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

© Amel Hidouri, Said Jabbour, and Badran Raddaoui; licensed under Creative Commons License CC-BY 4.0

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 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 \textbf{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. \textbf{FCHUIM} is based on a closed high utility representation and it employs a nested list to eliminate non-frequent itemsets. Furthermore, \textbf{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. \textbf{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 \textbf{2P-UF}, \textbf{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 \textbf{FUFM}, called \textbf{P-FUFM} [26], is a two-phases based algorithm that aims to reduce the running time of the \textbf{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 \( \Omega \) denote a universe of items (or symbols), called alphabet. The elements of \( \Omega \) are denoted by the letters \( a, b, c, \) etc. A subset of \( \Omega \) \( (I \subseteq \Omega) \) is called an itemset. The set of all itemsets over \( \Omega \) are denoted as \( 2^\Omega \), and the capital letters \( I, J, K, \) etc. are used to represent the elements of \( 2^\Omega \). 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: \( \text{Cover}(I, D) = \{i \in \mathbb{N} \mid (i, J) \in D \text{ and } I \subseteq J\} \). The support of \( I \) in \( D \) is then defined as the cardinality of \( \text{Cover}(I, D) \), i.e., \( \text{Supp}(I, D) = |\text{Cover}(I, D)| \). A high utility itemset \( I \subseteq \Omega \) s.t. \( \text{Supp}(I, D) \geq 1 \) is closed if and only if for any itemset \( J \) with \( I \subseteq J \), \( \text{Supp}(J, D) < \text{Supp}(I, D) \).

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 \( w_{\text{ext}}(a) \) for the external utility of \( a \). In addition, each item \( a \in \) a transaction \( T_i \) is associated with a positive value \( w_{\text{int}}(a, T_i) \), 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: \( u(a, T_i) = w_{\text{int}}(a, T_i) \times w_{\text{ext}}(a) \). 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 \( \theta \), the classical high utility itemset mining problem aims at finding the set of all itemsets in \( D \) whose utility value is no less than \( \theta \). 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 \( \theta \) 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 \( \theta \) and \( \delta \) respectively, an itemset \( I \) is a FHUI in \( D \) iff. \( \text{Supp}(I, D) \geq \delta \) 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 \( \delta \) 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 \( D \) is \{a\}, \{a, b\}, and \{a, b, g\}.

**Table 1 Sample Transaction Database.**

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Cover</th>
<th>Items</th>
<th>Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(a, 8)</td>
<td></td>
<td>(b, 2)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(b, 6)</td>
<td></td>
<td>(c, 3)</td>
<td>(c, 2)</td>
</tr>
<tr>
<td>3</td>
<td>(c, 4)</td>
<td></td>
<td>(d, 3)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(a, 6)</td>
<td></td>
<td>(b, 7)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(a, 8)</td>
<td>(f, 2)</td>
<td>(g, 1)</td>
<td></td>
</tr>
</tbody>
</table>

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 \( \theta' \) a local minimum utility threshold. Then, the utility-based cover of an itemset \( I \) in \( D \) is defined as: \( \text{Cover}_u(I, D) = \{(i, T_i) \in D, I \subseteq T_i, u(I, T_i) \geq \theta'\} \). Then, the utility-based support of \( I \) is the cardinality of its utility-based cover, i.e., \( \text{Supp}_u(I, D) = |\text{Cover}_u(I, D)| \).
Property 3. Let $D$ be a transaction database, $I$ an itemset, and $\delta$ a minimum support threshold. Then, if $\text{Supp}_u(I, D) \geq \delta$, then $\text{Supp}(I, D) \geq \delta$.

Given a support and local minimum utility thresholds $\delta$ and $\theta'$ 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 $\mathcal{L}$ be a propositional language built up inductively from a countable set $\mathcal{PS}$ of propositional variables, the boolean constants $\top$ (true or 1) and $\bot$ (false or 0) and the classical logical connectives $\{\neg, \wedge, \vee, \to, \leftrightarrow\}$ in the usual way. We use the letters $x, y, z, \ldots$ to range over the elements of $\mathcal{PS}$. Propositional formulas of $\mathcal{L}$ are denoted by $\Phi, \Psi, \ldots$. A literal is a propositional variable ($x$) of $\mathcal{PS}$ or its negation ($\neg x$). A clause is a (finite) disjunction of literals. For any formula $\Phi$ from $\mathcal{L}$, $\mathcal{P}(\Phi)$ denotes the symbols of $\mathcal{PS}$ occurring in $\Phi$. A formula in conjunctive normal form (CNF, for short) is a finite conjunction of clauses. A Boolean interpretation $\Delta$ of a CNF formula $\Phi$ is defined as a function from $\mathcal{P}(\Phi)$ to $\{0, 1\}$. A model of a formula $\Phi$ is an interpretation $\Delta$ that satisfies $\Phi$, i.e., if there exists an interpretation $\Delta : \mathcal{P}(\Phi) \to \{0, 1\}$ that satisfies all clauses in $\Phi$. The formula $\Phi$ is satisfiable if it has at least one model. In the sequel, we write $|=\ |$ for the logical consequence relation and $|=\text{UP}$ 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 $\Phi$, SAT determines whether exists a model for each clauses in $\Phi$. 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

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

---

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

---

CP 2022
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 ($a \in \Omega \setminus T_i$); 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 $\theta$. 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 $\delta$ 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 $\Phi_{\text{fhuim}} = (1) \land (2) \land (4)$. 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 $\Phi_{\text{cfhuim}} = \Phi_{\text{fhuim}} \land (3)$.

\[
\bigwedge_{i=1}^{m} (\neg q_i \leftrightarrow \bigvee_{a \in \Omega \setminus T_i} p_a) \quad (1) \quad \sum_{i=1}^{m} q_i \geq \delta \quad (2) \quad \bigwedge_{a \in \Omega} (p_a \lor \bigvee_{a \in T_i} q_i) \quad (3) \\
\sum_{i=1}^{m} \sum_{a \in T_i} u(a, T_i) \times (p_a \land q_i) \geq \theta \quad (4)
\]

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

**Proposition 5.** Let $D$ be a transaction database, $\theta$ a minimum high utility threshold, and $\delta$ a minimum support threshold. Let $\Phi_{\text{cfhuim}} = \Phi_{\text{fhuim}} \land (3)$ be a propositional formula. Then, there exists a one-to-one mapping between the models of $\Phi_{\text{cfhuim}}$ 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: $\bigwedge_{i=1}^{m} r_i \leftrightarrow \sum_{a \in T_i} u(a, T_i)(q_i \land p_a) \geq \theta'$. Now, to
Then, there exists a one-to-one mapping between the models of which the nonlinear constraints are linearized like \([6]\) which used cutting resolution. The we add the cardinality constraint (Constraint (6)). If that is the case, we call such candidate itemset as a FLHUI.

\[
\sum_{i=1}^{m} r_i \geq \delta \quad (6)
\]

\[\Phi_{\text{cflhuim}}(a,T) = (1) \wedge (5) \wedge (6) \wedge (3) \text{ be a propositional formula. Then, there exists a one-to-one mapping between the models of } \Phi_{\text{cflhuim}} \text{ and the set of closed FLHUIs in } D.\]

\[\text{Proposition 6. Let } D \text{ be a transaction database, } \theta' \text{ a local minimum utility threshold, and } \delta \text{ a minimum support threshold. Let } \Phi_{\text{cflhuim}} = (1) \wedge (5) \wedge (6) \wedge (3) \text{ be a propositional formula.}\]

\[\text{Then, there exists a one-to-one mapping between the models of } \Phi_{\text{cflhuim}} \text{ and the set of closed FLHUIs in } D.\]

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

\[
\neg q_1 \leftrightarrow (p_2 \vee p_4 \vee p_6 \vee p_f) \quad p_6 \vee q_2 \vee q_3
\]
\[
\neg q_2 \leftrightarrow (p_2 \vee p_4 \vee p_f \vee p_8) \quad p_6 \vee q_3 \vee q_4
\]
\[
\neg q_3 \leftrightarrow (p_2 \vee p_4 \vee p_f \vee p_9) \quad p_4 \vee q_4 \vee q_5
\]
\[
\neg q_4 \leftrightarrow (p_2 \vee p_f \vee p_f \vee p_9) \quad p_4 \vee q_2 \vee q_5
\]
\[
\neg q_5 \leftrightarrow (p_2 \vee p_4 \vee p_6) \quad p_3 \vee q_2 \vee q_5
\]
\[
r_1 \leftrightarrow (p_1 \wedge (5p_1 + 2p_2 + p_8 \geq 10)) \quad p_f \vee q_1 \vee q_3 \vee q_4
\]
\[
r_2 \leftrightarrow (q_2 \wedge (6p_2 + 3p_5 + 2p_9 \geq 10)) \quad p_f \vee q_1 \vee q_3 \vee q_4
\]
\[
r_3 \leftrightarrow (q_3 \wedge (4p_2 + 3p_6 \geq 10)) \quad p_f \vee q_2 \vee q_3 \vee q_4
\]
\[
r_4 \leftrightarrow (q_4 \wedge (6p_2 + 4p_6 + p_9 \geq 10)) \quad p_f \vee q_2 \vee q_3 \vee q_4
\]
\[
r_5 \leftrightarrow (q_5 \wedge (8p_2 + 7p_5 + 2p_6 + p_9 \geq 10)) \quad p_f \vee q_2 \vee q_3 \vee q_4
\]
\[
r_1 + r_2 + r_3 + r_4 + r_5 \geq 2
\]

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 \(\sum_{i=1}^{n} a_i x_i \text{ op } b\), where \(a_i\) and \(b\) are real coefficients, \(x_i\) (\(1 \leq i \leq n\)) are propositional variables, and the operator \text{ 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 \( \theta' \). 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 \( \theta' \).

### 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 \( y \rightarrow \sum_{i=1}^{n} w_i x_i \geq k \) [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 $\Phi$ and a variable $x_1 \in PS(\Phi)$, the models of $\Phi$ are those of $\Phi \land x_1$ and $\Phi \land \neg x_1$. By generalizing such principle for a subset of variables $\{x_1, \ldots, x_n\}$, the models of $\Phi$ are those of $\Psi_1, \ldots, \Psi_n$ where $\Psi_i = \Phi \land x_i \land \bigwedge_{1 \leq j < i} \neg x_j$. In our case, we have $\{x_1, \ldots, x_n\} = \{p_{a_1}, \ldots, p_{a_n}\}$ and $\Phi$ corresponds to $\Phi_{fh\text{ain}}$ (or $\Phi_{fh\text{ains}}$ for closed patterns) or $\Phi_{fl\text{ain}}$ (or $\Phi_{fl\text{ains}}$ for closed patterns). Hence, solving $\Psi_i = \Phi \land p_{a_i} \land \bigwedge_{1 \leq j < i} \neg p_{a_j}$ 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 $\Phi_i$ 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 $\Phi \land \Psi_n$ 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](image-url)  
*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 Frequent (Local) High Utility Itemset Mining Approach.

\begin{algorithm}
\begin{algorithmic}[1]
\State $S \leftarrow \emptyset$;
\For {$i \in [1..n]$}
\If {$\text{Supp}_u(a_i,D) \geq \delta$}
\State $D_i \leftarrow \{ (k,T_k) \in D \mid a_i \in T_k \}$, \quad $\Omega = \langle a_1,\ldots,a_n \rangle \leftarrow \text{items}(D_i)$, \quad $\Gamma \leftarrow \emptyset$;
\For {$T_j \in D_i$}
\If {$\sum_{c \in T_j} u(c,T_j) < \theta$}
\State $\Gamma \leftarrow \Gamma \land \neg r_i$, \quad $D_i \leftarrow D_i \setminus \{ T_j \}$;
\EndIf
\EndFor
\For {$b \in \text{items}(D_i)$}
\If {$\text{Supp}_u(b,D_i) < \delta$}
\State $\Gamma \leftarrow \Gamma \land \neg p_b$;
\EndIf
\EndFor
\State $\Psi \leftarrow p_{a_i} \land \bigwedge_{1 \leq j < i} \neg p_{a_j}$;
\State $\Phi \leftarrow \Phi^{\text{flhuim}}(D_i,\theta,\delta) \land \Psi \land \Gamma$; \quad /* $\Phi^{\text{flhuim}}$ for FLHUIM */
\State $S \leftarrow S \cup \text{DPLL Enum}(\Phi)$
\EndIf
\EndFor
\EndFor
\State $\text{return } S$;
\end{algorithmic}
\end{algorithm}

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 \cite{28} and FUFM \cite{27}. 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 \cite{12}. All characteristics of both real and synthetic datasets are summarised in Table 4: the number of transactions (\#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 ($\delta$) and the minimum high utility ($\theta$) 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 \cite{28}. 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 $\theta$ and $\delta$ values. Notice that the symbol (\textendash) means that the algorithm is not able to complete the mining process under the fixed time out (i.e., TO). The size of FHUI 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 $\delta$ and $\theta$ 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 able to mine the target itemsets under the timeout except for $\delta = 50\%$ and $\theta = 400k$ 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 $\theta$ and $\delta$ 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 $\delta$ as well as the minimum utility $\theta$ thresholds have a strong impact on the performance of the mining process. Specifically, for low values of $\theta$ and $\delta$, SATFHUIM needs more time to discover all itemsets. The density value also has an impact on the execution.

\footnote{We have used the C++ implementation for HU-FIMi, and the Python implementation for FUFM.}

\footnote{We did not provide a comparison with FCHUIM because the source code is not public.}
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 becomes 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](image-url) Running time of SATFHUIM w.r.t. minimum support threshold on real-world datasets.
According to Figure 5, it is clear that the number of FHUIs always depends on the chosen values of $\theta$ and $\delta$. 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 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 $T_{60D10kI1k}$. Note that $T_{60D10kI1k}$ 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 $\theta'$ and $\delta$ 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 $\delta$ and $\theta'$ 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 $\delta$ and $\theta'$ 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>
<thead>
<tr>
<th>Dataset</th>
<th>$\delta$(%)</th>
<th>$\theta'$</th>
<th>Time(s)</th>
<th>Memory (MB)</th>
<th>$#\text{Var}$</th>
<th>$#\text{Flu}$</th>
<th>Time(s)</th>
<th>Memory (MB)</th>
<th>$#\text{Flu}$</th>
<th>$#\text{CFlu}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>70</td>
<td>1.00</td>
<td>100</td>
<td>2.51</td>
<td>68</td>
<td>1759415</td>
<td>47874</td>
<td>25694</td>
<td>17201</td>
<td>1404</td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>1.00</td>
<td>100</td>
<td>2.51</td>
<td>68</td>
<td>1759415</td>
<td>47874</td>
<td>25694</td>
<td>17201</td>
<td>1404</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>1.00</td>
<td>100</td>
<td>2.51</td>
<td>68</td>
<td>1759415</td>
<td>47874</td>
<td>25694</td>
<td>17201</td>
<td>1404</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>1.00</td>
<td>100</td>
<td>2.51</td>
<td>68</td>
<td>1759415</td>
<td>47874</td>
<td>25694</td>
<td>17201</td>
<td>1404</td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>1.00</td>
<td>100</td>
<td>2.51</td>
<td>68</td>
<td>1759415</td>
<td>47874</td>
<td>25694</td>
<td>17201</td>
<td>1404</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>1.00</td>
<td>100</td>
<td>2.51</td>
<td>68</td>
<td>1759415</td>
<td>47874</td>
<td>25694</td>
<td>17201</td>
<td>1404</td>
</tr>
<tr>
<td>Retail</td>
<td>6</td>
<td>0.29</td>
<td>100</td>
<td>98</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
<td>184</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.29</td>
<td>100</td>
<td>98</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
<td>184</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.29</td>
<td>100</td>
<td>98</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
<td>184</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.29</td>
<td>100</td>
<td>98</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
<td>184</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.29</td>
<td>100</td>
<td>98</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
<td>184</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.29</td>
<td>100</td>
<td>98</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
<td>184</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td>Mushroom</td>
<td>40</td>
<td>0.64</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.64</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td>Accidents</td>
<td>40</td>
<td>0.64</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.64</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.64</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.61</td>
<td>95</td>
<td>33844708</td>
<td>36447084</td>
<td>55728869</td>
<td>69675386</td>
<td>16470</td>
<td>753221</td>
<td>331677</td>
</tr>
</tbody>
</table>

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 $\theta'$ and $\delta$ values. Instead, each bar corresponds to the average number of FLHUIs for each $\delta$ threshold in terms of the average of all fixed $\theta'$ 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 $\theta'$ and $\delta$ 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 SATFLHUIM algorithm w.r.t. $\theta'$ threshold values. For instance, on Chess, the average number of FLHUIs is about 21744362 for all fixed $\theta'$ values and $\delta = 30\%$, whereas the number of FHUls is 24081372 for the same $\delta$ value and $\theta' = 200\%$.
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


A. Hidouri, S. Jabbour, and B. Raddaoui


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
1 \[ \Delta = \emptyset; S = \emptyset; \]
2 if \( (\Phi \models p) \) then
3    return DPLL_Enum(\( \Phi \land p \)); /* unit clause */
4 end
5 if \( (\Phi \models \bot) \) then
6    return \emptyset; /* conflict */
7 end
8 if check_Pseudo_Boolean_constraint() == False then
9    return \emptyset;
10 end
11 if \( (\Delta \models \Phi) \) then
12    S ← S ∪ \{Δ\}; /* new found model */
13    return S;
14 end
15 p = select_variable(Var(\( \Phi \)));
16 \[ \Delta \leftarrow \Delta \cup \{p\}; S \leftarrow S \cup \text{DPLL}_\text{Enum}(\Phi \land \Delta) \];
17 \[ \Delta \leftarrow \Delta \cup \{-p\}; S \leftarrow S \cup \text{DPLL}_\text{Enum}(\Phi \land \Delta) \];
18 return S;
```

Table 4 Datasets Characteristics.

<table>
<thead>
<tr>
<th>Instance</th>
<th>#Trans</th>
<th>#Items</th>
<th>AvgTransLen</th>
<th>Density(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>3196</td>
<td>75</td>
<td>37</td>
<td>49.33</td>
</tr>
<tr>
<td>Mushroom</td>
<td>8124</td>
<td>119</td>
<td>23</td>
<td>19.33</td>
</tr>
<tr>
<td>Retail</td>
<td>88172</td>
<td>16470</td>
<td>10.3</td>
<td>0.08</td>
</tr>
<tr>
<td>Accidents</td>
<td>340183</td>
<td>468</td>
<td>33.8</td>
<td>7.22</td>
</tr>
<tr>
<td>Kosarak</td>
<td>999980</td>
<td>41270</td>
<td>8.1</td>
<td>0.02</td>
</tr>
<tr>
<td>Chainstore</td>
<td>1112099</td>
<td>46686</td>
<td>7.23</td>
<td>0.02</td>
</tr>
<tr>
<td>T60D10kI1k</td>
<td>10000</td>
<td>10000</td>
<td>30.4</td>
<td>3.34</td>
</tr>
</tbody>
</table>

Table 4 Datasets Characteristics.