A Parallel Declarative Framework for Mining High Utility Itemsets

Amel Hidouri[1,2](✉), Said Jabbour[2], Badran Raddaoui[4], Mouna Chebbah[3], and Boutheina Ben Yaghlane[1]

[1] LARODEC, University of Tunis, Tunis, Tunisia
boutheina.yaghlane@ihec.rnu.tn
[2] CRIL - CNRS UMR 8188, University of Artois, Lens, France
{hidouri,jabbour}@cril.fr
[3] LARODEC, Univ. Manouba, ESEN, Manouba, Tunisia
mouna.chebbah@esen.tn
[4] SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Paris, France
badran.raddaoui@telecom-sudparis.eu

Abstract. One of the most active research topics in data mining is pattern discovery involving the well-known task of enumerating interesting patterns from databases. The problem of mining high utility itemsets is to find the set of items with the highest utility values based on a given minimum utility threshold. However, due to the advancement of big data technologies, finding all itemsets is much more harder due to the huge number of patterns and the large required resources. Parallel processing is an effective way to efficiently address the problem of mining patterns from large databases. Based on classical propositional logic, we propose in this paper a parallel method to handle efficiently the problem of discovering high utility itemsets from transaction databases. To do this, a decomposition technique is used to splitting the original problem of mining high utility itemsets into smaller and independent sub-problems that can be handled easily in a parallel manner. Then, empirical evaluations on different real-world datasets show that the proposed method is very efficient while being flexible enough to handle additional user constraints when discovering closed high utility itemsets.

Keywords: Data mining · High utility · Symbolic Artificial Intelligence · Propositional satisfiabilty · Parallel solving

1 Introduction

Pattern extraction is a well-known task in data mining that aims to infer knowledge based on different types of interesting measures. Discovering High Utility Itemsets (HUIM, for short) is one of the fundamental tasks in pattern discovery that generalizes the classical problem of frequent itemsets mining (FIM, for short). In fact, traditional FIM techniques are still insufficient for discovering the most valuable itemsets, e.g., when an itemset with a high profit is regarded as infrequent. Unfortunately, the FIM task is not appropriate to deal with this research question because the importance of an itemset is binary and it is only determined by the occurrence of such itemset in the database. To address the previous limitation, the HUIM task was introduced as new paradigm to discover the set of items that appear together in a transaction database and have a high importance to the end-user, i.e., which is expressed by a utility function. So, an itemset is coined a *high utility itemset* (HUI, for short) if its utility value is greater than a user specified threshold.

In the literature, different proposals have been developed to handle HUIM. These approaches mainly differ in terms of data structures and search strategies used to exploring the search space and the database scanning optimization. Nonetheless, the main limitation of this line of research concerns its sequential computational power constraints, which make it more expensive and causes scalability issues, especially when dealing with highly dense and huge databases. Recently, parallel computing with multi-core machine has received a great attention to overcome this limitation and to improve the performance of sequential algorithms. Particularly, the popularity of multi-core architectures allows various data mining problems to be handled in parallel. Parallel computing has been recently used to improve the performance of HUIM algorithms by accelerating the mining of high utility itemsets in transaction databases. Among these proposals, one can cite **pEFIM** [14], an extension of the EFIM algorithm, and **MCHMiner** [17], an extension of iMEFIM (especially exploited for dynamic transaction databases), that used simple static load balancing between different processors. Moreover, the **CLB** and **PLB** algorithms [1], parallel-based extensions of ULB-Miner [4], combine the tree and utility-list structures while applying a parallel processing when mining candidate itemsets. Additionally, other approaches are based on distributed systems to find HUIs from transaction databases. Among these algorithms, we note **P-FHM+** [15] and **PHUI-Growth** [13].

Symbolic Artificial Intelligence has recently been used to solve different data mining problems [3]. It is based on a declarative language to express constraints over patterns. Declarative methods have been then used to model data mining tasks as a constraints network or a propositional formula (Propositional Satisfiability SAT, for short) where the found models correspond exactly to the required patterns. More interestingly, such methods enable users to constraint the desired motifs by adding new constraints without modeling the underlying problem from scratch. Unfortunately, the main challenging point for such approaches is the scalability issue when dealing with large datasets.

Parallel SAT solving has received a lot of attention in the SAT community. Indeed, two kinds of approaches have been proposed. The first category divides the search space using a divide-and-conquer principle that primarily divides the search space using the well-known guiding-path concept [18]. The second one makes use of a Portfolio [7] of DPLL engines, allowing them to compete and collaborate to be the first to solve a given instance. In this paradigm, each solver works on the original formula, and the search space is not divided. In general, the portfolio employs a variety of search engines to increase the likelihood that one of them will solve the problem, as well as various strategies for search space exploration with clause sharing to avoid redundancy. The divide-and-conquer paradigm is clearly the most convenient for our purpose, as our goal is to split the transaction database in order to generate several formulas of reasonable size.

In this paper, we present `PSAT-HUIM`, a new parallel algorithm for efficiently enumerating high utility itemsets embedded in transaction databases. Based on the divide and conquer paradigm, we present a parallel declarative method, that extends the work of [9], to discover the set of high utility itemsets. We also show through extensive experiments on different real-world datasets the efficiency of our `PSAT-HUIM` approach.

2 Background

In this section, we present the relevant preliminaries and definitions related to the HUIM and the propositional satisfiability problems, respectively.

2.1 High Utility Itemset Mining Problem

Let Ω represents a universe of unique items (or symbols) that appear in a database. A transaction database $\mathscr{D} = \{T_1, T_2, \ldots, T_m\}$ is a set of m transactions such that each transaction T_i is a set of items (i.e., $T_i \subseteq \Omega$), and T_i has a unique identifier i called its transaction identifier (TID, for short). In what follows, we give some additional definitions related to the high utility mining problem.

Definition 1 (Internal Utility). *For each transaction T_i such that $a \in T_i$, a positive number $w_{int}(a, T_i)$ is called the internal utility of the item a (e.g., purchase quantity).*

Definition 2 (External Utility). *Each item $a \in \Omega$ is associated with a positive number $w_{ext}(a)$, called its external utility (e.g., unit profit).*

Definition 3 (Utility of an item/itemset in a transaction). *Given a transaction database $\mathscr{D}$, the utility of an item a in a transaction $(i, T_i) \in \mathscr{D}$, denoted by $u(a, T_i)$, is $u(a, T_i) = w_{int}(a, T_i) \times w_{ext}(a)$. Then, the itemset's utility X in T_i, written as $u(X, T_i)$ is defined as follows:*

$$u(X, T_i) = \sum_{a \in X} u(a, T_i) \tag{1}$$

Definition 4 (Utility of an itemset in a database). *Let $\mathscr{D}$ be a transaction database. The utility of an itemset X in $\mathscr{D}$, denoted by $u(X, \mathscr{D})$, is defined as:*

$$u(X, \mathscr{D}) = \sum_{(i, T_i) \in \mathscr{D} \ | \ X \subseteq T_i} u(X, T_i) \tag{2}$$

Example 1. We assume that Consider the transaction database in Table 1. We assume that this database represents a set of products in a retail store. In addition, the external utility of items, i.e., the price of each item, is given in Table 2. Clearly, the utility of the itemset $\{b, d\}$ is computed as follows: $u(\{b, d\}) = u(\{b, d\}, T_2) + u(\{b, d\}, T_4) = 24$.

Table 1. A sample transaction database

TID	Items				
T_1	$(a, 3)$	$(c, 1)$		$(e, 3)$	
T_2	$(a, 2)$	$(b, 1)$	$(c, 3)$	$(d, 4)$	$(e, 3)$
T_3	$(a, 5)$		$(c, 1)$		$(d, 1)$
T_4	$(b, 4)$		$(d, 3)$	$(e, 1)$	
T_5	$(a, 1)$	$(d, 2)$		$(e, 2)$	

Table 2. The external utilities of the itemsets

Item	Unit profit
a	4
b	2
c	1
d	2
e	3

Definition 5 (High Utility Itemset). *An itemset X is called a high utility itemset (HUI) in a database $\mathscr{D}$ if its utility value is greater than a minimum utility threshold θ, i.e., $u(X) \geq \theta$.*

Definition 6 (Closed high utility itemset). *Let $\mathscr{D}$ be a transaction database. X is called a closed high utility itemset if there exists no high utility itemset X' such that $X \subset X'$, and $\forall(i, T_i) \in \mathscr{D}$, if $X \in T_i$ then $X' \in T_i$.*

Problem Statement. Given a transaction database $\mathscr{D}$ and a user-specified minimum utility threshold θ, the goal of computing (closed) high utility itemsets problem consists of finding the set of all (closed) high utility itemsets in $\mathscr{D}$ with a utility no less than θ, i.e.,
$HUI = \{X : u(X, \mathscr{D}) \mid X \subseteq \Omega, u(X, \mathscr{D}) \geq \theta\}$

Example 2. Let us consider again the transaction database given in Example 1. Given a minimum utility threshold $\theta = 45$, then the set of high utility itemsets with their utility in the database of Table 1 is $\{\{a, c\} : 45, \{a, d\} : 46, \{a, e\} : 59\}$.

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. More formally,

Definition 7 (Transaction Utility). *The transaction utility of a transaction* T_i *in a database* $\mathscr{D}$*, denoted by* $TU(T_i)$*, is the sum of the utility of all items in* T_i*, i.e.,*

$$TU(T_i) = \sum_{a \in T_i} u(a, T_i) \tag{3}$$

Definition 8 (Transaction Weighted Utilization). *The transaction weighted utilization of an itemset* X *in a transaction database* $\mathscr{D}$*, denoted by* $TWU(X, \mathscr{D})$*, is defined as:*

$$TWU(X, \mathscr{D}) = \sum_{(i,T_i) \in \mathscr{D} \ \mid \ X \subseteq T_i} TU(T_i) \tag{4}$$

2.2 Propositional Logic and SAT Problem

A propositional language $\mathscr{L}$ consists of three main components: a countable set of propositional variables $\mathscr{P}$ for which we use the letters p, q, r, etc. to range over $\mathscr{P}$, a set of logical connectives such as $\neg, \wedge, \vee, \rightarrow$ and the two logical constants $\top$ (*true* or 1) and $\bot$ (*false* or 0). A literal is a propositional variable p or its negation $\neg p$. Propositional formulas of $\mathscr{L}$ will be denoted as Φ, Ψ, etc. We also use $\mathscr{P}(\Phi)$ to denote the set of propositional variables occurring in the propositional formula Φ. A Boolean interpretation Δ of a formula Φ is defined as a function from $\mathscr{P}(\Phi)$ to $(0, 1)$, i.e., $\Delta : \mathscr{P}(\Phi) \rightarrow (0, 1)$. Now, a model of a formula Φ is a Boolean interpretation Δ that satisfies Φ. We use $Mod(\Phi)$ to denote the set of models of Φ. A formula Φ is satisfiable if there exists a model of Φ. Then, Φ is valid or a theorem, if every Boolean interpretation is a model of Φ. As usual, $\models$ refers to the logical inference and $\models_{up}$ the one restricted to unit propagation.

Now, a clause is a disjunction of literals. A formula Φ is in conjunctive normal form (CNF, for short) if Φ can be written as a conjunction of a finite set of clauses. We stress here that any propositional formula can be rewritten as a CNF one by applying the linear Tseitin's encoding [16]. The decision problem verifying the satisfiability of a propositional formula in CNF is called SAT. Given a CNF formula Φ, SAT aims to identify if Φ possesses a model or not. Interestingly, state-of-the-art SAT solvers have been shown to be very useful for handling various real-world problems, including overlapping community detection in graphs [10, 11], data mining [12], etc.

3 SAT Encoding of (Closed) High Utility Itemset Mining

In this section, we review the formulation of closed HUIM problem into propositional satisfiability [8]. The proposed encoding consists in a set of propositional variables to represent both items and transactions of the considered transaction database $\mathscr{D}$. More precisely, each item a (resp. each transaction identifier i), is associated with a propositional variable, denoted by p_a (resp. q_i). Given a Boolean interpretation $\varDelta$, the itemset and its cover i.e., the set of transactions in which it appears are simply expressed as the sets $\{a \in \varOmega \mid \varDelta(p_a) = 1\}$ and $\{i \in \mathbb{N} \mid \varDelta(q_i) = 1\}$, respectively. Now, the translation of the HUIM task into propositional satisfiability is obtained by introducing a set of logical constraints as depicted by Fig. 1. The first propositional formula (5) encodes the cover of the candidate itemset. This formula expresses that the itemset appears in the i^{th} transaction, i.e., $q_i = true$. In other words, 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 ($a \in \varOmega \backslash T_i$); when q_i is *false*. This means that at least an item not appearing in the transaction i is set to *true*.

$$\bigwedge_{i=1}^{m}(\neg q_i \leftrightarrow \bigvee_{a\in\varOmega\backslash T_i} p_a) \quad (5) \qquad \bigwedge_{a\in\varOmega}(p_a \vee \bigvee_{a\notin T_i} q_i) \quad (6) \qquad \sum_{i=1}^{m}\sum_{a\in T_i} u(a,T_i)\times(p_a \wedge q_i) \geqslant \theta \quad (7)$$

$$\sum_{i=1}^{m}\sum_{a\in T_i} u(a,T_i)\times r_{ai} \geqslant \theta \quad (8) \qquad \bigwedge_{i=1}^{m}\bigwedge_{a\in T_i}(r_{ai} \leftrightarrow p_a \wedge q_i) \quad (9)$$

Fig. 1. SAT-based encoding scheme for HUIM.

The constraint over the utility of the itemset X in $\mathscr{D}$ is expressed using the linear inequality (7) requiring at least a threshold θ. Notice that Constraint (7) can be translated into clauses or managed in the solver as proposed in [8]. By the use of additional variables, Constraint (7) can be rewritten as a conjunction of the constraints (8) and (9). Lastly, the propositional formula (6) allows to select the set of closed HUIs in $\mathscr{D}$. It ensures that if the candidate itemset is involved in all transactions containing the item a, then a must belong to the itemset. As shown in [8], the CNF formula (5) $\wedge$ (8) $\wedge$ (9) encodes the HUIM problem, while the formula (5) $\wedge$ (8) $\wedge$ (9) $\wedge$ (6) encodes the closed HUIM problem.

Example 3. The formula encoding the problem of mining closed HUIs of Example 1 with $\theta = 20$ is as follows:

$\neg q_1 \leftrightarrow (p_b \vee p_d)$ $\neg q_3 \leftrightarrow (p_b \vee p_e)$ $\neg q_4 \leftrightarrow (p_a \vee p_c)$ $\neg q_5 \leftrightarrow (p_b \vee p_c)$

$r_{a1} \leftrightarrow p_a \wedge q_1$ $r_{c1} \leftrightarrow p_c \wedge q_1$ $r_{e1} \leftrightarrow p_e \wedge q_1$ $r_{a2} \leftrightarrow p_a \wedge q_2$ $r_{b2} \leftrightarrow p_b \wedge q_2$

$r_{c2} \leftrightarrow p_c \wedge q_2$ $r_{d2} \leftrightarrow p_d \wedge q_2$ $r_{e2} \leftrightarrow p_e \wedge q_2$ $r_{a3} \leftrightarrow p_a \wedge q_3$ $r_{c3} \leftrightarrow p_c \wedge q_3$

$r_{d3} \leftrightarrow p_d \wedge q_3$ $r_{b4} \leftrightarrow p_b \wedge q_4$ $r_{d4} \leftrightarrow p_d \wedge q_4$ $r_{e4} \leftrightarrow p_e \wedge q_4$ $r_{a5} \leftrightarrow p_a \wedge q_5$

$r_{d5} \leftrightarrow p_d \wedge q_5$ $r_{e5} \leftrightarrow p_e \wedge q_5$

$(p_a \vee q_4) \wedge (p_b \vee q_1 \vee q_3 \vee q_5) \wedge (p_c \vee q_4 \vee q_5) \wedge (p_d \vee q_1) \wedge (p_e \vee q_3)$

$12r_{a1} + r_{c1} + 9r_{e1} + 8r_{a2} + 2r_{b2} + 3r_{c2} + 8r_{d2} + 9r_{e2} + 20r_{a3} + r_{c3} + 2r_{d3} + 8r_{b4} + 6r_{d4} + 3r_{e4} +$

$4r_{a5} + 4r_{d5} + 6r_{e5} \geq 20$

4 Parallel High Utility Itemsets Mining Using Propositional Satisfiability

In this section, we present a parallel approach based on propositional logic to computing high utility itemsets from transaction databases. Our proposed algorithm improves the state-of-the-art SAT-based approach presented in [9] by utilizing a multi-core architecture. In fact, two types of solvers have been developed in the literature to solve SAT problems in parallel: divide-and-conquer and portfolios. The first category is of particular interest in this paper, since in pattern mining we deal with an enumeration task. Thus, the divide-and-conquer paradigm aims to split the search space into many sub-spaces, explored in parallel by SAT solvers. Generally, divide-and-conquer based approaches use the well-known guiding-path concept to divide the search space [2]. In fact, a guiding-path consists to restrict the search space to a given region by adding constraints to the original problem. However, the main challenge of parallel processing is the load balancing between processors, which is important to avoid idleness. Our proposed approach relies on an effective static load balancing that initially split the work among processors using a heuristic cost function, while each processor has a direct and equal access to memory, i.e., shared-memory (SMP) architecture. The main advantage of this system is that it is easy to implement. More precisely, in our case, we decompose the enumeration of the whole models of Φ by generating a number of sub-formulas $\{\Phi_1, \ldots, \Phi_n\}$ using the guiding-path concept. Such sub-formulas are then distributed between cores for resolution. The goal is then to avoid encoding the whole transaction database by generating numerous sub-problems with reasonable size that can be solved in a parallel.

Formally, let Φ be a formula and $\Psi_1, \ldots, \Psi_n$ a set of formulas over $\mathscr{P}(\Phi)$. Then, $\Psi_1, \ldots, \Psi_n$ is a guiding-path set if and only if:

$$\top \equiv \Psi_1 \vee \ldots \vee \Psi_n$$

$$\Psi_i \wedge \Psi_j \models \bot, \forall\ i \ \neq \ j$$

Clearly, the satisfiability of Φ is related to the satisfiability of at least $\Phi \wedge \Psi_i$, for $i \in [1..n]$. Moreover, the following result holds:

$$Mod(\Phi) = \bigcup_{i=1}^{n} Mod(\Phi \wedge \Psi_i)$$

In addition, for $i \neq j$, we have $Mod(\Phi \wedge \Psi_i) \cap Mod(\Phi \wedge \Psi_j) = \emptyset$ allowing to enumerate the models of $\Phi \wedge \Psi_i$ independently.

For the high utility itemsets mining task using propositional logic, we define the guiding path set as:

$$\Psi_i = p_{a_i} \wedge \bigwedge_{j<i} \neg p_{a_j}$$

It is easy to check that this set of formulas is a guiding path one. In fact, Ψ_i requires that the literal p_{a_i} is *true*, i.e., the itemset must contain a_i while the literals that represent the items a_j with $j < i$ are *false*. Interestingly, $\Phi(\mathscr{D}, \theta) \wedge \Psi_i$ is equivalent to $\Phi(\mathscr{D}_i, \theta) \wedge \Psi_i$ i.e., the encoding can be restricted to transactions involving a_i where the items a_j, $j < i$ are removed (as depicted in Fig. 2). As performed, the size of the generated sub-formulas $\Phi(\mathscr{D}_i, \theta) \wedge \Psi_i$ can be considerably reduced.

Example 4. Let us consider again the transaction database in Table 1. Figure 2 shows the sub-table obtained by considering the guiding-path set as explained above.

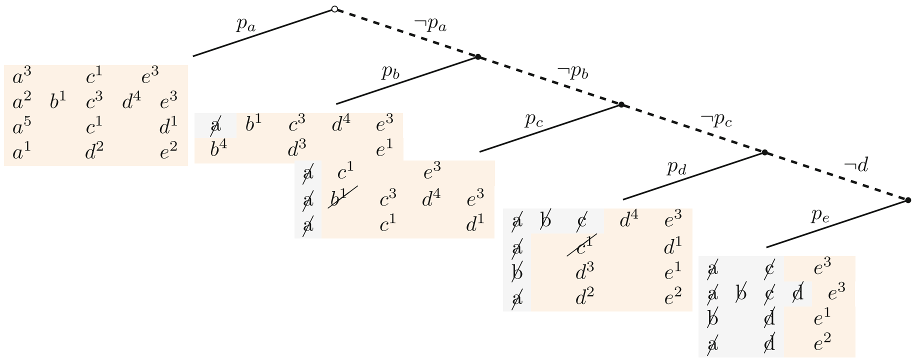

Fig. 2. Item based partitioning tree of the database in Table 1

Now, our parallel SAT-based approach to enumerating all (closed) HUIs from transaction databases is depicted in Algorithm 1. The algorithm takes as inputs a transaction database $\mathscr{D}$, a minimum utility threshold θ, and a number of cores k, and returns all (closed) HUIs embedded in $\mathscr{D}$. First, the SAT-based model

Algorithm 1: Parallel SAT for High Utility Itemset Mining (PSAT-HUIM)

```
Input: $\mathscr{D}$: a transaction database, $\theta$: a minimum utility threshold, $k$: a number
       of cores
Output: $S$: the set of all high-utility itemsets
1  $\Omega = \langle a_1, \ldots, a_n \rangle \leftarrow items(\mathscr{D})$;
2  $S \leftarrow \emptyset$ ;                                   /* set of models (HUIs) */
3  $\Gamma \leftarrow \emptyset$;
4  for $i$ in $[0..k-1]$ do
5      $initSolver(i)$;
6      $S_i \leftarrow \emptyset$;
7  end
8  for $i$ in $[1..n]$ do
9      if $TWU(a_i, \mathscr{D}) \geq \theta$ then
10         $\mathscr{D}_i \leftarrow \{(j, T_j) \in \mathscr{D} \mid a_i \in T_j\}$ ;
11         for $b \in items(\mathscr{D}_i)$ do
12             if $TWU(b, \mathscr{D}_i) < \theta$ then
13                 $\Gamma \leftarrow \Gamma \wedge \neg p_b$;
14             end
15         end
16         $\Psi_i \leftarrow p_{a_i} \wedge \bigwedge_{j<i} \neg p_{a_j}$;
17         $S_i \leftarrow dpll\_Enum(\Phi(\mathscr{D}_i, \theta) \wedge \Psi_i \wedge \Gamma, \theta)$;      /* Solved by calling
           Solver[$i\%k$] */
18         $S \leftarrow S \cup S_i$ ;
19     end
20 end
21 return $S$;
```

enumeration solvers are initialized. Each one is associated to a given thread or core i. Notice that our division strategy consists to assign the sub-problem i to the solver $i\%k$. During the decomposition, an item a_i is selected and the transactions containing a_i are then picked to construct the sub-database $\mathscr{D}_i$ for encoding. Clearly, all items b with $TWU(b, D_i) < \theta$ cannot be part of a (closed) HUI, and then are propagated to *false* (lines 11–14). The set of k solvers is then launched in parallel, the solver number $(i\%k)$ is run successively on the formula Φ_{D_i} and the set of models S_i are returned (line 17). Finally, the union of all models S_i for $i \in [1..k]$ yields the entire set of models. Such set corresponds to the entire set of found (closed) HUIs, that will be finally returned by our algorithm. Let us notice that the guiding path for the HUIM task is generated using an ordering over the variables encoding items. This order must be well chosen for an efficient SAT resolution. The strategy used in this paper consists to sort the items according to their frequency in the initial database. The motivation behind such heuristic is to allow generating small sub-problems by starting with less frequent items.

5 Experimental Results

We conducted an experimental evaluation of our parallel SAT-based formulation of HUIM task using numerous real-world transaction databases. Experiments are carried out on a personal computer equipped with an Intel Core i7 processor and 16 GB of RAM at 2.8 Ghz running macOS 10.13.4. Empirical evaluations were performed on seven real-life datasets commonly used in the HUIM literature. These datasets are available from the Open source Data Mining Library (SPMF) [6]. These benchmarks are *Chess*, *Foodmart*, *Mushroom*, *Retail*, *Accidents*, *Kosarak* and *Chainstore*. They have a variety of characteristics and contain data from real-world scenarios. Our algorithm is written in C++, and we used the MiniSAT solver [5], which is slightly modified for the model enumeration problem. Let us note here that the computation time for our approach includes both the encoding and solving time. In Table 3, we report the number of transactions (#Trans), the number of items (#Items), and the average transaction length (AvgTransLen).

Table 3. Datasets characteristics

Instance	#Trans	#Items	AvgTransLen
Chess	3196	75	37
Foodmart	4141	1559	4.42
Mushroom	8124	119	23
Retail	88162	16470	10.3
Accidents	340183	468	33.8
Kosarak	990002	41270	8.1
Chainstore	1112949	46086	7.23

We conduct two types of experiments in order to evaluate the efficiency of our proposed SAT method for computing the set of all HUIs from databases. In the first one, we assess our algorithm's performance on each dataset for various minimum utility thresholds while considering 1, 2 and 4 cores. In the second, we demonstrate the effectiveness of our load balancing strategy among the different cores. Figure 3 displays the running time of our SAT-based algorithm on each dataset.

5.1 Parallel Evaluation

According to Fig. 3, the performance of our SAT-based approach depends on the considered dataset and the minimum utility threshold values as well as the considered cores number. As expected, the parallel based approach allows to significantly reduce the running time of mining the set of HUIs. Clearly, this latter decreases as the number of cores increases. For almost tested datasets, the solving time is divided by two when the number of cores pass from 1 to 2. More specifically, for *Accidents* dataset and $\theta = 17.5 \times 10^6$, the running time with one core exceeds 450 s while this time is approximately equal to 300 seconds with two cores, and for `kosarak` we move from 90 seconds with one core to 50 seconds with two cores for $\theta = 1.2 \times 10^6$. We can also remark that the gain in terms of solving time is greater when the number of cores increases from 1 to 2 rather than 2 to 4. To summarize, the multi-core architecture outperforms sequential architecture on all tested datasets.

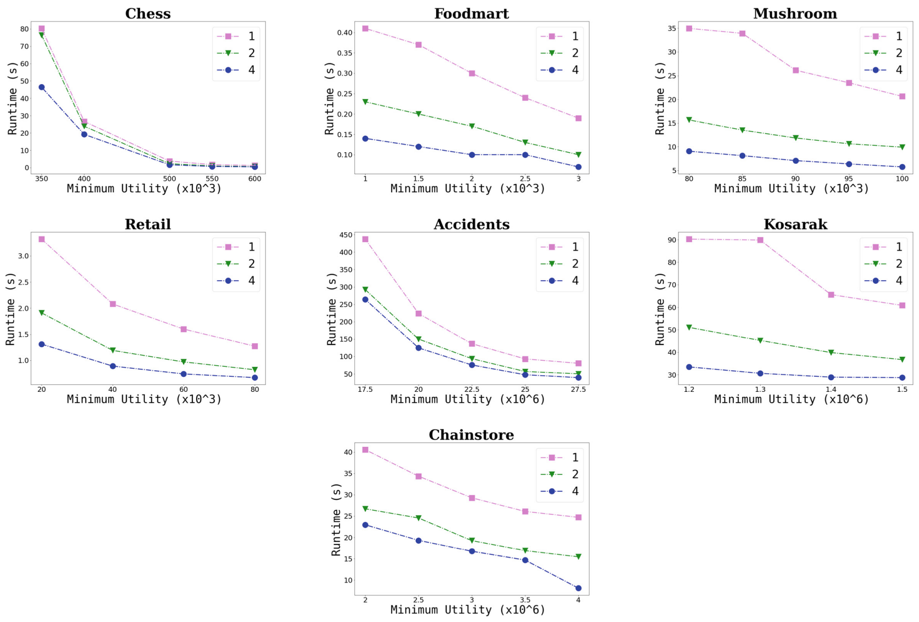

Fig. 3. Performance gain w.r.t. the number of cores

Let us recall here that we use a splitting method where the resulting sub-formulas are shared fairly between cores for a better load balancing. The next sub-section is dedicated to the load balancing analysis.

5.2 Load Balancing

This subsection provides an empirical analysis of the CPU time with different number of cores on all datasets to assess the suitability of our load balancing strategy. For this, we fix the number of cores to 4 for each value of minimum utility threshold. We report in Fig. 4 the average running time over cores as well as the minimum and maximum time to quantify the idleness of some cores. We mention here that the tighter the difference between minimum and maximum running time is, the better the load balancing is. As we can observe the relative

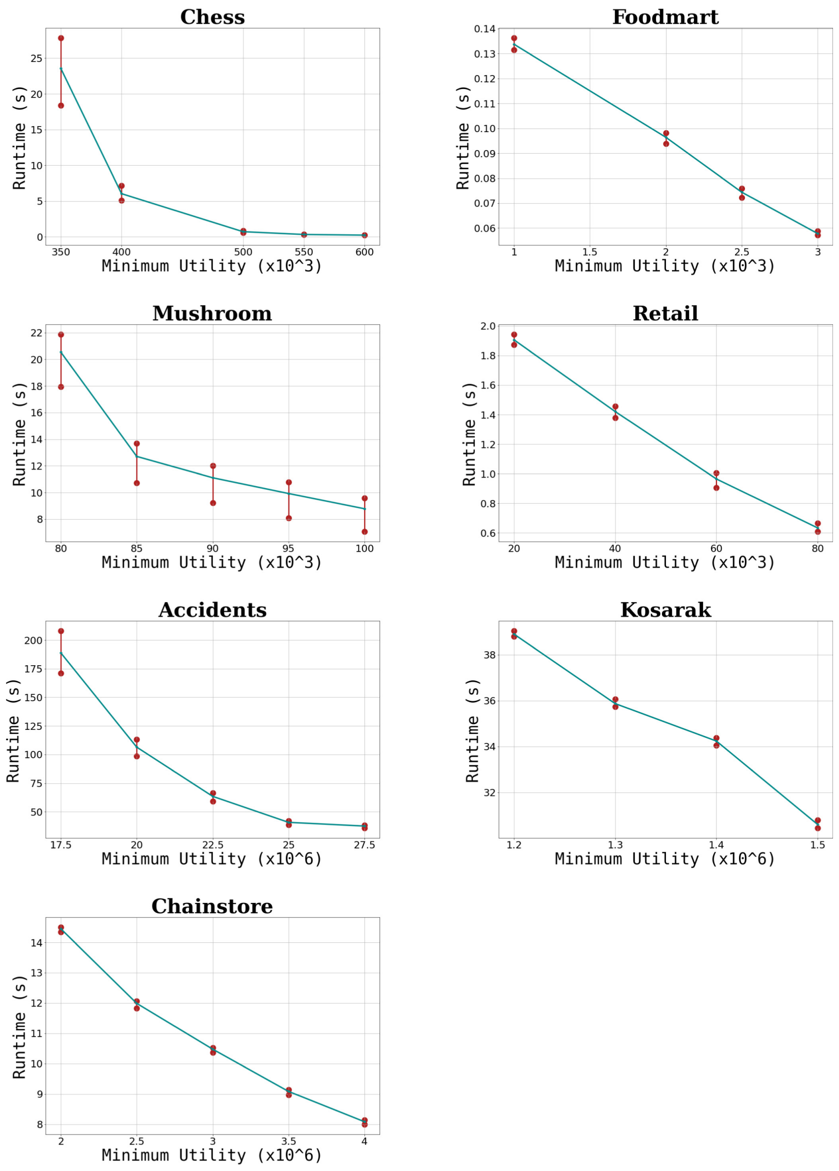

Fig. 4. Load unbalancing between cores

load unbalancing for the datasets *Chainstore*, *Retail* and *Kosarak* is very limited. Thus, all the cores spent almost the same time to solve their assigned sub-problems. Overall, our guiding paths generation principle allows to balance the number of found models between the different threads. These results indicate that our SAT method is both efficient and scalable.

6 Conclusion

In this paper, we considered a parallel approach to compute the set of all (closed) high utility itemsets using propositional logic. The proposed approach is based on divide and conquer paradigm for a multi-processor architecture. A decomposition approach is provided to avoid modeling the entire transaction database by considering numerous but smaller sub-problems that can be handled easily in parallel. The empirical evaluation tends to support our splitting strategy by providing interesting load balancing among cores.

As a future work, we want to investigate two research directions. In the first one, we plan to develop a parallel SAT approach with a dynamic splitting strategy in order to reduce the load balancing effect. In the second, we aim to extend our proposal for a distributed approach to avoid the disadvantage of shared memory in multi-core architecture when handling more large datasets.

Acknowledgements. This research has received support from the ANR CROQUIS (Collecting, Representing, cOmpleting, merging, and Querying heterogeneous and UncertaIn waStewater and stormwater network data) project, grant ANR-21-CE23-0004 of the French research funding agency Agence Nationale de la Recherche (ANR).

References

1. Atmaja, E.H.S., Sonawane, K.: Parallel algorithm to efficiently mine high utility itemset. In: ICT Analysis and Applications, pp. 167–178 (2022)
2. Böhm, M., Speckenmeyer, E.: A fast parallel sat-solver-efficient workload balancing. Ann. Math. Artif. Intell. **17**, 381–400 (1996). https://doi.org/10.1007/BF02127976
3. Coquery, E., Jabbour, S., Sais, L., Salhi, Y., et al.: A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In: ECAI, pp. 258–263 (2012)
4. Duong, Q.-H., Fournier-Viger, P., Ramampiaro, H., Nørvåg, K., Dam, T.-L.: Efficient high utility itemset mining using buffered utility-lists. Appl. Intell. **48**(7), 1859–1877 (2017). https://doi.org/10.1007/s10489-017-1057-2
5. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
6. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S., et al.: SPMF: a java open-source pattern mining library. J. Mach. Learn. Res. **15**, 3389–3393 (2014)
7. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a parallel sat solver. J. Satisfiability Boolean Model. Comput. **6**, 245–262 (2010)
8. Hidouri, A., Jabbour, S., Raddaoui, B., Yaghlane, B.B.: A sat-based approach for mining high utility itemsets from transaction databases. In: International Conference on Big Data Analytics and Knowledge Discovery, pp. 91–106 (2020)
9. Hidouri, A., Jabbour, S., Raddaoui, B., Yaghlane, B.B.: Mining closed high utility itemsets based on propositional satisfiability. Data Knowl. Eng. **136**, 101927 (2021)
10. Jabbour, S., Mhadhbi, N., Raddaoui, B., Sais, L.: Sat-based models for overlapping community detection in networks. Computing **102**(5), 1275–1299 (2020)

11. Jabbour, S., Mhadhbi, N., Raddaoui, B., Sais, L.: A declarative framework for maximal k-plex enumeration problems. In: AAMAS (2022, to appear)
12. Jabbour, S., Sais, L., Salhi, Y.: Mining top-k motifs with a sat-based framework. Artif. Intell. **244**, 30–47 (2017)
13. Lin, Y.C., Wu, C.W., Tseng, V.S.: Mining high utility itemsets in big data. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 649–661 (2015)
14. Nguyen, T.D., Nguyen, L.T., Vo, B.: A parallel algorithm for mining high utility itemsets. In: International Conference on Information Systems Architecture and Technology, pp. 286–295 (2018)
15. Sethi, K.K., Ramesh, D., Edla, D.R.: P-FHM+: parallel high utility itemset mining algorithm for big data processing. Procedia Comput. Sci. **132**, 918–927 (2018)
16. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning, pp. 466–483 (1983)
17. Vo, B., Nguyen, L.T., Nguyen, T.D., Fournier-Viger, P., Yun, U.: A multi-core approach to efficiently mining high-utility itemsets in dynamic profit databases. IEEE Access **8**, 85890–85899 (2020)
18. Zhang, H., Bonacina, M.P., Hsiang, J.: PSATO: a distributed propositional prover and its application to quasigroup problems. J. Symb. Comput. **21**, 543–560 (1996)