On the Enumeration of Frequent High Utility Itemsets: A Symbolic AI Approach

Amel Hidouri [1]
CRIL – CNRS UMR 8188, University of Artois, France
LARODEC, University of Tunis, Tunisia

Said Jabbour
CRIL – CNRS UMR 8188, University of Artois, France

Badran Raddaoui
SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France

Abstract

Mining interesting patterns from data is a core part of the data mining world. High utility mining, an active research topic in data mining, aims to discover valuable itemsets with high profit (e.g., cost, risk). However, the measure of interest of an itemset must primarily reflect not only the importance of items in terms of profit, but also their occurrence in data in order to make more crucial decisions. Some proposals are then introduced to deal with the problem of computing high utility itemsets that meet a minimum support threshold. However, in these existing proposals, all transactions in which the itemset appears are taken into account, including those in which the itemset has a low profit. So, no additional information about the overall utility of the itemset is taken into account. This paper addresses this issue by introducing a SAT-based model to efficiently find the set of all frequent high utility itemsets with the use of a minimum utility threshold applied to each transaction in which the itemset appears. More specifically, we reduce the problem of mining frequent high utility itemsets to the one of enumerating the models of a formula in propositional logic, and then we use state-of-the-art SAT solvers to solve it. Afterwards, to make our approach more efficient, we provide a decomposition technique that is particularly suitable for deriving smaller and independent sub-problems easy to resolve. Finally, an extensive experimental evaluation on various popular datasets shows that our method is fast and scale well compared to the state-of-the art algorithms.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Information systems → Data mining

Keywords and phrases Data Mining, High Utility Itemsets, Propositional Satisfiability

Digital Object Identifier 10.4230/LIPIcs.CP.2022.27

1 Introduction

The broad topic of data mining research aims to discover a set of relevant patterns that together represent the properties of the data. The successful use of data mining in e-commerce and e-marketing became a core practice in the retail industry. Pattern discovery, one of the most important sub-fields of data mining, involves computing interesting patterns in databases. Retailers are using such patterns to find customer habits in order to provide better services and increase sales. Traditional itemset mining models can be characterized into two lines of work. The first one, known as *Frequent Itemsets Mining* (in short, FIM), is based on the popular metric of support (i.e., the number of transactions involving the itemset) to determining how interesting a motif is. Specifically, FIM seeks to identify patterns whose frequency exceeds a predefined threshold. Nevertheless, frequency alone is often considered as a poor measure of interestingness [29]. In fact, frequent itemsets have a significant bottleneck

[1] Corresponding author

28th International Conference on Principles and Practice of Constraint Programming (CP 2022).
Editor: Christine Solnon; Article No. 27; pp. 27:1–27:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in that they only reflect the occurrence of items in a database and miss their importance, e.g., items that can be rare but generate more profit. Typically, the significance of an item in each transaction can be different. The second line of research consists of *High Utility Itemset Mining* (HUIM, for short), which is an extension of FIM. Basically, HUIM is a research area designed to address the shortcomings of frequency-based algorithms by taking in addition to the item frequency the significance/interestingness of items into account, such as the price, the quantity, etc., when mining patterns. A high utility itemset is then an itemset whose utility value is greater than a user-specified threshold. Generally speaking, the utility can be quantified in terms of cost, risk, profit or any other user preference relations among items. The HUIM task has emerged as a key data mining primitive in many practical applications, including market analysis, customer trend analysis and financial analysis [13]. The two aforementioned lines of research, however, are generally considered separately. To be precise, the two frameworks were designed with different goals in mind, either for mining the set of items that occur frequently while ignoring their profit, or for computing the set of items that yield the highest gain values as a sole criterion while avoiding the frequency measure. In the last case, non-frequent itemsets could be considered as high utility itemsets. In recent years, with ever-advancing technology, one may be more interested in finding at the same time frequently purchased items with high gain values. In fact, computing such itemsets can help sales managers understand customer research behavior and what she/he needs, as well as provide appropriate product combinations. For example, a retail store manager can use this knowledge to make decisions to keep products neighboring or also to promote products. These itemsets can also be used to assess the risk of selling a product by removing for instance very seldom interesting products from the market.

In the literature, a number of frequent utility-driven mining methods have been studied, each with its own advantages. Specifically, Wei et al. [30] proposed a novel algorithm, called `FCHUIM`, that combines the frequency and utility constraints to find frequent high utility itemsets. This approach considers a candidate itemset as frequent if and only if it is covered by a minimum number of transactions and his utility in these transactions is greater than a user-specified minimum utility value. `FCHUIM` is based on a closed high utility representation and it employs a nested list to eliminate non-frequent itemsets. Furthermore, `HU-FIMi` [28] is another single phase algorithm to compute efficiently high utility frequent itemsets in transactional databases. It exploits different orderings of items and introduces two new pruning measures (cutoff and suffix utility) in order to reduce the search space exploring cost. Another specialized form of frequent utility-based data mining field is to extract only the itemsets with the highest utility values (i.e., above a fixed threshold value). As a result, transactions involving the itemset with the lowest utility value are not considered as covers for this itemset because no valuable information is added to the itemset's overall utility value in the database. `2P-UF` algorithm [32] was the first to address the problem of computing such patterns. It considers utility frequent motifs to be a subset of the high utility itemset problem. This approach is a two-phases algorithm based on a quasi-support measure that addresses the issue of the support-utility measure's non-monotonicity. Consequently, this algorithm is not suitable to handle large-scale databases. In contrast to `2P-UF`, `FUFM` algorithm [27] was introduced to handle the frequent-utility task as a subset of the frequent itemset mining problem. To find such patterns among frequent itemsets, it employs a metric known as extended support. Since it is based primarily on the frequent itemset mining approach, this algorithm is very simple and fast. A parallel version of `FUFM`, called `P-FUFM` [26], is a two-phases based algorithm that aims to reduce the running time of the `FUFM` algorithm. In the first step, this algorithm generates candidates, and in the second phase, it computes utilities. The scheme is to implement a parallel generation of candidates as well as the corresponding utilities. Unfortunately, all these designed algorithms suffer from a degradation in scalability for large scale databases.

In this paper, we are mainly interested in mining more efficiently frequent high utility itemsets from transaction databases in the case when the frequency metric is applied to transactions in which the itemset appears, and also when such metric is restricted to the transactions with highest gain value. To address these two problems, we propose a novel symbolic framework based on propositional logic to efficiently extract a concise set of itemsets that are frequently encountered while yielding the highest profit margins in a transaction database. In fact, symbolic Artificial Intelligence (AI) approaches, such as Boolean Satisfiability (SAT) and Constraint Programming (CP), are applied in data mining by translating the problem of mining patterns in terms of constraints and then delegate the enumeration of solutions to the appropriate solver (e.g., [2, 9, 14, 16, 20]). The application of symbolic AI to data mining is supported by its theoretical and algorithmic foundations and the flexibility it affords, i.e., the ability to add new user-specified constraints to control interesting patterns without the need to modify, from scratch, the underlying algorithms. Furthermore, the close relationship between constraint-based languages and pattern discovery enables data mining problems to benefit from a variety of powerful propositional satisfiability-based solving techniques in order to improve the efficiency of such approaches. In this paper, we propose two SAT-based approaches: the first aims to compute the set of frequent high utility itemsets, while the second is for identifying all patterns that are local high utility frequent itemsets.

2 Formal Preliminaries

2.1 High Utility Itemset Mining

Let Ω denote a universe of items (or symbols), called alphabet. The elements of Ω are denoted by the letters a, b, c, etc. A subset of Ω ($I \subseteq \Omega$) is called an *itemset*. The set of all itemsets over Ω are denoted as 2^Ω, and the capital letters I, J, K, etc. are used to represent the elements of 2^Ω. Typically, a *transaction* is an ordered pair (i, I) where $1 \leq i \leq m$, called the *transaction identifier* (TID, for short), and I an itemset, i.e., $(i, I) \in \mathbb{N} \times 2^{|\Omega|} \setminus \emptyset$. When there is no confusion, a transaction will be simply denoted as T_i. A *transaction database* D is defined as a finite non-empty set of transactions where each transaction identifier refers to a unique itemset. Given a transaction database D and an itemset I, the *cover* of I in D is defined as follows: $\texttt{Cover}(I, D) = \{i \in \mathbb{N} \mid (i, J) \in D \textit{ and } I \subseteq J\}$. The *support* of I in D is then defined as the cardinality of $\texttt{Cover}(I, D)$, i.e., $\texttt{Supp}(I, D) = |\texttt{Cover}(I, D)|$. A high utility itemset $I \subseteq \Omega$ s.t. $\texttt{Supp}(I, D) \geq 1$ is *closed* if and only if for any itemset J with $I \subset J$, $\texttt{Supp}(J, D) < \texttt{Supp}(I, D)$.

In the high utility setting, each item $a \in \Omega$ is associated with a positive number that indicates its *external utility* (e.g., unit profit). We write $w_{ext}(a)$ for the external utility of a. In addition, each item a in a transaction T_i is associated with a positive value $w_{int}(a, T_i)$, called its *internal utility*. Based on these two kinds of utility, the *utility of an item* a in a transaction T_i, written $u(a, T_i)$, is computed as follows: $u(a, T_i) = w_{int}(a, T_i) \times w_{ext}(a)$. Now, the utility of an itemset I in a transaction T_i, denoted by $u(I, T_i)$, is defined as $u(I, T_i) = \sum_{a \in I \subseteq T_i} u(a, T_i)$. Then, the *utility of an itemset* I in the entire database D is defined as $u(I, D) = \sum_{T_i \in D \ \mid \ I \subseteq T_i} u(I, T_i)$.

Given a transaction database D and a user-specified utility threshold θ, the classical high utility itemset mining problem aims at finding the set of all itemsets in D whose utility value is no less than θ. More formally, the aim is to compute the set $\{I : u(I, D) \mid I \subseteq \Omega, u(I, D) \geq \theta\}$. An itemset with a utility greater than the minimum utility threshold θ is called a *high utility itemset* (HUI, for short).

In order to prune the search space, existing proposals of HUIM use the so-called *Transaction Weighted Utilization* (TWU, for short), which is an upper bound of the utility measure, together with the property of anti-monotonicity in order to filter out the candidate itemsets that are not high utility [22]. More formally, the transaction utility of a transaction T_i in D, denoted by $TU(T_i)$, is the sum of the utility of all items in T_i, i.e., $TU(T_i) = \sum_{a \in T_i} u(a, T_i)$. Then, the transaction weighted utilization of an itemset X in a transaction database D, denoted by $TWU(X, D)$, is defined as: $TWU(X, D) = \sum_{(i,T_i) \in D \ \mid \ X \subseteq T_i} TU(T_i)$.

Another task in data mining related to HUIM problem consists in enumerating the set of HUIs by taking into account the frequency. Such itemsets are called *frequent high utility itemsets* (FHUIs, for short). To be precise, given a support and utility minimum thresholds δ and θ respectively, an itemset I is a FHUI in D iff. $\texttt{Supp}(I, D) \geq \delta$ and $u(I, D) \geq \theta$. We denote by FHUIM the task of computing the set of FHUIs in D. Clearly, the HUIM task is a particular case of FHUIM where δ is set to 1.

▶ **Example 1.** Consider the transaction database shown in Table 1 (which will be used throughout the paper). For the sake of simplicity, we set the external utility of each item to 1. In fact, every transaction database can be represented as a single table by multiplying the internal and external utilities of items. In that sense, each item has a single number that represents its utility in the transaction. Let $\theta = 20$ be a minimum utility threshold, and $\delta = 2$ be a minimum support threshold. Then, the set of FHUIs in D are $\{a\}$, $\{a, b\}$, and $\{a, b, g\}$.

Table 1 Sample Transaction Database.

TID	Items						
T_1	$(a, 8)$	$(b, 2)$					$(g, 1)$
T_2		$(b, 6)$	$(c, 3)$		$(e, 2)$		
T_3			$(c, 4)$	$(d, 3)$			
T_4	$(a, 6)$			$(d, 4)$	$(e, 1)$		
T_5	$(a, 8)$	$(b, 7)$				$(f, 2)$	$(g, 1)$

Now, we wish to emphasize that the utility of itemsets is over-estimated when mining FHUIs from transaction databases. In other words, all transactions containing the candidate itemset are considered without regard for their utility value in these transactions: even if the utility of an itemset I in a transaction T is low, T is chosen if it contains I. To be precise, the utility measure of an itemset I (i.e., $u(I, D)$) takes into account the utility of I in all the transactions where I appears. To alleviate such over-estimation in the problem of mining FHUIs, another specialized form of HUIM is to restrict the cover of the candidate itemset I to only the transactions in which the utility of I (i.e., $u(I, T_i)$) is greater than a local minimum utility threshold. Such HUIs will be called *frequent local high utility itemsets* (FLHUIs, for short). To define such sets, we need the following additional terminology.

▶ **Definition 2.** *Assume D is a transaction database, and θ' a local minimum utility threshold. Then, the utility-based cover of an itemset I in D is defined as:* $\texttt{Cover}_u(I, D) = \{(i, T_i) \in D,\ I \subseteq T_i,\ \text{and } u(I, T_i) \geq \theta'\}$. *Then, the utility-based support of I is the cardinality of its utility-based cover, i.e.,* $\texttt{Supp}_u(I, D) = |\texttt{Cover}_u(I, D)|$.

▶ **Property 3.** *Let D be a transaction database, I an itemset, and δ a minimum support threshold. Then, if* $\mathtt{Supp}_u(I, D) \geq \delta$*, then* $\mathtt{Supp}(I, D) \geq \delta$*.*

Given a support and local minimum utility thresholds δ and θ' respectively, an itemset I is a FLHUI in D iff. $u(I, D) \geq \theta'$.

▶ **Example 4.** Let us consider again Example 1. For the minimum support threshold $\delta = 2$ and the local minimum utility threshold $\theta' = 10$, the sets $\{a, b\}$ and $\{a, b, g\}$ are FLHUIs.

To avoid ambiguity, we will refer to the second problem of enumerating all FLHUIs from transaction databases as FLHUIM. This paper deals with a suitable reduction of both the problems of FHUIM and FLHUIM to the propositional satisfiability model enumeration task, and then the use of state-of-the-art SAT solvers to solve these two problems.

2.2 Propositional logic

Let $\mathscr{L}$ be a propositional language built up inductively from a countable set $\mathtt{PS}$ of propositional variables, the boolean constants $\top$ (*true* or 1) and $\bot$ (*false* or 0) and the classical logical connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ in the usual way. We use the letters x, y, z, etc. to range over the elements of $\mathtt{PS}$. Propositional formulas of $\mathscr{L}$ are denoted by Φ, Ψ, etc. A literal is a propositional variable (x) of $\mathtt{PS}$ or its negation ($\neg x$). A clause is a (finite) disjunction of literals. For any formula Φ from $\mathscr{L}$, $\mathscr{P}(\Phi)$ denotes the symbols of $\mathtt{PS}$ occurring in Φ. A formula in conjunctive normal form (CNF, for short) is a finite conjunction of clauses. A Boolean interpretation Δ of a CNF formula Φ is defined as a function from $\mathscr{P}(\Phi)$ to $\{0, 1\}$. A model of a formula Φ is an interpretation Δ that satisfies Φ, i.e., if there exists an interpretation $\Delta : \mathscr{P}(\Phi) \rightarrow \{0, 1\}$ that satisfies all clauses in Φ. The formula Φ is satisfiable if it has at least one model. In the sequel, we write $\models$ for the logical consequence relation and $\models_{\mathtt{UP}}$ for the consequence relation restricted to the application of unit propagation[2].

The propositional satisfiability problem (SAT, for short) is a decision problem used to solve constraint satisfaction problems. Specifically, given a CNF formula Φ, SAT determines whether exists a model for each clauses in Φ. The application of SAT solvers in a range of real-world scenarios, e.g., electronic design automation, software and hardware verification [25], data mining [4], overlapping community detection in networks [17–19], has resulted from the progress of this NP-Complete problem over the previous decade. SAT technology has widely been used mainly in decision problems and its extensions such as SAT Modulo Theory (SMT), Maximum Satisfiability (Max-SAT), Quantified Boolean Formulas (QBF) but also recently in model enumeration problems built on top of modern SAT solvers such as Conflict Driven Clause Learning (CDCL) solver.

3 Computing High Utility Itemsets with Propositional Satisfiability

3.1 A SAT Approach to Frequent High Utility Itemset Mining

In this subsection, we deal with the translation of the FHUIM problem into propositional logic, so that SAT solvers can be used to enumerate FHUIs from transaction databases. We recall first that the traditional HUIM task has been recently reduced to SAT [15]. Specifically, in order to have a one-to-one mapping between the set of HUIs and the models of the

[2] Unit propagation is a kind of inference technique based on resolution with unit clauses (i.e., clauses containing exactly a single literal), e.g., $\Phi \wedge x \wedge (\neg x \vee \alpha) \models_{\mathtt{UP}} \alpha$.

underlying propositional formula, a set of propositional variables and logical constraints have been introduced. More formally, given a transaction database D, the proposed encoding associates to each item a (resp. transaction identifier i) of D a propositional variable referred to as p_a (resp. q_i). Figure 1 depicts the different constraints that accomplish the SAT-based encoding scheme of the HUIM problem. To be precise, Constraint 1 encodes the candidate itemset's cover. This constraint expresses the presence of the itemset in the i^{th} transaction, i.e., $q_i = 1$. More specifically, the candidate itemset is not supported by the i^{th} transaction (i.e., q_i is *false*), when there exists an item a (i.e., p_a is *true*) that does not belong to the transaction ($a \in \Omega \backslash T_i$); when q_i is *false*. This means that at least one item not appearing in the transaction i is set to *true*. Constraint (3) captures the closedness requirement of HUIs. It ensures that if the candidate itemset is involved in all transactions containing the item $a \in \Omega$, then a must be added to the itemset (i.e., a must be propagated to *true*). The constraint over the utility of the candidate itemset in the database D is expressed using the linear inequality (4) w.r.t. the user threshold θ. Notice that Constraint (4) takes into account the TWU property to prune the search space.

In contrast to the work of [15], we tackle in this paper the problem of mining the set of all FHUIs from transaction databases. To do this, the previous encoding (Constraints (1), (3), and (4)) should be extended with Constraint 2. In fact, Constraint (2) requires that at least δ transactions involves the candidate itemset to be considered as a frequent HUI. Consequently, the problem of FHUIM is encoded in propositional logic with the propositional formula $\Phi^{\texttt{fhuim}} = (1) \wedge (2) \wedge (4)$. Moreover, a user can be interested in a more concise representation of FHUIs, called *closed* FHUIs (in short CFHUIs). Recall that the closedness constraint is encoded as Constraint 3. We shall note the formula encoding the computation of CFHUIs as $\Phi^{\texttt{cfhuim}} = \Phi^{\texttt{fhuim}} \wedge (3)$.

$$\bigwedge_{i=1}^{m} (\neg q_i \leftrightarrow \bigvee_{a \in \Omega \backslash T_i} p_a) \quad (1) \qquad \sum_{i=1}^{m} q_i \geq \delta \quad (2) \qquad \bigwedge_{a \in \Omega} (p_a \vee \bigvee_{a \notin T_i} q_i) \quad (3)$$

$$\sum_{i=1}^{m} \sum_{a \in T_i} u(a, T_i) \times (p_a \wedge q_i) \geqslant \theta \quad (4)$$

Figure 1 SAT-based Encoding Scheme for the FHUIM Problem.

▶ **Proposition 5.** *Let D be a transaction database, θ a minimum high utility threshold, and δ a minimum support threshold. Let $\Phi^{\texttt{cfhuim}} = \Phi^{\texttt{fhuim}} \wedge (3)$ be a propositional formula. Then, there exists a one-to-one mapping between the models of $\Phi^{\texttt{cfhuim}}$ and the set of CFHUIs in D.*

3.2 A SAT Approach to Frequent Local High Utility Itemset Mining

This subsection presents our formulation of the problem of FLHUIM into propositional satisfiability. First, recall that a FLHUI in a transaction database D is a local high utility itemset and it meets a minimum support threshold. In contrast to the previous SAT-based encoding (see Figure 1), three subsets of propositional variables are introduced, namely $\{p_a,\ a \in \Omega\}$, $\{q_i,\ i \in [1..m]\}$, and $\{r_i, i \in [1..m]\}$. Afterwards, to restrict the frequency metric to the transactions with highest gain value, we consider the new Constraint (5). Specifically, this constraint states that an itemset I is covered by a transaction T_i if it is contained in T_i and the utility of I in T_i meets the required utility threshold. Alternatively, Constraint (5) can be rewritten as the following formula: $\bigwedge_{i=1}^{m} r_i \leftrightarrow \sum_{a \in T_i} u(a, T_i)(q_i \wedge p_a) \geq \theta'$. Now, to constrain the candidate itemset to be frequent, i.e., to be covered by at least δ transactions, we add the cardinality constraint (Constraint (6)). If that is the case, we call such candidate itemset as a FLHUI.

$$\bigwedge_{i=1}^{m}(r_i \leftrightarrow q_i \wedge (\sum_{a \in T_i} u(a,T_i)\ p_a \geq \theta')) \quad (5) \qquad\qquad \sum_{i=1}^{m} r_i \geq \delta \quad (6)$$

Figure 2 SAT Encoding Scheme for Frequent Local High Utility Itemset Mining Problem.

▶ **Proposition 6.** *Let D be a transaction database, θ' a local minimum utility threshold, and δ a minimum support threshold. Let $\Phi^{\texttt{cflhuim}} = (1) \wedge (5) \wedge (6) \wedge (3)$ be a propositional formula. Then, there exists a one-to-one mapping between the models of $\Phi^{\texttt{cflhuim}}$ and the set of closed FLHUIs in D.*

▶ **Example 7.** Let us consider the transaction database depicted by Table 1. Then, the formula that encodes the problem of enumerating all closed FLHUIs in D with $\theta' = 10$ and $\delta = 2$ is written as follows:

$$\begin{array}{lcll}
\neg q_1 & \leftrightarrow & (p_c \vee p_d \vee p_e \vee p_f) & p_a \vee q_2 \vee q_3 \\
\neg q_2 & \leftrightarrow & (p_a \vee p_d \vee p_f \vee p_g) & p_b \vee q_3 \vee q_4 \\
\neg q_3 & \leftrightarrow & (p_a \vee p_d \vee p_f \vee p_g) & p_c \vee q_1 \vee q_4 \vee q_5 \\
\neg q_4 & \leftrightarrow & (p_b \vee p_c \vee p_f \vee p_g) & p_d \vee q_1 \vee q_2 \vee q_5 \\
\neg q_5 & \leftrightarrow & (p_c \vee p_d \vee p_e) & p_e \vee q_1 \vee q_3 \vee q_5 \\
r_1 & \leftrightarrow & q_1 \wedge (8p_a + 2p_b + p_g \geq 10) & p_f \vee q_1 \vee q_2 \vee q_3 \vee q_4 \\
r_2 & \leftrightarrow & q_2 \wedge (6p_b + 3p_c + 2p_e \geq 10) & p_g \vee q_2 \vee q_3 \vee q_4 \\
r_3 & \leftrightarrow & q_3 \wedge (4p_c + 3p_d \geq 10) & \\
r_4 & \leftrightarrow & q_4 \wedge (6p_a + 4p_d + p_e \geq 10) & \\
r_5 & \leftrightarrow & q_5 \wedge (8p_a + 7p_b + 2p_f + p_g \geq 10) & \\
\multicolumn{3}{l}{r_1 + r_2 + r_3 + r_4 + r_5 \geq 2} &
\end{array}$$

The SAT encodings of FHUIM and FLHUIM tasks involve the so-called Pseudo-Boolean constraints[3] (e.g. Constraints (2), (4) and (6)). Solving such kind of constraints has received an important attention by the SAT community since Pseudo-Boolean constraints naturally arise in many propositional encodings of real-world problems, including classical pattern mining, product configuration and community discovery in networks. A common way to solve Pseudo-Boolean constraints is by transformation into a SAT equivalent propositional formula and then use SAT solvers in order to verify satisfiability using various state-of-the-art encoding techniques [11]. Another way is to handle Pseudo-Boolean-constraints directly in the SAT solver [1]. Pseudo-Boolean problems can also be modeled as an integer program, in which the nonlinear constraints are linearized like [6] which used cutting resolution. The solver bsolo [23] also combines integer programming techniques with SAT-solving. On the other hand, cutting resolution has also been used to solve Pseudo-Boolean constraints [6]. Conflict analysis, introduced by Marques-Silva and Sakallah [24], is an important component of modern SAT-solvers. It allows SAT solvers to learn conflict clauses from intractable sub-problems. These clauses allow the solver to prune other branches of the search tree and use non-chronological backtracking. However, its extension to Pseudo-Boolean constraints is not obvious [5,8].

In our case, each transaction gives rise to a Pseudo-Boolean constraint. Moreover, we deal with an enumeration based problem. Consequently, due to some scalability concerns, we choose to manage these constraints lazily as in [21]. As described in Constraint (6), the PB

[3] A Pseudo-Boolean constraint is an expression of the form $\sum_{i=1}^{n} a_i x_i \ op\ b$, where a_i and b are real coefficients, x_i $(1 \leq i \leq n)$ are propositional variables, and the operator op belongs to $\{=, \leq, <, >, \geq\}$.

constraint allows to catch the conditions under which r_i is propagated particularly to *false*: either when q_i is *false*, or when the utility of the candidate itemset in T_i is less than the fixed utility threshold θ'. The latter is managed thanks to the use of counters that allow to check if the sum of the utilities of items in the transaction T_i not assigned to false is less than θ'.

3.3 SAT-based Enumeration for High Utility Mining

In this subsection, we present our method based on the classical DPLL procedure [7] to enumerating the set of FHUIs and FLHUIs in transaction databases. We use a DPLL procedure to avoid adding blocking clauses and then to circumvent the growing up of the sub-formulas size. The enumeration is performed by a simple backtrack at each found model. This is motivated by the fact that the number of models can be large particularly in pattern mining. Algorithm 2 (See appendix) illustrates the pseudo-code of the enumeration process. As mentioned previously, since our encoding includes both clauses and Pseudo-Boolean constraints, the latter are managed lazily following [21]. As will be shown in the empirical evaluation, the encoding of all Pseudo-Boolean constraints into CNF can make the resolution inefficient for large databases, since each variable r_i implies a Pseudo-Boolean constraint, which can be huge depending on the number of transactions in the database. This can make the SAT-based encoding intractable in practice. To address this issue, we propose a slight modification of the DPLL-based algorithm by employing *propagators*, a key component of CSP and SMT (Satisfiability Modulo Theory) solvers. The propagator is based on counters. In fact, it proceeds in the same manner as the well-known TWU measure used in the HUIM literature. When the propositional variable p_a becomes *false*, the utility of the item a is subtracted from the sum of the utility of the sub-tables in which the item a appears. Propagators are used in Constraint (2). We use counters to detect when the total weight of a transaction is less than the fixed threshold. If it is the case, then the variable q_i is propagated to *false*. As a result, our algorithm is divided into two parts: unit propagation for the propositional part and propagators for dealing with Pseudo-Boolean constraints to handle frequency and utility based constraints. Furthermore, propagators' primary function is to check the satisfiability of the Pseudo-Boolean constraint. The idea behind using propagators within our DPLL based approach is mainly to check the consistency of the Pseudo-Boolean constraints or to infer useful propagation. More precisely, for a Pseudo-Boolean constraint of the form $\sum_{i=1}^{n} w_i x_i \geq k$, a counter is initialized with the value $w = \sum_{i=1}^{n} w_i$. Each time an x_i is assigned to *false*, its corresponding weight w_i is subtracted from w leading to $w - w_i$. If such value is less than k, then a backtrack is performed. Note that the FHUIM task involves a cardinality and a Pseudo-Boolean constraints, while in the FLHUIM problem we can rewrite the encoding using conditional Pseudo-Boolean Constraints of the form $y \to \sum_{i=1}^{n} w_i x_i \geq k$ [3]. Such constraints allows to capture conditions under which $\neg y$ can be propagated. It is also worth noting that assigning the variables representing items (p_a, $a \in \Omega$) allows to fix the remaining variables via propagation, i.e., p_a, $a \in \Omega$ is a *strong backdoor* [31]. Then, from an heuristic point of view, it is better to assign such variables first.

3.4 A Decomposition-based Approach for FHUIM & FLHUIM

In practice, the DPLL based enumeration algorithm suffers from scalability issues, particularly when the size of the propositional encoding is very large. This can have a significant impact on the approach's efficiency, as stated by [15]. In fact, without decomposition the number of clauses of the encoding is equal to $|\Omega| \times |D| - (\sum_{T_i \in D} |T_i|)$. This is equivalent to the number of missing items in the database. This value can be very huge for large datasets (.e.g., for

Kosarak for $\delta = 1000$, the number of non missing items is 34009483, then the number of clauses that represent Constraint (5) is $41270 \times 990002 - 34009483$, which exceeds 40 billion clauses). To address this issue, we apply a decomposition scheme to split the transaction databases into numerous bases of reasonable size. Using decomposition, the size of each sub-problem is significantly reduced. For instance, Constraint (5) leads to 31205214 clauses for *Kosarak*, which is the total number of clauses of the different sub-problems instead of 40 billion clauses. The main idea of decomposition is to avoid encoding the entire database in favor of solving many independent sub-problems of small size rather than a single large problem. Basically, given a propositional formula Φ and a variable $x_1 \in \mathtt{PS}(\Phi)$, the models of Φ are those of $\Phi \wedge x_1$ and $\Phi \wedge \neg x_1$. By generalizing such principle for a subset of variables $\{x_1, \dots, x_n\}$, the models of Φ are those of $\Psi_1, \dots, \Psi_n$ where $\Psi_i = \Phi \wedge x_i \wedge \bigwedge_{1 \leq j < i} \neg x_j$. In our case, we have $\{x_1, \dots, x_n\} = \{p_{a_1}, \dots, p_{a_n}\}$ and Φ corresponds to $\Phi^{\mathtt{fhuim}}$ (or $\Phi^{\mathtt{cfhuim}}$ for closed patterns) or $\Phi^{\mathtt{flhuim}}$ (or $\Phi^{\mathtt{cflhuim}}$ for closed patterns). Hence, solving $\Psi_i = \Phi \wedge p_{a_i} \wedge \bigwedge_{1 \leq j < i} \neg p_{a_j}$ can be obtained by considering only transactions containing the item a_i. In fact, since the variable p_{a_i} is *true*, the models of Φ_i is restricted to those containing p_{a_i}, which means the itemset including a_i. Clearly, splitting the CNF formula generates a set of independent sub-formulas that encode subsets of a specific subset of transactions of the original database. Consequently, this allows to avoid modeling the entire database and without causing too large number of clauses as well as the associated computational problems. It is important to note here that the order in which we solve the generated sub-problems has further a tremendous impact on the effectiveness of our approach. In particular, we believe that starting from the last sub-problem $\Phi \wedge \Psi_n$ is the best choice since it is the simplest one. In fact, all the variables representing items are assigned to *false* except one. This results to a smaller encoding size for the current sub-problem compared to the previous sub-problems. Interestingly, the declarativity of our SAT approach is preserved when applying the decomposition technique. Thus, one just need to add the user-specific constraints to each sub-formula encoding the sub-table obtained by decomposition.

▶ **Example 8.** Let us reconsider the transaction database in Table 1. Figure 3 depicts the sub-tables obtained by applying the decomposition principle.

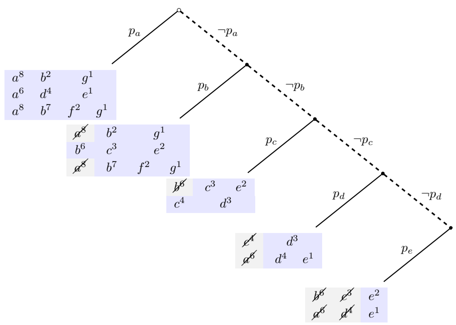

Figure 3 Item-based decomposition tree of the database in Table 1.

Algorithm 1 depicts our decomposition-based algorithm for mining frequent (local) high utility itemsets from transaction databases. This algorithm takes a transaction database, a (local) minimum utility threshold and a minimum support threshold as input, and returns all (closed) frequent (local) high utility itemsets. Based on the previously stated decomposition principle, our algorithm splits the transaction table into multiple independent sub-problems and restricts the encoding to a sub-table each time in order to enumerate all models corresponding to motifs of interest using the enumeration procedure described in Algorithm 2. To reduce the search space and, probably the encoding size, a pre-processing process is used to prune all itemsets that cannot be included in the final output. More specifically, in both FHUIM and FLHUIM, infrequent items are ignored. For the FHUIM method, if $TWU(a, D) < \theta$, then the item a is discarded, whereas for FLHUIM task, if $\sum_{a \in T} u(a, T) < \theta'$ with $T \in D$, then the transaction is not considered to be part of the search space.

Algorithm 1 **SAT** based **F**requent (Local) **H**igh **U**tility **I**temset **M**ining Approach.

Input: D: a transaction database, θ: utility threshold, δ: support threshold
Output: S: the set of frequent (local) high utility itemsets

$S \leftarrow \emptyset$;
for $i \in [1..n]$ **do**
  if $\mathtt{Supp}_u(a_i, D) \geq \delta$ **then**
    $D_i \leftarrow \{(k, T_k) \in D \mid a_i \in T_k\}$, $\Omega = \langle a_1, \ldots, a_n \rangle \leftarrow items(D_i)$, $\Gamma \leftarrow \emptyset$;
    for $T_j \in D_i$ **do**
      if $\sum_{c \in T_j} u(c, T_j) < \theta$ **then**
        $\Gamma \leftarrow \Gamma \wedge \neg r_i$, $D_i \leftarrow D_i \setminus \{T_j\}$;
      end
    end
    for $b \in items(D_i)$ **do**
      if $\mathtt{Supp}_u(b, D_i) < \delta$ **then**
        $\Gamma \leftarrow \Gamma \wedge \neg p_b$;
      end
    end
    $\Psi \leftarrow p_{a_i} \wedge \bigwedge_{1 \leq j < i} \neg p_{a_j}$;
    $\Phi \leftarrow \Phi^{fhuim}(D_i, \theta, \delta) \wedge \Psi \wedge \Gamma$; /* Φ^{flhuim} for FLHUIM */
    $S \leftarrow S \cup \mathtt{DPLL\_Enum}(\Phi)$
  end
end
return S;

4 Empirical Investigation

4.1 Experimental setup

Algorithm 1 is implemented in C++ language top-on the SAT solver MiniSAT [10], which is adapted to compute all models of a propositional formula by performing a DPLL procedure [7] as explained above. Our motivation here relies on the fact that we face on the problem of enumerating a huge number of models. For this, we adapt MiniSAT solver by keeping watched literals for unit propagation. Obviously, the restart and clause learning components can be disabled for better scalability and also to avoid growing the sub-formulas size. Note that the decomposition is performed by considering the frequency of items in ascending order, and the resulting sub-problems are addressed in a sequential manner. We have compared our proposed approaches against two baselines, namely `HU-FIMi` [28] and `FUFM` [27][4]. In our empirical evaluation, we conduct experiments over different commonly used benchmark datasets in the HUIM setting. These datasets are downloaded from the open-source data mining library SPMF [12]. All characteristics of both real and synthetic datasets are summarised in Table 4: the number of transaction (#Trans), the number of items (#Items), the average length of transactions (AvgTransLen), and the density (Density(%) for each dataset (see appendix).

Our experiments were performed on a machine with Intel Xeon quad-core processors with 32GB of RAM running at 2.66 GHz on Linux CentOS. Timeout was set to two hours for each run of an algorithm on a dataset. All experiments were conducted by varying the minimum support (δ) and the minimum high utility (θ) thresholds. It should also be mentioned that for our proposed algorithms, the computation time includes both the time needed for generating the CNF formulas and that for computing all models (i.e., itemsets of interest) of these formulas. We also note that the reported runtime in all the experiments is in seconds.

4.2 Results on Mining FHUIs

To evaluate the performance of our approach for mining frequent high utility itemsets, we consider a representative sample of real-world datasets (*Chess*, *Retail*, *Kosarak*, and *Chainstore*). We compared our `SATFHUIM` algorithm to the existing method named `HU-FIMi` [28][5]. Our approach is compared to this baseline according to running time and memory consumption for different minimum support and utility thresholds. Table 2 summarizes the empirical performance of our method against `HU-FIMi` on each dataset for each θ and δ values. Notice that the symbol (–) means that the algorithm is not able to complete the mining process under the fixed time out (i.e., TO). The size of FHUIM encoding in terms of the number of variables (#Var) and clauses (#Clauses) is given in Table 2.

According to our experimental results, our method outperforms the baseline across all datasets. In fact, `SATFHUIM` achieves interesting results on all databases when δ and θ are varied. For *Retail*, *Chainstore* and *Kosarak* datasets, `SATFHUIM` was respectively up to 39, 30 and 70 times faster than `HU-FIMi`. It can also be observed that on *Chess* dataset, `HU-FIMi` was enable to mine the target itemsets under the timeout except for $\delta = 50\%$ and $\theta = 400$k where the baseline took more than 4000 seconds to mine all FHUIs. However, for the same dataset `SATFHUIM` is able to scale for all minimum support and utility threshold values with a maximum running time of 1300 seconds. In terms of memory usage, `SATFHUIM` performs very well on both dense and sparse datasets, except for *Retail* and *Kosarak* datasets. This can be explained by the fact that even if the size of the generated sub-bases is small, the sub-problems could be numerous for these two datasets.

In our experiments, we also investigate the behavior of our SAT-based proposal `SATFHUIM` w.r.t. the running time and the number of FHUIs while varying both the values of θ and δ thresholds. The empirical results are depicted in Figures 4 and 5. The results show that `SATFHUIM` is able to solve all datasets even for small support and utility thresholds values where the number of obtained itemsets is huge. As illustrated in Figure 4, it is clear that the performance of our algorithm depends on the overall dataset characteristics. In addition, the minimum support δ as well as the minimum utility θ thresholds have a strong impact on the performance of the mining process. Specifically, for low values of θ and δ, `SATFHUIM` needs more time to discover all itemsets. The density value also has an impact on the execution

[4] We have used the C++ implementation for `HU-FIMi`, and the Python implementation for `FUFM`.
[5] We did not provide a comparison with `FCHUIM` because the source code is not public.

Table 2 Experimental results using different values of θ and δ.

			SATFHUIM					HU-FIMi		
Dataset	**δ(%)**	**θ**	**Time(s)**	**Memory (MB)**	**#conf**	**#Var**	**#Clauses**	**Time(s)**	**Memory (MB)**	**#cand**
Chess	30	200k	**1341.21**	**118**	5417007	75	4003366	–	–	–
		300k	**380.89**	**117**	3722143		4003184	–	–	–
		400k	**250.98**	**118**	539285		4002718	–	–	–
	40	200k	**363.171**	**117**	673551	75	3240897	–	–	–
		300k	**192.04**	**117**	1071967		3240897	–	–	–
		400k	**47.32**	**117**	426604		3240870	4022.47	149.16	1064837
	50	200k	**102.33**	**87**	103010	75	2930270	–	–	–
		300k	**70.14**	**87**	212158		2930270	3650.06	169.83	2485694
		400k	**28.15**	**87**	170565		2930243	1786.03	96.87	634997
Retail	0.2	1k	**1.83**	290	3167	16470	5835070	195.87	**145.44**	95912
		8k	**1.6**	207	1529		5500626	100.68	**146.92**	5877
		30k	**3.7**	156	1049		5084829	46.52	**131.91**	958
	0.4	1k	**0.96**	125	163	16470	2696209	61.48	**123.78**	30167
		8k	**0.92**	125	325		2650864	36.74	**123.22**	2652
		30k	**0.81**	124	402		2520038	18.95	**122.71**	482
	0.6	1k	**1.99**	125	163	16470	1785564	30.02	**118.44**	14591
		8k	**0.71**	**105**	121		1764079	20.64	117.87	1624
		30k	**0.64**	**105**	217		1710400	12.99	118.4	318
Kosarak	0.2	200k	**38.91**	**3011**	21977	41270	117031877	–	–	–
		400k	**32.20**	3004	4887		115627789	5348.66	**876.24**	191819
		600k	**28.71**	2986	1425		111434918	4002.07	**870.56**	75344
	0.4	200k	**18.23**	1514	1174	41270	49791548	1231.01	**860.74**	56174
		400k	**17.10**	1516	1306		49636024	849.15	**858.98**	18521
		600k	**16.03**	1509	816		49166188	652.24	**856.3**	7198
	0.6	200k	**12.79**	1154	446	41270	31646467	540.8	**839.82**	16552
		400k	**12.37**	1153	594		31594030	385.69	**826.5**	5531
		600k	**11.78**	1156	514		31498639	294.35	**826.89**	2358
Chainstore	0.2	1M	**7.8**	**754**	234	46086	15404456	245.79	939.4	1095
		1.4M	**7.5**	**752**	288		15184184	208.97	933.56	670
		1.8M	**7.19**	**747**	313		14899579	462.76	932.04	500
	0.4	1M	**5.44**	**523**	42	46086	7915721	123.13	896.05	503
		1.4M	**5.34**	**523**	69		7915721	117.42	877.91	346
		1.8M	**5.17**	**523**	89		7915721	269.54	887.14	240
	0.6	1M	**4.24**	**465**	10	46086	4481837	81.33	841.47	292
		1.4M	**4.16**	**465**	15		4481837	77.11	839.75	181
		1.8M	**4.08**	**465**	22		4481837	77.15	838.29	141

time. To be precise, `SATFHUIM` becomes slow on dense datasets, for instance it takes more than 1000 seconds to find all FHUIs for *Chess* dataset. In contrast, on sparse datasets, it is become easier to compute all FHUIs even for low thresholds values. This is the case for *Chainstore* dataset where the time needed to enumerate all patterns is only 80 seconds for low threshold values.

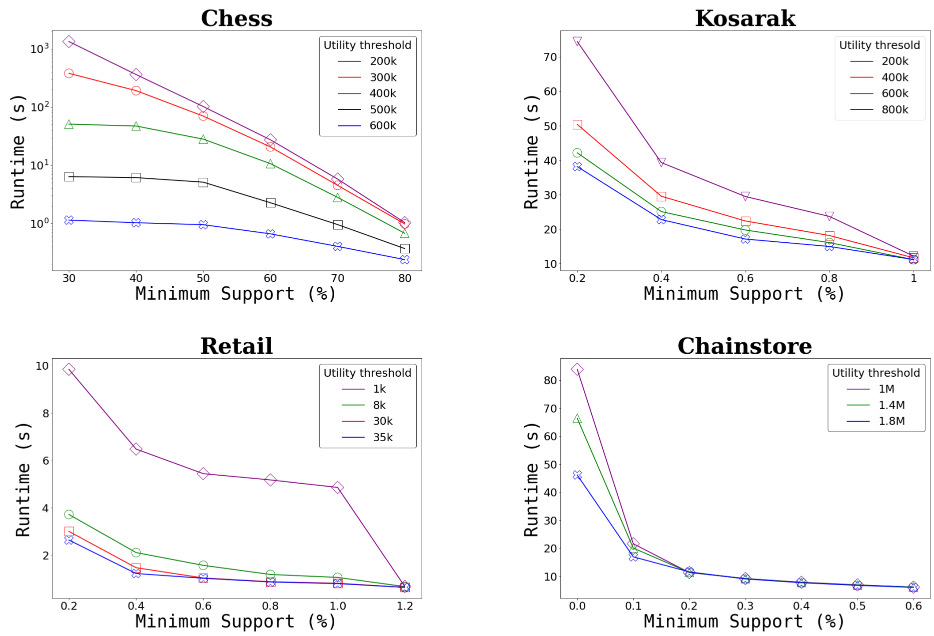

Figure 4 Running time of `SATFHUIM` w.r.t. minimum support threshold on real-world datasets.

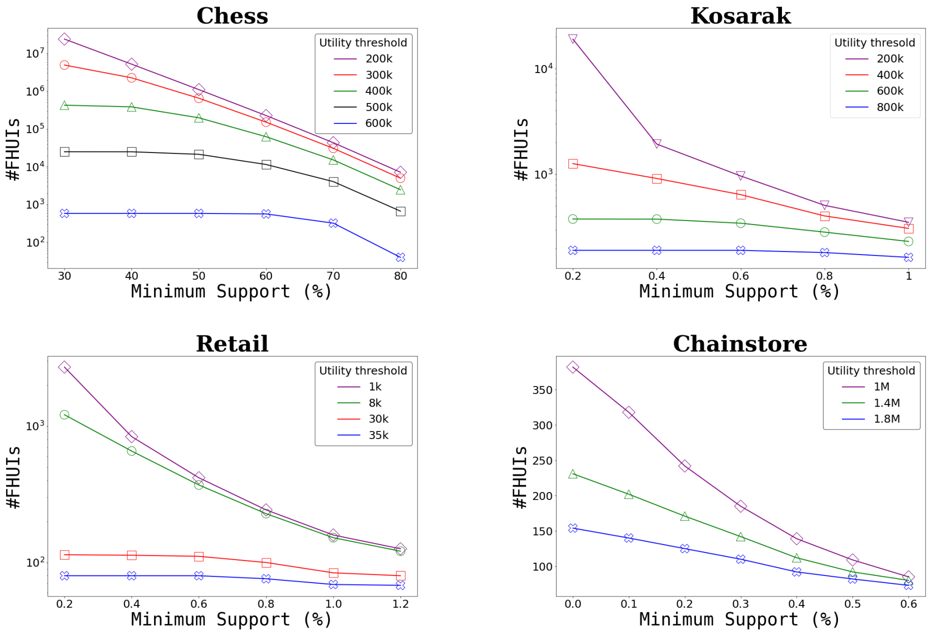

Figure 5 Number of **FHUIs** w.r.t. minimum support threshold on real datasets.

According to Figure 5, it is clear that the number of FHUIs always depends on the chosen values of θ and δ. In fact, for lowest threshold values this number becomes huge even for small datasets. For instance, on *Chess* and *Retail* the number can be more than 10^7 and 10^4, respectively, but still clearly small compared with the number of itemsets generated by HUIM algorithms[6] as the support constraint allows to discard an important number of patterns. Overall, we note that the number of FHUIs is always less to the number of HUIs.

4.3 Results on Mining FLHUIs

The second experiment was conducted to evaluate the performance of our **SATFLHUIM** algorithm and compare it with the state-of-the-art method **FUFM** [27]. This experiment was carried on the real datasets (i.e., *Chess*, *Retail*, *Mushroom*, *Accidents*, and *Chainstore*) and also on a synthetic one called *T60D10kI1k*. Note that *T60D10kI1k* was constructed using the transaction database generator in SPMF [12] (see Table 4 for the characteristics of this dataset). For this dataset, the internal (resp. external) utility values was generated using a uniform distribution in range [1,10] (resp. [1,6]). The parameters θ' and δ values were varied for the different datasets. Similarly to the previous experiment, we compare our approach against the baseline **FUFM** on both running time and memory usage for mining FLHUIs, followed by the number of generated FLHUIs. As our **SATFLHUIM** algorithm allows us also to mine closed FLHUIs, we add in the last column of Table 3 the number of closed FLHUIs (#CFLHUIs). Table 3 shows in addition the encoding size in terms of the number of variables (#var) and clauses (#Clauses). All experimental results are shown in Table 3. According to this latter, both algorithms produced the same output. On running time, **SATFLHUIM** performs well on all of datasets. Interestingly enough, our method outperforms the baseline **FUFM** for low values of δ and θ' where the number of generated FLHUIs is large. For instance, on *Chess* dataset and $\theta' = 100$ and $\delta = 70\%$, **FUFM** takes more than 1 hour to

[6] If the minimum support threshold $\delta = 1$, the set of FHUIs correspond exactly to the set of HUIs.

find all FLHUIs whereas `SATFLHUIM` does not exceed 2 seconds for the same task. This is primarily due to the fact that `FUFM` is a candidate generation-based method with a large number of candidates. Furthermore, as the support and minimum utility thresholds decrease, `FUFM`'s runtime increases dramatically. From a memory usage point of view, there is a gap in terms of memory between `SATFLHUIM` and `FUFM`. For instance, `FUFM` consumes up to 18 times more memory than `SATFLHUIM` for *Chess* dataset. Roughly speaking, when the parameters δ and θ' decrease, the overall performance of the two algorithms begin to decrease, and vice versa. When compared to the number of FLHUIs, it can be seen that the number of closed FLHUIs does not decrease significantly for almost datasets, except for *Mushroom* and *Chess* where the number of closed FLHUIs decreases to half w.r.t. the number of FLHUIs.

Table 3 Experimental results using different values of θ' and δ.

			SATFLHUIM				FUFM			
Dataset	**δ(%)**	**θ'**	**Time(s)**	**Memory (MB)**	**#Var**	**#Clauses**	**Time(s)**	**Memory (MB)**	**#FLHUIs**	**#CFLHUIs**
		100	**2.51**	**68**		1594175	3784.971	1469	17201	9167
	70	130	**1.36**	**68**	75	1590608	3741.68	1214	6004	3469
		150	**0.83**	**68**		1587442	3752.159	1071	2197	1378
		100	**1.04**	**68**		1476874	1434.251	577	5347	3343
Chess	75	130	**0.57**	**68**	75	1472986	1420.615	530	1366	959
		150	**0.37**	**68**		1469384	1431.31	420	350	277
		100	**0.42**	**54**		1165792	541.532	259	1176	880
	80	130	**0.25**	**54**	75	1161235	540.855	271	147	129
		150	**0.19**	**54**		1157227	539.979	279	9	9
		10	**0.4**	**104**		814001	3108.718	186	39	39
	2	15	**0.39**	**104**	16470	785221	3139.677	186	31	31
		20	**0.38**	**104**		763051	3106.071	186	28	28
		10	**0.32**	**96**		554031	372.061	155	16	16
	4	15	**0.3**	96	16470	531227	375.671	155	14	14
		20	**0.3**	**96**		514102	376.369	155	13	13
Retail		10	**0.29**	**96**		494337	256.184	148	14	14
	6	15	**0.29**	**95**	16470	473078	254.18	148	12	12
		20	**0.27**	**95**		456605	255.564	148	7	7
		10	**0.3**	**96**		494337	232.043	145	12	12
	8	15	**0.28**	**95**	16470	473078	230.931	145	8	8
		20	**0.28**	**95**		456605	230.95	145	1	1
		30	**0.67**	91		2205546	88.592	**59**	2286	383
	30	40	**0.64**	91	119	2200604	88.75	**65**	1869	343
		50	**0.61**	91		2193490	88.957	**65**	1390	282
		30	**0.32**	72		1554782	46.279	55	268	84
Mushroom	40	40	**0.31**	72	119	1548637	45.725	**55**	146	53
		50	**0.28**	72		1539823	46.191	**59**	59	30
		30	**0.15**	58		806424	17.389	**49**	45	19
	50	40	**0.15**	58	119	799408	17.575	**46**	18	11
		50	**0.16**	58		790047	17.316	**47**	2	2
		40	**21.46**	**2544**		69675386	6798.121	3178	133	133
	70	45	**19.92**	**2555**	468	69542270	6950.867	3177	86	86
		48	**19.23**	**2553**		69466628	6841.843	3176	67	67
		40	**14.97**	**1977**		55728869	4190	3893	42	42
Accidents	75	45	**14.44**	**1977**	468	55599338	4216.008	3892	24	24
		48	**14.16**	**1976**		55523819	4259.47	2335	17	17
		40	**10.15**	1851		36447084	2238.2	**1051**	13	18
	80	45	**9.96**	1844	468	36271268	2170.082	**1488**	4	4
		48	**9.96**	1843		36177935	2133.187	**1488**	2	2
		100	**10.49**	**929**		18210234	777.52	1257	407	407
	0.2	500	**9.84**	**909**	46086	16437597	775.13	1257	51	51
		800	**9.61**	**903**		16199306	768.64	1259	10	10
		100	**6.22**	**769**		9475966	530.86	1263	142	142
Chainstore	0.4	500	**5.94**	**672**	46086	8255824	531.46	1265	18	18
		800	**5.91**	**668**		8101301	515.35	1266	3	3
		100	**4.26**	**676**		5617273	206.81	1256	79	79
	0.6	500	**4.08**	**611**	46086	4711977	212.15	1256	9	9
		800	**4.02**	**608**		4596272	198.97	1256	1	1

Figure 6 provides the execution times on the different datasets in the first line followed by the variation of the number of patterns in the second line. Due to space limitation, we did not provide the results for all θ' and δ values. Instead, each bar corresponds to the average number of FLHUIs for each δ threshold in terms of the average of all fixed θ' values.

According to these experimental results, the performance of our proposal is highly dependent on the dataset characteristics, and also on the thresholds values chosen. In fact, when θ' and δ are set to large values the runtime is quite similar for almost all datasets. This is understandable as the number of discovered patterns is small. However, for small threshold values there is a gap between the runtimes. It is also worth noting that the output size (i.e., the set of FLHUIs) is significantly decreased when compared to our `SATFHUIM` algorithm w.r.t. θ' threshold values. For instance, on *Chess*, the average number of FLHUIs is about 21744362 for all fixed θ' values and $\delta = 30\%$, whereas the number of FHUIs is 24081372 for the same δ value and $\theta' = 200$k.

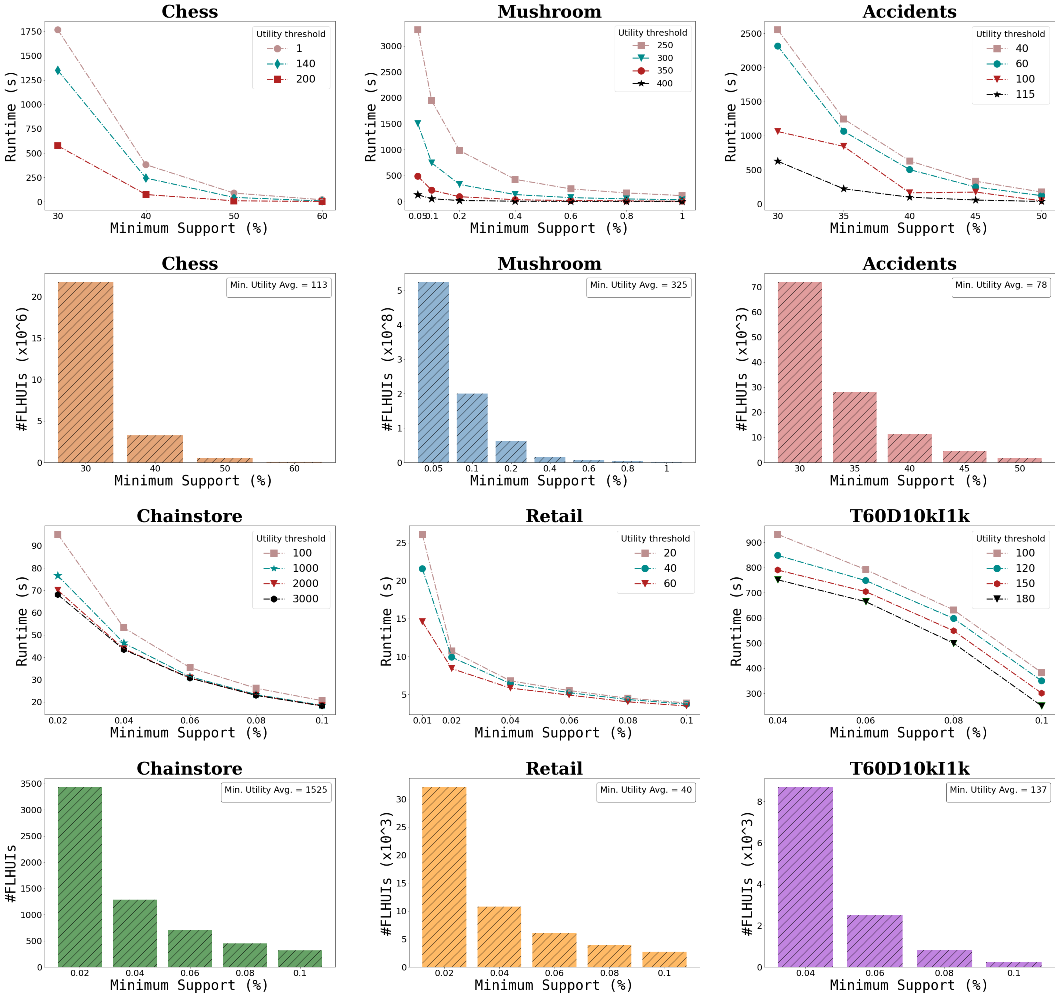

Figure 6 Experimental results of `SATFLHUIM` on several datasets.

5 Conclusion

In this paper we investigated how to solve the problem of mining (closed) FHUIs and FLHUIs from transaction databases using propositional logic. For the FHUIM task, we extended the existing approach of [15] with the frequency constraint, while for the FLHUIM problem we provided a new encoding using the well-known Pseudo-Boolean constraints. We extended the DPLL procedure to deal with both clauses and Pseudo-Boolean constraints in order to compute all models of CNF formulas. To scale up, a decomposition approach was presented, which allows the problem to be divided into several sub-problems of reasonable size. Empirical evaluation have shown how our approaches are very promising w.r.t. state-of-the-art.

In the future, we plan to investigate how to use propositional satisfiability to implement a limited but efficient clause learning in the context of patterns mining. In addition, by extending our approach for multi-objective optimization, we plan to investigate the problem of computing skyline HUIs from transaction databases using the two measures of interest (i.e., utility and frequency).

References

1 Fadi A Aloul, Arathi Ramani, Igor Markov, and Karem Sakallah. Pbs: a backtrack-search pseudo-boolean solver and optimizer. In *International Symposium on Theory and Applications of Satisfiability*, pages 346–353, 2002.

2 Mohamed-Bachir Belaid, Christian Bessiere, and Nadjib Lazaar. Constraint programming for mining borders of frequent itemsets. In *IJCAI*, pages 1064–1070, 2019.

3 Abdelhamid Boudane, Saïd Jabbour, Badran Raddaoui, and Lakhdar Sais. Efficient sat-based encodings of conditional cardinality constraints. In *LPAR*, pages 181–195, 2018.

4 Abdelhamid Boudane, Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. SAT-based data mining. *Int. J. Artif. Intell. Tools*, pages 1840002:1–1840002:24, 2018.

5 D. Chai and A. Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 305–317, 2005.

6 W. Cook, C.R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, pages 25–38, 1987.

7 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, pages 394–397, 1962.

8 Heidi E. Dixon and Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *AAAI*, pages 635–640, 2002.

9 Imen Ouled Dlala, Saïd Jabbour, Badran Raddaoui, and Lakhdar Sais. A parallel SAT-based framework for closed frequent itemsets mining. In *CP*, pages 570–587, 2018.

10 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2004.

11 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *J. Satisf. Boolean Model. Comput.*, pages 1–26, 2006.

12 Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. The spmf open-source data mining library version 2. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 36–40, 2016.

13 Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Vincent S. Tseng, and Philip S. Yu. A survey of utility-oriented pattern mining. *IEEE Transactions on Knowledge and Data Engineering*, pages 1306–1327, 2021.

14 Tias Guns, Anton Dries, Siegfried Nijssen, Guido Tack, and Luc De Raedt. Miningzinc: A declarative framework for constraint-based mining. *Artif. Intell.*, pages 6–29, 2017.

15 Amel Hidouri, Said Jabbour, Badran Raddaoui, and Boutheina Ben Yaghlane. Mining closed high utility itemsets based on propositional satisfiability. *DKE*, page 101927, 2021.

16 Saïd Jabbour, Fatima Ezzahra Mana, Imen Ouled Dlala, Badran Raddaoui, and Lakhdar Sais. On maximal frequent itemsets mining with constraints. In *CP*, pages 554–569, 2018.

17 Saïd Jabbour, Nizar Mhadhbi, Badran Raddaoui, and Lakhdar Sais. Triangle-driven community detection in large graphs using propositional satisfiability. In *AINA*, pages 437–444, 2018.

18 Saïd Jabbour, Nizar Mhadhbi, Badran Raddaoui, and Lakhdar Sais. Sat-based models for overlapping community detection in networks. *Computing*, 102(5):1275–1299, 2020.

19 Saïd Jabbour, Nizar Mhadhbi, Badran Raddaoui, and Lakhdar Sais. A declarative framework for maximal k-plex enumeration problems. In *AAMAS*, pages 660–668, 2022.

20 Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. Mining Top-k motifs with a SAT-based framework. *Artif. Intell.*, pages 30–47, 2017.

21 Daniel Le Berre and Anne Parrain. The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 59–64, 2010.

22 Ying Liu, Wei-keng Liao, and Alok Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 689–695, 2005.

23 Vasco Manquinho and J. Marques-Silva. On using cutting planes in pseudo-boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation*, 2006.

24 João P. Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.*, pages 506–521, 1999.

25 A Morgado and J Marques-Silva. Algorithms for propositional model enumeration and counting. Technical report, Citeseer, 2005.

26 A Sakthi Nathiarasan and M Manikandan. Performance oriented mining of utility frequent itemsets. In *International Conference on Circuits, Communication, Control and Computing*, pages 317–321, 2014.

27 Vid Podpecan, Nada Lavrac, and Igor Kononenko. A fast algorithm for mining utility-frequent itemsets. *Constraint-Based Mining and Learning*, page 9, 2007.

28 R Uday Kiran, T Yashwanth Reddy, Philippe Fournier-Viger, Masashi Toyoda, P Krishna Reddy, and Masaru Kitsuregawa. Efficiently finding high utility-frequent itemsets using cutoff and suffix utility. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 191–203. Springer, 2019.

29 Jilles Vreeken and Nikolaj Tatti. Interesting patterns. In *Frequent Pattern Mining*, pages 105–134. Springer, 2014.

30 Tianyou Wei, Bin Wang, Yuntian Zhang, Keyong Hu, Yinfeng Yao, and Hao Liu. FCHUIM: Efficient frequent and closed high-utility itemsets mining. *IEEE Access*, pages 109928–109939, 2020.

31 Ryan Williams, Carla Gomes, and Bart Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. *structure*, 23(4), 2003.

32 Jieh-Shan Yeh, Yu-Chiang Li, and Chin-Chen Chang. Two-phase algorithms for a novel utility-frequent mining model. In *Emerging Technologies in Knowledge Discovery and Data Mining*, pages 433–444, 2007.

A Appendix

Algorithm 2 DPLL_Enum: A DPLL backtrack search for Model Enumeration.

```
Input: Φ: a CNF formula
Output: S: the set of all models of Φ
Δ = ∅; S = ∅;
if (Φ ⊨_UP p) then
|   return DPLL_Enum(Φ ∧ p) ;                         /* unit clause */
end
if (Φ ⊨_UP ⊥) then
|   return ∅ ;                                        /* conflict */
end
if check_Pseudo_Boolean_constraint() == False then
|   return ∅;
end
if (Δ ⊨ Φ) then
|   S ← S ∪ {Δ} ;                                     /* new found model */
|   return ∅
end
p = select_variable(Var(Φ));
Δ ← Δ ∪ {p}; S ← S ∪ DPLL_Enum(Φ ∧ Δ);
Δ ← Δ ∪ {¬p}; S ← S ∪ DPLL_Enum(Φ ∧ Δ);
return S;
```

Table 4 Datasets Characteristics.

Instance	#Trans	#Items	AvgTransLen	Density(%)
Chess	3196	75	37	49.33
Mushroom	8124	119	23	19.33
Retail	88162	16470	10.3	0.06
Accidents	340183	468	33.8	7.22
Kosarak	990002	41270	8.1	0.02
Chainstore	1112949	46086	7.23	0.02
T60D10kI1k	10000	1000	30.4	3.04