of itemsets in the search space, which depends on the threshold value set by the user [3, 4]. However, in practice, the search space is often much smaller than $2^{|I|} - 1$ itemsets as not all items co-occur in a database. In the experimental evaluation section of this paper, it will be demonstrated that in practice the CLS-Miner algorithm outperforms the CHUD [20] and CHUI-Miner [26] algorithms on six benchmark datasets. One of the main reason is that CLS-Miner integrates novel pruning strategies allowing a larger part of the search space to be pruned. More specifically, the proposed Chain-EUCP and LBP strategies pruned up to 92.5% of the candidate CHUIs in our experiments. Thus, although the worst-case time complexity is roughly the same for the three algorithms, CLS-Miner has to process fewer itemsets in the search space. Other reasons for the excellent performance of CLS-Miner is that it incorporates a novel fast pre-check method, and that it also performs a single phase, thus avoiding the drawback of two-phase algorithms such as CHUD [20].

5.3 Illustrative example

To give a better understanding of how the proposed algorithm works, this subsection provides a detailed example of how the CLS-Miner algorithm is applied for the running example. Consider the database D of the running example depicted in Table 1, the corresponding unit profit values of items (Table 2), and that $minutil = 30$ (or, equivalently, $r\_minutil = 31\%$). Mining CHUIs using CLS-Miner is performed as follows.

1) The database D is scanned to calculate the TWU of each item. The result is shown in Table 3.
2) The list I^* of single promising items is created and sorted in ascending order of TWU values. The result is $I^* = \{f, g, d, b, a, e, c\}$.
3) The database D is scanned again to construct the utility-list of each single item in I^* (Fig. 2) and the $EUCS_D$ is

built (depicted in Fig. 1).

4) The coverage of each promising item is calculated (Table 5).

Table 5 The set of promising items with their support counts and coverage

1-itemset(P)	$\{f\}$	$\{g\}$	$\{d\}$	$\{b\}$	$\{a\}$	$\{e\}$	$\{c\}$
coverage(P)	$\{d,b,a,e,c\}$	$\{e,c\}$	$\{c\}$	$\{e,c\}$	$\{c\}$	$\{c\}$	
sup(P)	1	2	2	3	3	3	4

5) The *Search-CHUI* procedure begins its recursive search for CHUIs using item f. The set *PreSet* is initially empty and $PostSet = \{g, d, b, a, e, c\}$. The procedure explores the search space of CHUIs that are supersets of f by appending items from the $PostSet$ to f. According to the utility list of f (shown in Fig. 2), the sum of the *iutil* and *rutil* values of f is equal to 30, which is no less than *minutil*. Thus, extensions of f should be considered as potential HUIs. As no item precedes f according to the total order $\succ$, the set *PreSet* is empty and, thus, f is not subsumed by any items. Then, the algorithm considers appending items to f to compute its closure. The set *PostSet* of the closed itemset beginning with f is initialized to the empty set. Then, items are appended to f to try to generate the closure of f. Before appending an item to f, pruning conditions are checked to determine whether the resulting itemset could be a HUI. In this example, the algorithm first considers appending g to f to generate the itemset fg. The conditions are checked, and fg is pruned by the Chain-EUCP strategy, because there is a tuple $(f, g, 0)$ in the $EUCS_D$, which means that these two items do not co-occur in the database. Then, the algorithm considers appending d to f to create the itemset fd. That itemset is not pruned by the LBP strategy (as $con(fd) = 30 - 0 \times 30 = 30 = minutil$) and the Chain-EUCP strategy. Then, because d is in the coverage of f (Table 5), the utility-list of fd is constructed, that is $ul(fd) = \{(T_3, 17, 13)\}$. Because $sum(ul(fd).iutils) + sum(ul(fd).rutils) = 17 + 13 = 30 = minutil$, fd is considered as a potential HUI (a candidate). CLS-Miner next considers appending b, a, e, and c in the same way. This results in the CHUI $fdbaec$, having the utility-list $ul(fdbaec) = \{(T_3, 30, 0)\}$. Because the utility of the itemset $fdbaec$ is 30, which is no less than the *minutil* threshold, it is a CHUI and it is output. Then, f is added to the set $PreSet(f)$ and $PostSet(fdbaec) = \emptyset$. The loop with item f ends.

6) Next, CLS-Miner searches for CHUIs starting with item g. This process is similar to Step 5, with $PreSet=\{f\}$ and $PostSet=\{d, b, a, e, c\}$. The sum of the *iutil* and *rutil* values of $ul(g)$ is $(7 + 31 = 38) > 30 = minutil$. Then, it is found that g is not subsumed by the already-mined CHUI $fdbaec$. Thus, the algorithm considers extending g. The extension gd fails to pass the test of the *Chain-EUCP* strategy and it is, thus, pruned. The extensions gb and ga are also pruned in the same way. Next, the algorithm considers appending item e to g. The resulting itemset ge is not pruned by the two strategies, and it is found that e belongs to the coverage of g. The utility-list of ge is then constructed, that is $ul(ge) = \{(T_2, 17, 10)(T_5, 5, 6)\}$. Because ge may lead to a closed itemset, it is extended with item c in the same way. The utility-list of the itemset gec is $ul(gec) = \{(T_2, 17, 10)(T_5, 7, 4)\}$. However, $u(gec) = 24 < minutil = 30$ and thus gec is not a CHUI. Then, $PreSet(gec) = \{f, g\}$, $PostSet(gec) = \emptyset$, and the loop for g ends.

7) The remaining items are processed in the same way. Finally, the set of CHUIs is $CH = \{abcdef{:}30, bcde{:}40, bce{:}31, ace{:}31\}$, where the number next to each itemset indicates its utility.

6 Experimental evaluation

This section presents an extensive experimental evaluation to assess the performance of the proposed CLS-Miner algorithm, including its pruning strategies and the pre-check method for fast subsumption checks and closure computations. The performance of CLS-Miner is compared with the state-of-the-art CHUI-Miner [26] and CHUD [20] algorithms for CHUIM. CHUI-Miner was proposed recently for mining CHUIs, and it was shown to outperform the two-phase

f		
Tid	*iutil*	*rutil*
3	5	25

g		
Tid	*iutil*	*rutil*
2	5	22
5	2	9

d		
Tid	*iutil*	*rutil*
1	2	6
3	12	13
4	6	14

b		
Tid	*iutil*	*rutil*
3	4	9
4	8	6
5	4	5

a		
Tid	*iutil*	*rutil*
1	5	1
2	10	12
3	5	4

e		
Tid	*iutil*	*rutil*
2	6	6
3	3	1
4	3	3
5	3	2

c		
Tid	*iutil*	*rutil*
1	1	0
2	6	0
3	1	0
4	3	0
5	2	0

Fig. 2 Utility-lists of promising items

CHUD algorithm [20,26]. Experiments were performed on a computer equipped with a 64-bit Core i5, 2.4 GHz Intel Processor and 8 GB of RAM, running Windows 7 SP1 64-bit as the operating system. All the compared algorithms were implemented by extending the SPMF open-source java library [35] and ran on the J2SDK 1.7.0. Both real and synthetic datasets were used in the experiments. The characteristics of the datasets used in the experiments are given in Table 6, where #Trans, #Items, and #Avg indicate the number of transactions, the number of distinct items, and the average transaction length, respectively. The Mushroom, Retail, Chess, and Connect datasets were obtained from the FIMI Repository. The Foodmart dataset is the Microsoft foodmart 2000 database, and the ChainStore dataset was obtained from the NUMineBench software distribution [36].

Table 6 Details of the datasets

Dataset	#Trans	#Items	#Avg	Type
Mushroom	8,124	119	23.0	Dense
Chess	3,196	75	37	Dense
Connect	67,557	129	43.0	Dense
Retail	88,162	16,470	10.3	Sparse
Foodmart	4,141	1,559	4.4	Sparse
ChainStore	1,112,949	46,086	7.3	Sparse

The datasets have been selected because they include dense, sparse, and large datasets, and thus represent the main types of data seen in real-life applications. These datasets are also the main benchmark datasets used in the HUIM literature. All datasets except Foodmart and ChainStore do not include internal and external utility values. Thus, for these datasets, the internal and external utilities have been generated randomly in the [1, 5] and [1, 10] intervals, respectively, using a log-normal distribution, as in previous works [20,26].

6.1 Evaluation of the Chain-EUCP strategy

The first experiment was performed to evaluate the proposed Chain-EUCP strategy. In this experiment, CLS-Miner with the Chain-EUCP strategy (denoted as CLS-Miner$_{CE}$) was compared with a version of CLS-Miner without the Chain-EUCP strategy (denoted as CLS-Miner$_{NoCE}$). The algorithms were run on the Retail, ChainStore, Chess, and Connect datasets. The parameter *minutil* was varied and the number of candidates generated by the two versions of CLS-Miner was measured. Table 7 indicates the number of candidates generated by CLS-Miner$_{CE}$ and CLS-Miner$_{NoCE}$ on the Retail and ChainStore datasets. Results for the Chess and Connect datasets are provided in Table 8.

It can be observed that this strategy is quite effective on sparse datasets. Furthermore, it can be observed that when *minutil* is set to small values, few candidates are pruned. This is reasonable because fewer pairs of items in the $EUCS$ structure meet the condition $EUCS(x,y) < minutil$ when *minutil* is set to a small value. Thus, fewer candidates are pruned. However, the Chain-EUCP strategy is very effective and using this strategy, CLS-Miner$_{CE}$ considers up to 92% less candidates than CLS-Miner$_{NoCE}$.

Table 7 Number of candidates generated by CLS-Miner with/without the Chain-EUCP strategy on sparse datasets

Minimum utility/%	Retail		ChainStore	
	CLS-Miner$_{NoCE}$	CLS-Miner$_{CE}$	CLS-Miner$_{NoCE}$	CLS-Miner$_{CE}$
0.1	4,643,301	3,058,464	1,397,708	105,537
0.08	7,167,652	5,212,220	3,761,895	597,633
0.06	10,923,546	8,589,355	9,916,894	3,086,107
0.04	18,225,180	15,358,692	26,304,356	13,457,757
0.02	36,600,558	32,485,724	75,824,402	54,537,511
0.01	78,485,602	69,048,615	158,501,880	130,304,550

Table 8 Number of candidates generated by CLS-Miner with/without the Chain-EUCP strategy on dense datasets

Minimum utility/%	Chess		Connect	
	CLS-Miner$_{NoCE}$	CLS-Miner$_{CE}$	CLS-Miner$_{NoCE}$	CLS-Miner$_{CE}$
50	0	0	0	0
40	587	500	31	16
30	22,828	22,625	1,097	1,051
20	2,650,790	2,650,589	17,479	17,406
10	49,493,242	49,492,939	4,359,633	4,359,126

6.2 Evaluation of the LBP strategy

The second experiment was conducted to evaluate the effectiveness of the proposed LBP strategy at pruning the search space. This experiment was carried on the ChainStore, Retail, Chess, and Connect datasets. A version of CLS-Miner with the LBP strategy (denoted as CLS-Miner$_{Lbp}$) was compared with a version of CLS-Miner without the LBP strategy (denoted as CLS-Miner$_{NoLbp}$). The parameter *minutil* was varied and the number of candidates was measured for the two versions of CLS-Miner. Table 9 compares the number of candidates generated by CLS-Miner$_{Lbp}$ and CLS-Miner$_{NoLbp}$ on the ChainStore and Retail datasets, which are both sparse datasets. Results for the dense Chess and Connect datasets are provided in Table 10. These results show that candidate pruning using the LBP strategy can be very effective as it can prune up to 92.5% of the candidates.

6.3 Evaluation of pruning effectiveness

The influence of the two main pruning strategies Chain-

EUCP and LBP on the overall execution time of CLS-Miner was also evaluated. Figure 3 shows the execution times when each pruning strategy is applied individually and when both strategies are applied together in CLS-Miner. The performance of the Chain-EUCP and LBP strategies vary for different datasets. However, overall, it can be observed that the Chain-EUCP strategy is more effective than the LBP strategy on sparse datasets, and that the overall best performance is obtained when combining both pruning strategies for all datasets. The above analysis confirms the effectiveness of the proposed pruning strategies.

Table 9 Number of candidates generated by CLS-Miner with/without the LBP strategy on sparse datasets

Minimum utility/%	Retail		ChainStore	
	CLS-Miner$_{NoLbp}$	CLS-Miner$_{Lbp}$	CLS-Miner$_{NoLbp}$	CLS-Miner$_{Lbp}$
0.1	4,652,729	3,058,464	1,409,186	105,537
0.08	7,178,729	5,212,220	3,785,342	597,633
0.06	10,933,113	8,589,355	9,964,155	3,086,107
0.04	18,250,665	15,385,692	26,397,614	13,457,757
0.02	36,639,799	32,485,724	75,849,670	54,537,511
0.01	78,591,959	69,048,615	158,728,749	130,304,550

Table 10 Number of candidates generated by CLS-Miner with/without the LBP strategy on dense datasets

Minimum utility/%	Chess		Connect	
	CLS-Miner$_{NoLbp}$	CLS-Miner$_{Lbp}$	CLS-Miner$_{NoLbp}$	CLS-Miner$_{Lbp}$
50	0	0	0	0
40	873	500	140	16
30	23,148	22,625	1,385	1,051
20	2,651,131	2,650,589	17,801	17,406
10	49,501,084	49,492,939	4,359,928	4,359,126

6.4 Evaluation of the pre-check method for subsumption checking and closure computation

The effectiveness of the proposed pre-check method for fast subsumption checks and closure computations was also evaluated. Recall that a subsumption check consists of determining whether an itemset is included in a closed itemset that has already been found, and that closure computation consists of calculating the closure of an itemset. As mentioned, most closed itemset mining algorithms repeatedly perform subsumption checking and closure computation. Thus, it is crucial to optimize these operations. An experiment was carried out to assess the efficiency of the proposed pre-check method to optimize these two operations. We prepared another version of the proposed algorithm, called CLS-Miner$_{NoPreCheck}$, where the pre-check method used for subsumption checking and closure computation is deactivated. A comparison of the execution times of both versions of the algorithm is shown in Fig. 4 for the same datasets. It can be seen in this figure that the optimized version is up to 50% more efficient.

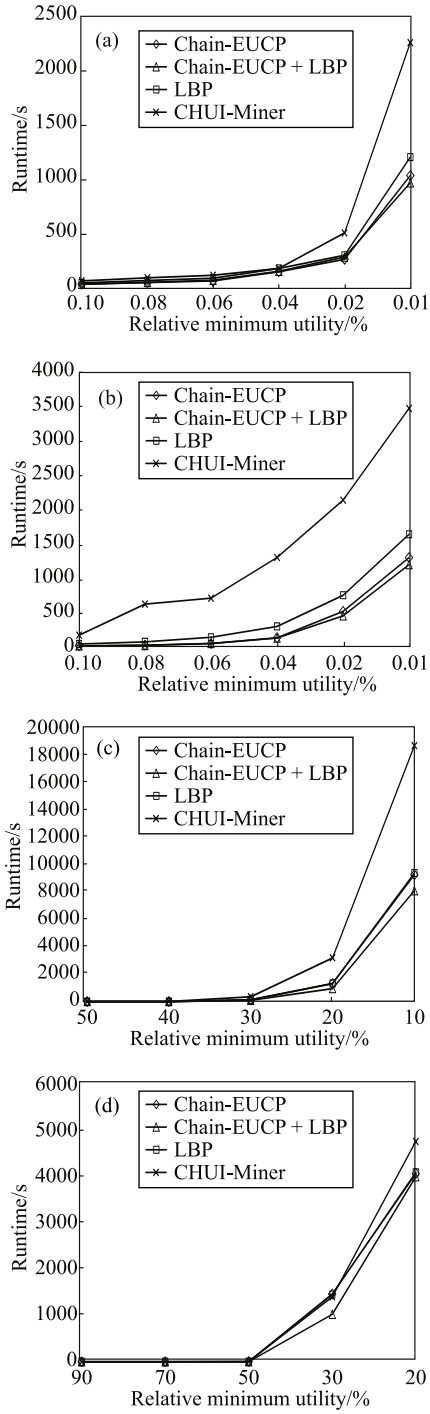


Fig. 3 Effect of pruning strategies. (a) Retail; (b) ChainStore; (c) Chess; (d) Connect

6.5 Efficiency of the proposed algorithm

This section compares the efficiency of the proposed CLS-

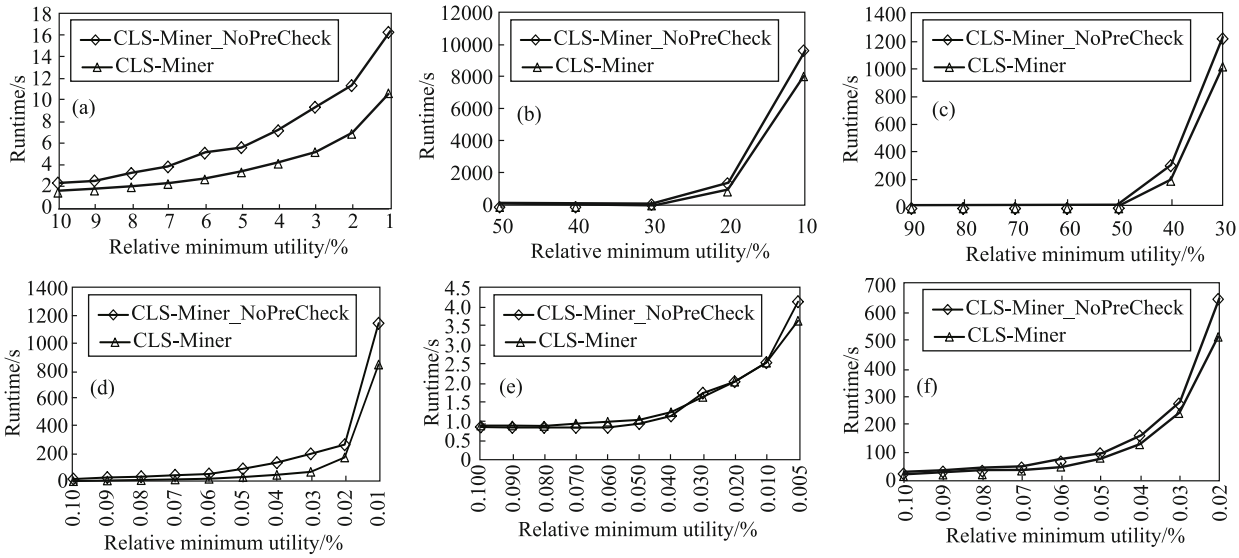


Fig. 4 Runtime of CLS-Miner with/without preCheckContain method. (a) Mushroom; (b) Chess; (c) Connect; (d) Retail; (e) Foodmart; (f) ChainStore;

Miner algorithm with CHUD [20] and CHUI-Miner [26] on the six datasets. CHUD is the first algorithm for mining CHUIs. CHUD includes three strategies called REG, RML, and DCM that greatly enhance its performance. However, CHUD is a two-phase algorithm. Therefore, it generates a huge number of candidates, and its repeatedly scans the database to calculate their exact utilities as mentioned above. CHUI-Miner [26] is a one-phase algorithm that relies on the utility-list structure [9] to mine CHUIs. A drawback of CHUI-Miner is that it has to fully construct the utility-lists of all the itemsets in its search space, and this is a costly operation. The proposed CLS-Miner algorithm addresses this drawback by introducing three strategies to prune low-utility itemsets before their utility-lists are constructed. CLS-Miner is also a one-phase algorithm. Therefore, we first compare the number of utility-list construction operations (candidates) performed by CHUI-Miner and the proposed CLS-Miner algorithm. Then, a comparison of runtimes and memory consumption of the three algorithms for mining CHUIs is provided.

The number of candidates generated by the CHUI-Miner and CLS-Miner algorithms are compared in Fig. 5. It can be observed that the proposed algorithm can generate far fewer candidates than CHUI-Miner, especially on sparse and large datasets such as ChainStore. The number of candidates produced by the proposed algorithm is about two orders of magnitudes smaller than that of CHUI-Miner.

A comparison of the execution times of the CHUD, CHUI-Miner, and CLS-Miner algorithms is presented in Fig. 6. Execution times include the time for reading the input database, discovering the patterns, and writing the CHUIs to an output file. Results show that the proposed CLS-Miner algorithm is faster than both CHUD and CHUI-Miner. When the *minutil* threshold is set to large values, the runtimes of the compared algorithm are quite similar. However, for small values of the *minutil* threshold, there is a big gap between the proposed algorithm and the two previous state-of-the-art algorithms. The reason for this is as follows. CHUD [20] is a two-phase algorithm. Hence, it suffers from the problem of generating a huge number of candidates and repeatedly scanning the database to calculate their utilities, which takes a lot of time. CHUI-Miner [26] is a single-phase algorithm as the proposed CLS-Miner algorithm. However, the proposed CLS-Miner algorithm relies on three novel effective pruning strategies that can prune a large part of the search space. Furthermore, the CLS-Miner algorithm also employs the efficient pre-check method to quickly perform subsumption checks and closure computations, which considerably reduces its execution time.

Table 11 compares the peak memory usage of the three algorithms for the six datasets when the *minutil* threshold is set to the smallest values used in the previous experiment. All memory measurements were done using the standard Java API. By consulting this table, it can be found that CHUD consumes the most memory among the three algorithms, on all datasets. The CHUI-Miner and CLS-Miner algorithms consume the same amount of memory on the Foodmart dataset. For the remaining datasets, the proposed algorithm slightly consumes more memory than CHUI-Miner. CHUI-Miner and CLS-Miner are single-phase algorithms that do not need to maintain candidates in memory. However, CLS-Miner needs

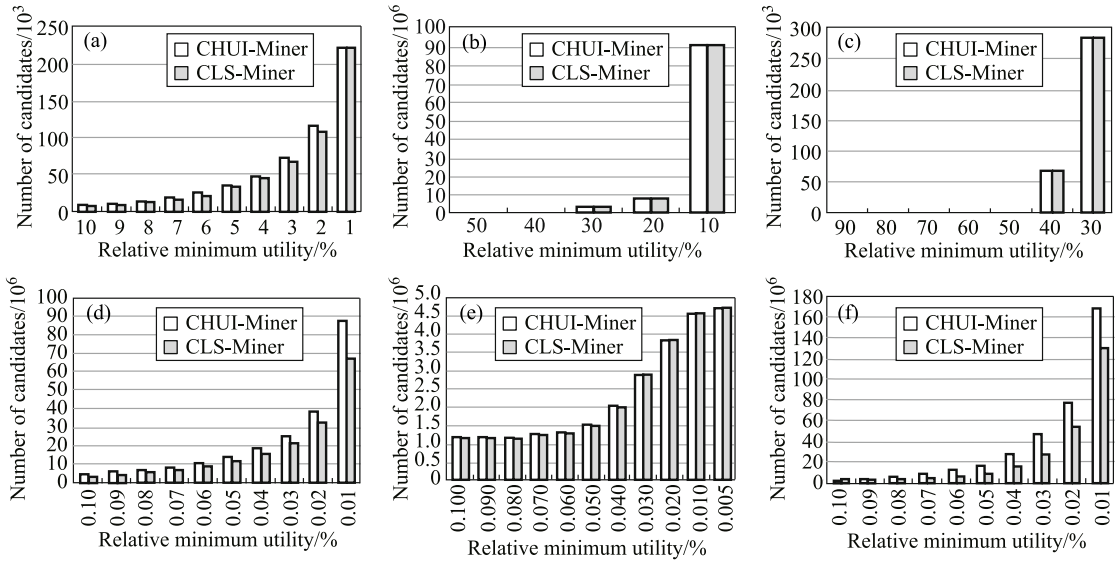


Fig. 5 Comparison of the number of candidates generated by CHUI-Miner and CLS-Miner. (a) Mushroom; (b) Chess; (c) Connect; (d) Retail; (e) Foodmart; (f) ChainStore

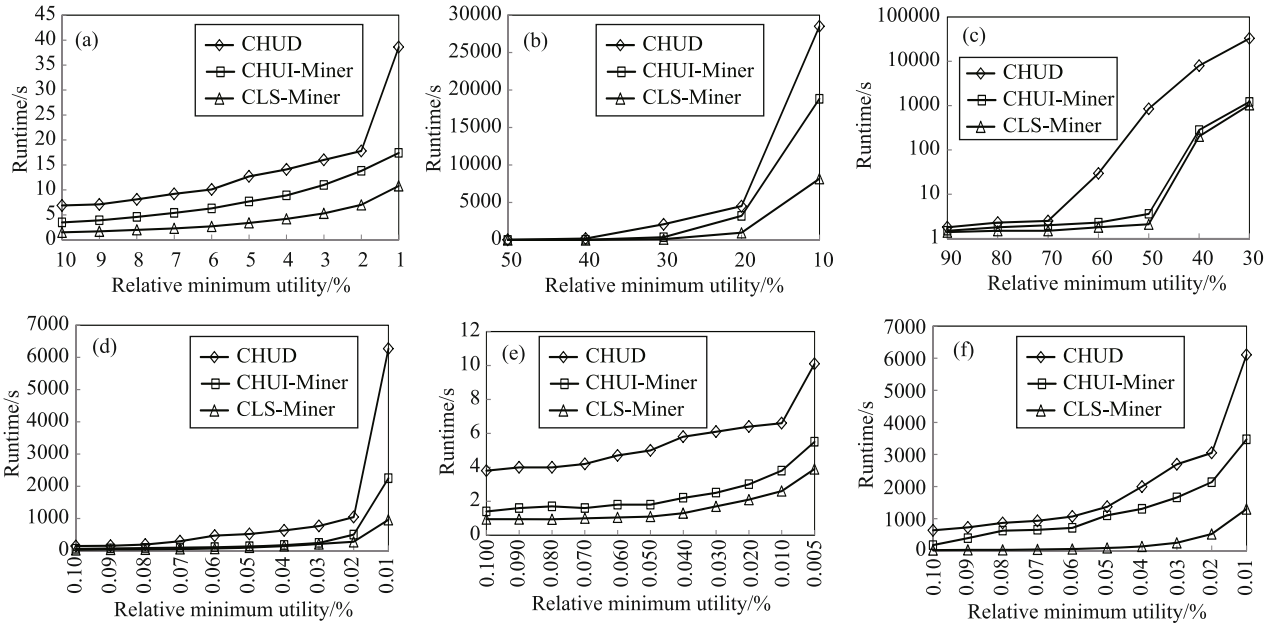


Fig. 6 Runtime comparison on different datasets. (a) Mushroom; (b) Chess; (c) Connect; (d) Retail; (e) Foodmart; (f) ChainStore

to store other structures in memory for its pruning strategies such as the EUCS and the coverage of items. Hence, considering this, it can be concluded that CLS-Miner has reasonable performance in terms of memory usage.

6.6 Scalability evaluation

We also performed experiments to assess the scalability of the proposed algorithm on a synthetic dataset T10I4NXKDYK, where the number of transactions Y and the number of items X were varied. For this experiment, the value of the *minutil* threshold was fixed to 0.5%, the number of items was varied from 2,000 to 10,000, and the number of transactions was varied from 100,000 to 500,000. Figure 7(a) shows the runtimes of the algorithms on the T10I4NXKD100K dataset when the number of distinct items is varied from 2,000 to 10,000. Figure 7(b) shows the runtimes of the compared algorithms on T10I4N4KDXK when the database size is varied from 100,000 to 500,000 transactions. It can be observed that the proposed algorithm has linear scalability with respect to the number of items and the number of transactions.

In conclusion, the experiments described previously show that the proposed methods are highly effective and the CLS-Miner outperforms CHUD and CHUI-Miner.

Table 11 Comparison of maximum memory consumption /MB

Dataset	CHUD	CHUI-Miner	CLS-Miner
Mushroom	318.7	266.1	329.8
Chess	749.6	134.4	140
Connect	1,967.8	536	620
Retail	390.4	282	305.2
Foodmart	277.8	169.1	169.1
ChainStore	2,698.5	1,259.6	1,402.3

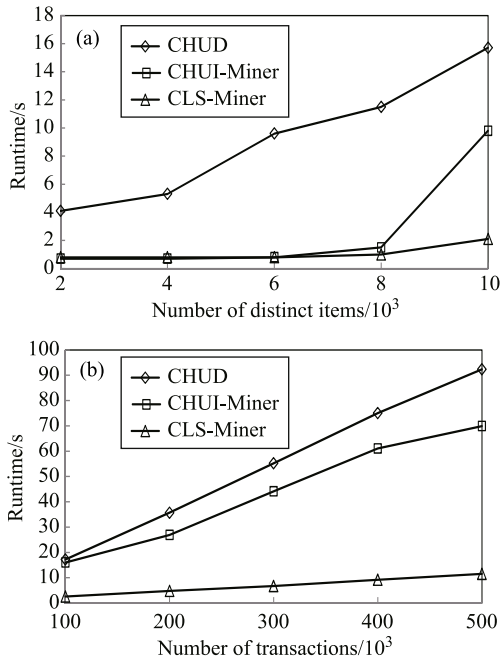


Fig. 7 Scalability of the compared algorithms under different parameter settings. (a) Varied number of items; (b) varied dataset sizes

7 Conclusion and future work

In this study, we introduced an efficient algorithm called CLS-Miner for mining CHUIs. The proposed algorithm relies on the utility-list structure to discover patterns in one phase. Moreover, to reduce the cost of utility-list intersection operations of utility-list-based algorithms, CLS-Miner introduces several novel ideas. First, an improved pruning strategy called Chain-EUCP based on the estimated utility of pairs of items was introduced. Second, an efficient LBP strategy was proposed to reduce the search space by pruning low-utility transitive extensions of itemsets. Third, a novel concept called coverage was presented to quickly discover the closure of itemsets and prune low-utility items. These strategies can prune itemsets without fully constructing their utility-lists. Finally, a fast pre-check method was introduced to quickly perform closure computations and subsumption checks. This method is useful for the problem of mining CHUIs.

An extensive experimental evaluation was conducted to evaluate the proposed CLS-Miner algorithm and the proposed techniques introduced in this algorithm. Results have shown that CLS-Miner and its techniques are highly efficient. In particular, the Chain-EUCP and the LBP strategies can prune up to 93% of candidates and the pre-check method can reduce the total runtime by up to 50%. In terms of runtime, CLS-Miner is up to several orders of magnitude faster than existing methods for mining CHUIs. A scalability experiment has also shown that the algorithm has linear scalability with respect to the number of items and the number of transactions.

In recent years, big data has emerged as an important research topic. Therefore, it is our plan to develop a distributed version of CLS-Miner that can be run on cloud-computing platforms for processing huge datasets.

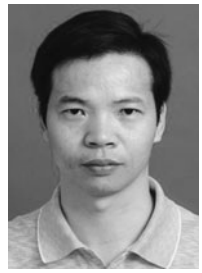
Acknowledgements The research was funded by the National Natural Science Foundation of China (Grant Nos. 61133005, 61432005, 61370095, 61472124, 61202109, and 61472126), and the International Science and Technology Cooperation Program of China (2015DFA11240 and 2014DFBS0010). T.-L. Dam was also partially supported by the science research fund of Hanoi University of Industry, Hanoi, Vietnam.

References

1. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases. 1994, 487–499
2. Zaki M J, Gouda K. Fast vertical mining using diffsets. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2003, 326–335
3. Han J W, Pei J, Yin Y W. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining and Knowledge Discovery, 2004, 8(1): 53–87
4. Han J, Wang J, Lu Y, Tzvetkov P. Mining top-k frequent closed patterns without minimum support. In: Proceedings of IEEE International Conference on Data Mining. 2002, 211–218
5. Grahne G, Zhu J. Fast algorithms for frequent itemset mining using fp-trees. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(10): 1347–1362
6. Deng Z H, Lv S L. PrePost+: an efficient N-lists-based algorithm for mining frequent itemsets via children-parent equivalence pruning. Expert Systems with Applications, 2015, 42(13): 5424–5432
7. Dam T L, Li K, Fournier-Viger P, Duong Q H. An efficient algorithm for mining top-rank-k frequent patterns. Applied Intelligence, 2016, 45(1): 96–111
8. Liu Y, Liao W K, Choudhary A. A two-phase algorithm for fast discovery of high utility itemsets. In: Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining. 2005, 689–695
9. Liu M, Qu J. Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management. 2012, 55–64

10. Tseng V, Shie B E, Wu C W, Yu P. Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(8): 1772–1786
11. Fournier-Viger P, Wu C W, Zida S, Tseng V. FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proceedings of International Symposium on Methodologies for Intelligent Systems. 2014, 83–92
12. Song W, Liu Y, Li J. BAHUI: fast and memory efficient mining of high utility itemsets based on bitmap. International Journal of Data Warehousing and Mining, 2014, 10(1): 1–15
13. Song W, Zhang Z, Li J. A high utility itemset mining algorithm based on subsume index. Knowledge and Information Systems, 2015, 1–26
14. Duong Q H, Liao B, Fournier-Viger P, Dam T L. An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies. Knowledge-Based Systems, 2016, 104: 106–122
15. Ahmed C, Tanbeer S, Jeong B S, Lee Y K. Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(12): 1708–1721
16. Yang Q. Three challenges in data mining. Frontiers of Computer Science in China, 2010, 4(3): 324–333
17. Chen J, Li K, Tang Z, Bilal K, Yu S, Weng C, Li K. A parallel random forest algorithm for big data in a spark cloud computing environment. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(4): 919–933
18. Song W, Liu Y, Li J. Mining high utility itemsets by dynamically pruning the tree structure. Applied Intelligence, 2014, 40(1): 29–43
19. Fournier-Viger P, Lin J C W, Duong Q H, Dam T L. In: Fujita H, Ali M, Selamat A, et al, eds. FHM+: Faster High-Utility Itemset Mining Using Length Upper-Bound Reduction. Cham: Springer International Publishing, 2016, 115–127
20. Tseng V S, Wu C W, Fournier-Viger P, Yu P S. Efficient algorithms for mining the concise and lossless representation of high utility itemsets. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(3): 726–739
21. Pasquier N, Bastide Y, Taouil R, Lakhal L. Efficient mining of association rules using closed itemset lattices. Information Systems, 1999, 24(1): 25–46
22. Lucchese C, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(1): 21–36
23. Zaki M J, Hsiao C J. Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(4): 462–478
24. Fournier-Viger P. FHN: efficient mining of high-utility itemsets with negative unit profits. In: Proceedings of International Conference on Advanced Data Mining and Applications. 2014, 16–29
25. Tseng V, Wu C W, Fournier-Viger P, Yu P. Efficient algorithms for mining top-k high utility itemsets. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(1): 54–67
26. Wu C W, Fournier-Viger P, Gu J Y, Tseng V S. Mining closed+ high utility itemsets without candidate generation. In: Proceedings of Conference on Technologies and Applications of Artificial Intelligence. 2015, 187–194
27. Chan R, Yang Q, Shen Y D. Mining high utility itemsets. In: Proceedings of the 3rd IEEE International Conference on Data Mining. 2003, 19–26
28. Lan G C, Hong T P, Tseng V S. An efficient projection-based indexing approach for mining high utility itemsets. Knowledge and Information Systems, 2014, 38(1): 85–107
29. Gouda K, Zaki M J. Genmax: An efficient algorithm for mining maximal frequent itemsets. Data Mining and Knowledge Discovery, 2005, 11(3): 223–242
30. Uno T, Kiyomi M, Arimura H. LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: Proceedings of IEEE ICDM Workshop on Frequent Itemset Mining Implementations. 2004
31. Szathmary L, Valtchev P, Napoli A, Godin R, Boc A, Makarenkov V. A fast compound algorithm for mining generators, closed itemsets, and computing links between equivalence classes. Annals of Mathematics and Artificial Intelligence, 2014, 70(1–2): 81–105
32. Fournier-Viger P, Wu C W, Tseng V S. Novel concise representations of high utility itemsets using generator patterns. In: Proceedings of the International Conference on Advanced Data Mining and Applications. 2014, 30–43
33. Shie B E, Philip S Y, Tseng V S. Efficient algorithms for mining maximal high utility itemsets from data streams with different models. Expert Systems with Applications, 2012, 39(17): 12947–12960
34. Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules. In: Proceedings of the International Conference on Database Theory. 1999, 398–416
35. Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu C W, Tseng V S. SPMF: a Java open-source pattern mining library. Journal of Machine Learning Research, 2014, 15: 3569–3573
36. Pisharath J, Liu Y, Liao W K, Choudhary A, Memik G, Parhi J. NU-MineBench 2.0. CUCIS Technical Report CUCIS-2005-08-01. 2005

Thu-Lan Dam received the BS and MS degrees in computer science from Hanoi University of Science and Technology, Vietnam in 2004 and 2009. She is currently working toward the PhD degree in computer science at College of Computer Science and Electronic Engineering, Hunan University, China. Her research interests are optimization, operational research, data mining and knowledge discovery.



Kenli Li received the PhD degree in computer science from Huazhong University of Science and Technology, China in 2003. He was a visiting scholar at University of Illinois at Urbana Champaign, USA from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University, China, and deputy di-

rector of National Supercomputing Center in Changsha. His major research areas include parallel and distributed processing, cloud computing, scheduling algorithms, computer simulation, biological computing, and big data management. He has published more than 100 research papers in international conferences and journals such as IEEE-TC, IEEE-TPDS, JPDC, ICPP, CCGrid. He is an outstanding member of CCF. He is a member of the IEEE and serves on the editorial boards of IEEE Transactions on Computers.

Philippe Fournier-Viger is a full professor at School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, China. He received his PhD degree in cognitive computer science from the University of Quebec in Montreal, Canada in 2010. He has published more than 85 research papers in refereed international conferences and journals. His research interests include data mining, pattern mining, text mining, intelligent tutoring systems, knowledge representation and cognitive modeling. He is the founder of the popular SPMF open-source data mining library.

Quang-Huy Duong received the MS degree in computer science from Hunan University, China in 2016. He is currently working toward the PhD degree in computer science at the Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology, Norway. His research interests are optimization, data stream mining and bioinformatics.