A Multi-Core Approach to Efficiently Mining High-Utility Itemsets in Dynamic Profit Databases

BAY VO, (Member, IEEE), LOAN T.T. NGUYEN, TRINH D.D. NGUYEN, PHILIPPE FOURNIER-VIGER, AND UNIL YUN

Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh City 700000, Vietnam
School of Computer Science and Engineering, International University, Ho Chi Minh City 700000, Vietnam
Vietnam National University, Ho Chi Minh City 700000, Vietnam
Institute of Research and Development, Duy Tan University, Da Nang 550000, Vietnam
School of Humanities and Social Sciences, Harbin Institute of Technology, Shenzhen 518055, China
Department of Computer Engineering, Sejong University, Seoul 05006, South Korea

Corresponding author: Loan T.T. Nguyen (nttloan@hcmiu.edu.vn)

This work was supported by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.05-2018.01.

ABSTRACT Analyzing customer transactions to discover high-utility itemsets is a popular task, which consists of finding the sets of items that are purchased together and yield a high profit. However, many studies assume that transactional data is static while in real-life, it changes over time. For example, the unit profits of items may vary from one week to another because sale prices and production costs may change. Many algorithms for mining high-utility itemsets (HUI) ignore this important property and thus are inapplicable or generate inaccurate results on real data. To address this issue, this paper proposes a novel algorithm named Multi-Core HUI Miner (MCH-Miner). It adapts techniques introduced in the iMEFIM algorithm to run on a parallel multi-core architecture to efficiently mine HUIs in dynamic transaction databases. An empirical evaluation shows that in most cases, MCH-Miner is significantly faster than iMEFIM, and that the cost of database scans is reduced.

INDEX TERMS Data mining, high utility itemset, dynamic profit, parallel, multithread.

I. INTRODUCTION

A key problem in the field of data mining is frequent itemset mining (FIM), which was introduced in 1994 by Agrawal and Skirant [1]. It consists of identifying frequently occurring sets of items in a transaction database, that is frequent itemsets (FI). FIM algorithms rely on the support framework to discover these itemsets that have occurrence frequencies that are no less than a minimum support threshold. To efficiently solve the problem, a property called downward closure property was proposed [1]. This property helps to reduce the search space by pruning unpromising candidates early. However, FIM treats all items of a transaction database equally, that is as having the same importance. This assumption is a major drawback of FIM because for many real-world applications, items of a transaction database may have different importance degrees. For instance, breads and beverages are purchased much more often every day than cars. As result, FIM algorithms are more likely to discover the former itemset than the latter. But although the latter seldomly appears, its yields a much higher profit.

To address the above drawback of FIM, the task of high-utility itemset mining (HUIM) was proposed. The purpose of HUIM is to reveal itemsets that are useful or profitable (have a high utility) in a transaction database [2]. HUIM can be considered as an extension of FIM, and it has been the subject

| Abbreviations | Definition |
|---------------|-----------|
| EUCP | Estimated Utility Co-occurrence Pruning |
| EUCS | Estimated Utility Co-occurrence Structure |
| FCF | First Come, First Served |
| FIM | Frequent Itemset Mining |
| HDP | High-utility Database Projection |
| ITM | High-utility Transaction Merging |
| HUI | High-utility Itemset Mining |
| MCH-Miner | Multi-Core HUI Miner |
| minsup | Minimum Support threshold |
| minutil | Minimum Utility threshold |
| P-set | Projected set |
| TWDC | TWU Downward Closure model |
of many recent studies [3]–[7]. Traditional HUIM algorithms assume that each item has its own fixed utility value and a quantity value when appearing in a transaction. In practical applications, one can set these values based on his preferences and needs. The utility may represent information such as the weight, cost or profit of an item. An HUIM algorithm operates on databases with utility information to discover itemsets having a utility that is no less than a user-specified threshold, named the minimum utility (minutil). Itemsets meeting this constraint are called high-utility itemsets (HUI). These itemsets are useful for many real-world applications such as click stream analysis, user behavior analysis and cross marketing. Discovering high-utility itemsets is a combinatorial problem that is challenging. In FIM, the support measure is anti-monotonic and satisfies the downward closure property, which is useful to reduce the search space. However, since the utility measure in HUIM is not anti-monotonic nor monotonic, this property does not hold [7]. Thus, the search space of HUIM is huge, which leads to long execution times and high memory usage.

Table 2 presents an example transaction database having five transactions, denoted as $T_1$, $T_2$, . . . , $T_5$. Each transaction stores information about a set of purchased items. For instance, consider the transaction $T_4$ (the fifth row of Table 2). It indicates that a customer has purchased items $a$, $b$, $d$ and $e$ and that the purchase quantities of these items are 5, 2, 1 and 2, respectively. The database also stores information about the unit profits of items appearing in each transaction. In the case of $T_4$, the unit profits of $a$, $b$, $d$ and $e$ are 1, 2, 5 and 4, respectively. Based on this information given in each transaction, the utility value of an item in a transaction can be easily calculated as the product of its quantity by its unit profit. Furthermore, by summing up the utility values of every item of an itemset in all transactions that contains it, we obtain the utility value of that itemset for the whole database. For instance, consider the itemset $\{bc\}$, which is appearing in transactions $T_1$, $T_2$ and $T_3$. Its utility value is calculated as: $$(1 \times 2 + 2 \times 1) + (1 \times 2 + 3 \times 1) + (4 \times 2 + 1 \times 1) = 18,$$ whereas its support value (occurrence frequency) is 3. Similarly, the utility of itemset $\{de\}$ can be calculated as 22 and its support is 2. If the minutil threshold is set to 20 then $\{de\}$ is an HUI. This example clearly shows that the itemset that has a higher occurrence frequency is not always the one that yield the highest profit (utility), which in this case is $\{bc\}$.

FIM discovers FIs, that is itemsets having a support that is no less than the minimum support threshold (minsup). Similarly, in HUIM, the discovered HUIs are the itemsets having a utility that is no less than minutil. Since the proposal of HUIM, several algorithms were presented to discover the complete set of HUIs such as Two-Phase [8], IHUP [9], UPGrowth [10], EFIM [11].

However, all the algorithms mentioned above assume that the unit profit of each item is fixed. Take the database shown in Table 2 as example. It can be seen that item $d$, which is appearing in $T_1$, $T_2$, $T_3$ and $T_4$, has a fixed unit profit of 5 in all these transactions, and the same is true for the other items. The above assumption is unpractical for real-world databases and applications where the unit profit of items may vary over time due to various reasons. For instance, a change in supply cost or promotions, variable transportation fees or taxes may influence the unit profits of items. As a result, if traditional HUI mining algorithms are applied on real-world databases with dynamic profit values possible. This new database type is called dynamic profit database. Also, the authors proposed a compact format to store the transactions along with all their utility information [12]. Using this framework, all the currently available HUIM algorithms can now be applied to dynamic profit databases and generate accurate results. Table 3 presents an example of dynamic profit database, which is an extended version of the database given in Table 2, designed to store dynamic item profit values. For instance, consider item $a$. Its unit profit values are 1.0, 1.1, 1.1 and 1.2 when appearing in $T_2$, $T_3$, $T_4$ and $T_5$, respectively.

**TABLE 3. A sample transaction database having dynamic profit values.**

| TID | Transaction | Unit profit |
|-----|-------------|-------------|
| $T_1$ | b(1)c(2)d(1)c(1) | b(2)c(1)d(5) a(1) |
| $T_2$ | a(4)b(1)c(3)d(1)e(1) | a(1)b(2)c(1)d(5) e(4) |
| $T_3$ | a(4)c(2)d(1) | a(1)c(1)d(5) |
| $T_4$ | a(5)b(2)d(1)c(2) | a(1)b(2)d(5) e(4) |
| $T_5$ | a(3)b(4)c(1)f(2) | a(1)b(2)c(1)f(3) |

Along with the introduction of the dynamic utility framework, an efficient algorithm was also designed by Nguyen et al. to mine HUI from this new database type. The algorithm, named iMEFIM [12], extends the EFIM algorithm. It combines the newly introduced framework with all the efficient techniques presented in the EFIM algorithm. Furthermore, a novel data structure named P-set is utilized to significantly reduce the cost of database scans.

The following sections are organized as follows. Section II briefly reviews relevant related work on HUIM. Section III describes the problem of mining HUIs in dynamic profit databases and presents important definitions. Section IV discusses parallelism and presents an extension of the iMEFIM algorithm to further boost its performance using parallel computing. Section V describes the evaluation of the extended algorithm by comparing its performance with that of the
original algorithm. Finally, a conclusion is drawn and future work are discussed in Section VI. The abbreviations frequently used in this manuscript is presented in Table 1.

II. RELATED WORK
As mentioned above, HUIM is more challenging than FIM because the utility measure does not satisfy the downward closure property as stressed by Yao and Hamilton [7], [13]. Thus, it cannot be used to prune candidate patterns. To reduce the search space of the HUIM problem, efficient pruning strategies and utility-based upper-bounds are required [14]. The first HUIM algorithm based on this idea is UMiner [7]. However, this algorithm is incomplete, i.e. it may not discover the complete set of HUIs in a transaction database. To address this issue, the Transaction Weighted Utilization (TWU), a safe utility-based upper-bound, was proposed and adopted in several algorithms [8], [9], [15]. It is used to efficiently and safely pruning candidates. Since the TWU is anti-monotonic, it satisfies the downward closure property. Based on this property, Liu et al. designed the TWU downward closure model (TWDC) [8] to prune unpromising candidate patterns from the search space early, and hence reduce the time for discovering HUIs. Combining the Apriori algorithm with the TWDC model, Liu et al. has introduced an algorithm named Two-Phase [8]. As the name implies, the algorithm requires two separated phases to mine HUIs. First, the algorithm obtains a list of candidates having TWU no less than the $\text{minutil}$ threshold. Next, a database scan is performed to determine the exact utility value of each candidate. The algorithm returns only the patterns having utility values greater or equal to the $\text{minutil}$ threshold. Thus, Two-Phase can return the complete set of HUIs. However, it can still have very long runtimes. By adopting the proposed TWDC model, many algorithms were then introduced to increase the effectiveness of the HUIM process by lowering the number of candidates generated when using the TWU [16]–[18]. Based on the concept of pattern growth tree introduced by Han et al. [19], algorithms such as IHUP [9], UP-Growth [10] were proposed. These algorithms all mine HUIs in two phases, first they generate candidate patterns, and then they scan the database to compute the exact utility of each candidate and filter those having a low utility. Besides, such algorithms suffer from a scalability issue since the number of generated candidates is huge when it is applied to large databases. The HUI-Miner algorithm, which was introduced by Liu and Qu [3], overcomes this limitation by mining HUIs in one phase without candidate generation. HUI-Miner requires only one database scan. To achieve this, the authors presented an efficient structure named Utility-List. Although that structure can be employed to efficiently prune candidates, the join operation on these lists is computationally expensive and not suitable for large databases. Fournier-Viger et al. then proposed an extended version of the HUI-Miner algorithm, named FHM [20]. The algorithm employs an Estimated Utility Co-occurrence Structure (EUCS) and a pruning strategy called EUCP to greatly reduce the mining time and number of database scans. Also extended from HUI-Miner, Krishnamoorthy proposed an algorithm named HUP-Miner [21]. The algorithm efficiently prunes unpromising candidates by extending the utility-list structure into a partitioned utility-list structure. It also utilizes a look-ahead strategy to minimize the time needed to construct utility lists. The goal of all the mentioned HUIM approaches is to increase the performance of the mining process by pruning candidates. A list-based algorithm named DHUPL (Damped High Utility Pattern mining based List) was presented to mine HUIs in data streams [22]. DHUPL prunes patterns using a damped window model that considers newly arrived data as more important than previous data [22]. Also, to handle dynamic databases where transactions are frequently deleted, Yun et al. adapted a pre-large method to mine HUIs efficiently while reducing the number of database scans to update results [23]. The SPHUI-Miner [24] algorithm, proposed by Bai et al. introduced an efficient and compact data format named HUI-TRPL to reduce memory consumption. The authors also presented two novel data structures, which are selective database projection list and Tail-Count list to prune the search space and to reduce the time needed to scan the database. Recently, Zida et al. introduced the EFIM algorithm [11], which is considered the state-of-the-art algorithm for mining HUIs. The authors presented the sub-tree utility and local utility, two novel and tighter upper bounds to efficiently prune generated candidates. Furthermore, two new techniques to deal with long transactions and partially identical transactions were presented. They are called High-utility Database Projection (HDP) and High-utility Transaction Merging (HTM). By employing these efficient techniques and strategies, EFIM was shown to outperform all the previously introduced algorithms. But EFIM performs a complete database projection for each promising itemset, which requires a long execution time and has scalability issue. To mine dynamic profit databases and take into account the dynamic aspect of real-world transaction databases, Nguyen et al. proposed a new utility framework [12]. Furthermore, a novel structure named P-set was also introduced to significantly reduce the cost of database scans. Based on the proposed techniques, the authors extended the EFIM algorithm into a more efficient algorithm, named iMEFIM [12]. Evaluation results have proven that iMEFIM has superior performance and scalability compared to the previous methods. The concept of HUIs was also extended in many algorithms. The CPHUI-List algorithm to efficiently mine closed potential HUIs [25] in uncertain databases without generating candidates. Gan et al. proposed the HUOPM algorithm (high-utility occupancy pattern mining) [26] that takes into account user preferences, such as frequency, utility, and occupancy when mining HUIs. To prune the search space, several data structures were introduced as a global and partially downward closure properties, such as a novel frequency-utility tree, utility-occupancy list and frequency-utility table. With the data obtained from the IoT devices, which contain positive, negative unit utilities
and uncertain, Gan et al. proposed an algorithm namely HUPNU [27] to discover high-utility pattern having these characteristics. The discovered patterns can be used in many applications, such as intrusion detection, risk prediction, decision-making, etc.

Many efficient data structures have been proposed to improve the performance of the mining process. To speed up the process of mining association rules regardless of the algorithm used, Luna et al. presented a novel data structure [28]. This approach is designed to be employed by a wide range of existing methods without changing their original process. Experimental results have shown that the introduced structure has enhanced the runtime by orders of magnitude and significantly reduced memory requirements. Vo et al. inherited a lattice-based approach [29], [30] to mine high utility association rules [31] and non-redundant high utility association rules [32].

As presented above, many algorithms were designed to mine HUIs efficiently. But those algorithms are mainly sequential algorithms and many of those are time-consuming when applied on specific types of database such as high-density databases and databases with long transactions. An approach to enhance the performance of pattern mining is to apply parallel computing models such as distributed, GPU and multi-core computing. With the increased popularity and availability of novel microprocessors that allows the simultaneous execution of multiple tasks to improve performance [33], many studies have introduced parallel methods for FIM such as pSPADE [34], Par-CSP [35] and Par-ClosP [36]. Based on their successes, it can be assumed that adapting HUIM algorithms to run on a multi-core architecture would also greatly boost their performance. Yu et al. introduced an effective load balancing strategy into FIM based on a parallel architecture [37], as well as Li et al. [33], Negrevergne et al. [38], Schlegel et al. [39] and Negrevergne et al. [40]. Parallel mining has been also applied to correlated pattern mining [41], class association rule mining [42], sequential pattern mining [43], sub-graph mining [44], and frequent sequential pattern mining [45]. However, these techniques have been rarely applied in HUIM. Chen and An introduced a parallelized version of the HUI-Miner algorithm, named PHUI-Miner [46]. The algorithm partitions the search space into sub-spaces that can be explored in parallel. Each sub-space is in turn assigned to a cluster node. This approach increases the overall performance of the mining process in terms of runtime. Nguyen et al. extended the EFIM algorithm using multi-core parallelism and proposed the pEFIM algorithm [47].

### III. PRELIMINARIES AND PROBLEM STATEMENT

#### A. PROBLEM STATEMENT

Let there be a dynamic profit database \( \mathcal{D} \) and a user-specified minimum utility threshold (\( \text{minutil} \)). The problem of mining high-utility itemsets in \( \mathcal{D} \) is to efficiently discover all the itemsets in \( \mathcal{D} \) having a utility that is no less than \( \text{minutil} \).

#### B. PRELIMINARIES

A transaction database \( \mathcal{D} \) is a set of transactions, denoted as \( \mathcal{D} = \{T_1, T_2, \ldots, T_n\} \). Let \( \mathcal{I} \) be the finite set of all distinct items contained in \( \mathcal{D} \), and \( X \subseteq \mathcal{I} \) be an itemset (\( X \subseteq \mathcal{I} \)). A transaction \( T_c \), \( T_c \subseteq \mathcal{I}, T_c \in \mathcal{D} \), has a unique transaction ID (\( TID_c \)). Consider a single item \( i \in \mathcal{I} \) contained in a transaction \( T_c \). The quantity of \( i \) in \( T_c \) is denoted as \( q(i, T_c) \). Furthermore, the unit profit of \( i \) in \( \mathcal{D} \) is denoted as \( pr(i) \). Many traditional approaches for HUIM assume that unit profits remain the same for all transactions of a database \( \mathcal{D} \).

**Definition 1:** The utility of an item \( i \) in a transaction \( T_c \), denoted as \( u(i, T_c) \), is calculated by the following formula:

\[
u(i, T_c) = q(i, T_c) \times pr(i)[11].\]

**Definition 2:** The transaction utility of a transaction \( T_c \), denoted as \( u(T_c) \) is calculated as:

\[
u(T_c) = \sum_{i \in T_c} u(i, T_c)[11].\]

**Definition 3:** Let \( X \) be an itemset in a transaction \( T_c \). Then, the utility of \( X \) in \( T_c \), denoted as \( u(X, T_c) \), is computed as:

\[
u(X, T_c) = \sum_{i \in X \wedge X \subseteq T_c} u(i, T_c)[11].\]

**Definition 4:** The utility of \( X \) in the database \( \mathcal{D} \), denoted as \( u(X) \), is defined as:

\[
u(X) = \sum_{c \in \mathcal{D}} u(X, T_c)[11].\]

**Definition 5:** An itemset \( X \) is considered to be an HUI if and only if \( u(X) \geq \text{minutil} \), otherwise, \( X \) is a low-utility itemset.

A problem with the above definition of an item’s utility is that it assumes that unit profits are fixed, which does not reflect the true nature of real-world databases. Thus, to address this issue, Nguyen et al. have introduced a new framework for handling dynamic profit databases [12]. With this new framework, the utility of a single item \( i \) appearing in a transaction \( T_c \) is defined as follows.

**Definition 6:** The utility of an item \( i \) in a transaction \( T_c \) is defined as \( u(i, T_c) = q(i, T_c) \times pr(i, T_c) \), where \( pr(i, T_c) \) represents the unit profit of item \( i \) when appearing in transaction \( T_c \).

By employing this new framework, all the existing HUIM algorithms can now be applied on the new type of databases where unit profits may vary. For instance, consider item \( \{a\} \) and the database of Table 3. The item \( \{a\} \) appears in \( T_2, T_3, T_4, \) and \( T_5 \). Its utility is calculated as:

\[
u(X) = u(a, T_2) + u(a, T_3) + u(a, T_4) + u(a, T_5) = 4.0 + 4.4 + 5.5 + 3.6 = 17.5.\]

To minimize the number of database scans for mining HUIs, Nguyen et al. proposed the concept of P-set and used it to limit the number of scanned transactions when considering an itemset \( X \) [12].

**Definition 7:** The P-set of an itemset \( X \), denoted as \( P - \text{set}(X) \), is defined as follows. \( P - \text{set}(X) = \{c | X \subseteq T_c \} \). For instance, for the database depicted in Table 3, \( P - \text{set}(\{cd\}) = \{1, 2, 3\} \) and \( P - \text{set}(\{a\}) = \{2, 3, 4, 5\} \).

**Definition 8:** The P-set of the itemset obtained by extending an itemset \( X \) with an item \( i \) is denoted as \( Pex - \text{set}(X, i) \), and is defined as follows: \( Pex - \text{set}(X, i) = P - \text{set}(X) \cap P - \text{set}(i) \) [12]. For example, \( Pex - \text{set}(\{cd\}, \{a\}) = \{1, 2, 3\} \cap \{2, 3, 4, 5\} = \{2, 3\} \) and \( P - \text{set}(\{acd\}) \).
Furthermore, Nguyen et al. also proposed a more compact representation of the dynamic database $D$ [12], which is shown in Table 4. This new representation helps reducing the required storage space and the time needed to calculate the utility of single items.

### TABLE 4. Compact format of the dynamic database from Table 3.

| TID | Transaction | Transaction utility |
|-----|-------------|---------------------|
| $T_1$ | b[2] c[2] d[5] g[1] | 10 |
| $T_2$ | a[4] b[1.9] c[2.7] d[4.8] e[4] | 17.4 |
| $T_3$ | a[4.4] c[2.2] d[5.5] | 12.1 |
| $T_4$ | a[5.5] b[4.4] d[5.5] e[8.8] | 24.2 |
| $T_5$ | a[3.6] b[9.6] c[1.2] f[7.2] | 21.6 |

During the HUI mining process, the TWDC model is used to safely reduce the search space by pruning unpromising candidates. This model was widely applied in previous studies [8], [9], [15]–[18]. The definition of TWU is given below.

**Definition 9:** The TWU of an itemset $X$ in a database $D$, denoted as $TWU(X)$, is defined as: $TWU(X) = \sum_{T \in D \times X \subseteq T} u(T)$ [8], [11]. For instance, $TWU([e]) = u(T_2) + u(T_4) = 17.4 + 24.2 = 41.6$.

**Definition 10:** Let $\succ$ be the ascending order of TWU, the set containing all single items $r$ that can be used to extend $X$, denoted as $E(X)$, is defined as follows. $E(X) = \{r | r \in J \land r \succ x, x \in X\}$ [11], where $\succ$ is a total order on items.

However, the TWU is considered as not being tight enough to prune many candidates, especially in large databases [11]. Thus, Zida et al. has proposed two new and tighter upper-bounds on the utility measure with respect to an item $z \in E(X)$, namely the sub-tree utility and local utility of an itemset $X$, denoted as $su(X, z)$ and $lu(X, z)$, respectively [11]. Based on the definitions of sub-tree and local utility, the definitions of primary and secondary items were also proposed by Zida et al. [11].

**Definition 11:** Let $X$ be an itemset, $X \subseteq J$. Primary and secondary items of $X$, denoted as $P(X)$ and $S(X)$ respectively, are defined as follows. $P(X) = \{i | i \in E(X) \land su(X, i) \geq \minutil\}$, $S(X) = \{i | i \in E(X) \land lu(X, i) \geq \minutil\}$ [11].

### IV. THE PROPOSED ALGORITHM

#### A. THE MCH-MINER ALGORITHM

According to Zaki [48], applying parallelism into a data mining process has several benefits but also brings several challenges. A single processor suffers from the bottleneck caused by the limitation of processing power, memory, etc., while multiple processors can solve these issues efficiently by executing tasks in parallel. Multiple processors are now widely available under the form of multi-core processors. Furthermore, many databases are split into several parts and stored on distributed servers. Serial data mining algorithms executed on a single-core processor are unsuitable for this situation. However, adapting current data mining methods for parallel execution brings new challenges such as load balancing, communication minimization and synchronization.

Zaki has pointed out three well-known approaches to support parallelism. They are task parallelism, data parallelism, and hybrid task/data parallelism. The task parallelism model partitions the search space of the algorithm into sub-spaces and then assigns each sub-space to a separated processor. All these processors operate on the same data. There are two task parallelism approaches: (i) divide and conquer, which divides the search space and assign each sub-space to a processor that handles it. (ii) task queue strategy, which dynamically assigns small portion of the search space to a processor whenever it becomes available. Data parallelism assign parts of a database to several processors, which are then concurrently processed using a same algorithm executed on a distributed computing platform. The hybrid task/data parallelism approach consists of performing a pipeline of tasks, in which each task may be using data parallelism and the output of one task is used as input of the next one.

This paper focuses onto task parallelism to improve the performance of the HUI mining process. A novel parallel algorithm is designed, called MCH-Miner. The aim is to put into use the current generation of multi-core processors to speed-up the pattern mining process. The algorithm adopts a divide and conquer strategy, which was deemed the most suitable for this problem. Consider the four items $a$, $b$, $c$ and $d$, from the running example. Figure 1 illustrates the search space of the sequential HUI algorithm iMEFIM as a set-enumeration tree. It contains 15 non empty itemsets.

![FIGURE 1. A set-enumeration tree representing the search space of the iMEFIM algorithm.](image-url)
Algorithm 1 The MCH-Miner

Input: $\mathcal{D}$: transaction database, $\text{minutil}$: minimum utility threshold
Output: Discovered HUIs from $\mathcal{D}$
1. $X \leftarrow \emptyset$; $\text{HUIs} \leftarrow \emptyset$;
2. Scan $\mathcal{D}$ to determine $lu(X, i), \forall i \in \mathcal{I}$;
3. $S(X) = \{i \in \mathcal{I} \land lu(X, i) \geq \text{minutil}\}$;
4. Sort $S(X)$ in ascending order of $lu(X, i)$;
5. FOR EACH transaction $T$, remove $i \notin S(X)$ from $T$.
   Delete null transactions;
6. FOR EACH $T \in \mathcal{D}$, sort $T$ in ascending order of $lu(X, i)$
   when read backward;
7. Scan $\mathcal{D}$ to determine $su(X, i).Pex-set(X, i), \forall i \in S(X)$;
8. $\mathcal{P}(X) = \{i \in S(X) \land su(X, i) \geq \text{minutil}\}$;
9. FOR EACH $i \in \mathcal{P}(X)$ DO PARALLEL
10. $\mathcal{P}_i(X) = \{i\}$;
11. $\text{HUIs} \cup = \text{Search}(i, \mathcal{D}, \mathcal{P}_i(X), S(X), Pex-set(X, i), \text{minutil})$;
12. END FOR;
13. RETURN $\text{HUIs}$;

Algorithm 2 The Search Function

Input: $X$: itemset,
$\mathcal{D}$: projected database,
$\mathcal{P}(X)$: primary items,
$S(X)$: secondary items,
Pex-set($X$, $i$): $P$-set extension of $X$, $\text{minutil}$.
Output: Discovered HUIs from the search space.
1. $\text{HUIs} \leftarrow \emptyset$;
2. FOR EACH $i \in \mathcal{P}(X)$ DO
3. $\beta \leftarrow X \cup \{i\}$;
4. Scan $\mathcal{D}$ using $\text{Pex-set}(X, i)$ to determine $u(\beta)$
   and construct $\mathcal{D}_\beta$;
5. IF $u(\beta) \geq \text{minutil}$ THEN $\text{HUIlist} \cup = \beta$;
6. Scan $\mathcal{D}_\beta$ to determine $su(\beta, z), lu(\beta, z),$
   $Pex-set(\beta, z), \forall z \in S(X)$ of each item $z$ succeeding $i$;
7. $\mathcal{P}(\beta) = \{z \in S(X) | su(\beta, z) \geq \text{minutil}\}$;
8. $S(\beta) = \{z \in S(X) | lu(\beta, z) \geq \text{minutil}\}$;
9. $\text{HUIs} \cup = \text{Search}(\beta, \mathcal{D}_\beta, \mathcal{P}(\beta), S(\beta),$
   $Pex-set(\beta, z), \text{minutil})$;
10. END FOR;
11. RETURN $\text{HUIs}$;

B. ADDRESSING THE LOAD BALANCING PROBLEM

Many parallel algorithms apply load balancing. It consists of
assigning approximately an equal amount of work (tasks) to
each process (e.g. processor) to ensure that all processes are
kept busy. This can be viewed as a problem of minimizing
idle time. Load balancing is important for parallel algorithms
since it can improve the overall performance when many
activities can be processed concurrently. Activities should be
assigned to separated processes of different machines, sepa-
rated processes on the same machine, or separated threads of
a same process.

For the proposed MCH-Miner algorithm, which adopts the
multi-core parallel computing model, tasks are executed on a
single multi-core processor. Hence, load balancing is quite
simple, since all the cores share the same memory space
(using the Shared Memory System) and all the tasks that
need to be executed are separated and their search spaces are
disjoint. Thus, these tasks are put into a task-pool having
a size equals to the number of available processor cores. Then,
the tasks are concurrently executed by processor cores, using
the first come-first served (FCFS) scheduling strategy, until
all the tasks have been completed.

In the multi-core implementation of an algorithm, it is a
good practice to start a task-pool of size equal to the
number of logical/physical processors available in the system
and to execute tasks from within the pool. This is to avoid
taking up all system resources for all the allocated tasks,
as $|\mathcal{P}(X)|$ could become very large when scanning large
databases, and by executing all the tasks associated with each

FIGURE 2. The partitioned search space used by MCH-Miner algorithm.
item \( i \in \mathcal{P}(X) \) at once without using a task-pool would greatly degrades the algorithm’s performance.

V. EXPERIMENTAL EVALUATION

Experiments were performed to evaluate the performance of the proposed MCH-Miner algorithm. Those experiments were designed to compare the performance of \textit{iMEFIM} with that of its parallel version \textit{MCH-Miner} in terms of execution time and scalability. The evaluation was conducted on several dynamic profit databases. Evaluations were performed using a personal computer equipped with a dual-core (four logical processors) Intel® Core i5 processor @ 2.7Ghz.
B. Vo et al.: Multi-Core Approach to Efficiently Mining High-Utility Itemsets

FIGURE 4. Scalability test on the four benchmark datasets.

8.0GB of 1866Mhz LPDDR3 RAM, 128GB PCIe-based flash storage for minimizing the I/O latency and running macOS 10.13.2. The algorithms used in the experiments were all implemented in Java using the SDK 8.0.

The experiments were carried out using standard databases that are used in HUIM studies to benchmark the performance of HUIM algorithms. The databases are Chainstore, Retail, Foodmart, Kosarak, Mushroom, Accidents, Connect and Chess. They can be obtained from the SPMF Open-source Data Mining Library (SPMF) [49] at the following URL: https://bit.ly/2xg2Nzr. For each database, the quantity of each item appearing in each transaction is randomly generated in the [1, 10] interval, while each item’s utility is randomly generated in the [2, 50] interval. To make the database contain dynamic profit values, the utility of each item in each transaction is varied by 2% to 10% compared to the original value. The Retail and Chainstore databases contain real utility values.

Table 5 presents the characteristics of the databases used in the experiments.

A. RUNTIME

The experiments were done to compare the performance of iMEFIM with that of its parallelized version, MCH-Miner. For each database, we varied the minutil threshold and recorded the runtime of each algorithm. Results are presented in Figure 3. In those experiments, the following databases were used: Chainstore, Retail, Foodmart, Kosarak, Chess, Mushroom, Accidents, and Connect, as shown in Figure 3a to Figure 3h, respectively. The experiments were completed by using 4 logical processor cores. It was found that the parallel MCH-Miner algorithm provides a big performance improvement compared to its sequential version, the iMEFIM algorithm. For high-density databases such as Accidents, Chess, Mushroom, and Connect, MCH-Miner was up to three times faster than iMEFIM. For large and sparse databases such as Chainstore and Kosarak, the execution time of MCH-Miner algorithm was also reduced by up to three times compared to iMEFIM. This can be explained that the iMEFIM has efficiently solved the problem of mining HUIs on sparse databases and thus the speed gained by using multi-core processing is not as high as on the high-density databases. On overall, it was observed that the performance of MCH-Miner remains better than iMEFIM, as the minutil threshold is lowered on each database. Thus, it is concluded that the MCH-Miner algorithm outperforms iMEFIM on all the test databases, especially on high-density databases.

Table 5. Characteristics of the benchmark databases.

| Database       | #Transactions | #Items | Avg. length |
|---------------|---------------|--------|-------------|
| Chainstore    | 1,112,949     | 46,086 | 7.2         |
| Kosarak       | 990,002       | 41,270 | 8.1         |
| Accidents     | 340,183       | 468    | 33.8        |
| Retail        | 87,943        | 16,465 | 10.3        |
| Foodmart      | 4,141         | 1,559  | 4.4         |
| Chess         | 3,196         | 75     | 37          |
| Mushroom      | 8,124         | 119    | 23          |
| Connect       | 67,557        | 129    | 43          |

TABLE 5. Characteristics of the benchmark databases.
B. SCALABILITY

To further assess the performance of the MCH-Miner algorithm, scalability tests were done on two spare databases Chainstore, Retail and two high-density databases Mushroom and Accidents. These datasets were replicated from 3 to 5 times their original transactions count. The suffix of each database name denotes the expanded size of the original database, i.e. Chainstore3x indicates the total number of transactions in the Chainstore database has been replicated 3 times, increased from 1,112,949 to 3,338,847. Specifically, Chainstore, Retail and Accidents had their total transactions multiplied 3 times and 5 times for the Mushroom database, hence we name them as Chainstore3x, Retail3x, Accidents3x and Mushroom5x. In this test, we fixed the minutil threshold on each test database and varied the number of transactions from 25% to 100% of their size. The runtime of each algorithm on the test databases were recorded and presented in Figure 4a to Figure 4d. It can be seen in all tests that the MCH-Miner has better scalability with respect to the number of transactions compare to that of iMEFIM when using 4 logical processor cores. The MCH-Miner scalability is almost linear when we increased the number of transactions in each database, especially for the case of the Chainstore database, it has linear scalability. The iMEFIM although has been proved to have the best scalability in [12]. Thus, the MCH-Miner algorithm, which is based on iMEFIM inherits this characteristic. Overall, the MCH-Miner was shown that it has better scalability than iMEFIM on all the test databases.

VI. CONCLUSIONS AND FUTURE WORK

In this study, we extended the state-of-the-art iMEFIM algorithm for mining HUIs by combining all its techniques and strategies with the multi-core parallel computing model. That model was adapted to exploit the full computing power of the current generation of processors. The result is a new and high-performance algorithm, named MCH-Miner, that mines HUIs in a dynamic profit database in parallel. Extensive experiments were done on eight benchmark databases. Results have shown that the proposed parallel algorithm is up to three times faster than the sequential iMEFIM algorithm on the test databases when using four processor cores and has better scalability. In the future, we intend to extend the proposed MCH-Miner algorithm using a distributed computing platform such as Apache Hadoop or Spark, to mine HUIs in large-scale databases.

REFERENCES

[1] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in Proc. 20th Int. Conf. Very Large Data Bases, 1994, pp. 487–499.
[2] W. Gan, C.-W. Lin, P. Fournier-Viger, H.-C. Chao, V. Tseng, and P. Yu, “A survey of utility-oriented pattern mining,” IEEE Trans. Knowl. Data Eng., early access, Sep. 20, 2019, doi: 10.1109/TKDE.2019.2942594.
[3] M. Liu and J. Qu, “Mining high utility items without candidate generation,” in Proc. 21st ACM Int. Conf. Inf. Knowl. Manage. (CIKM), 2012, pp. 55–64.
[4] J. Liu, X. Ju, X. Zhang, B. C. M. Fung, X. Yang, and C. Yu, “Incremental mining of high utility patterns in one phase by absence and legacy-based pruning,” IEEE Access, vol. 7, pp. 74168–74180, 2019.
[5] Q. Zhang, W. Fang, J. Sun, and Q. Wang, “Improved genetic algorithm for high-utility itemset mining,” IEEE Access, vol. 7, pp. 176799–176813, 2019.
[6] W. Song and C. Huang, “Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework,” IEEE Access, vol. 6, pp. 19568–19582, 2018.
[7] H. Yao, H. J. Hamilton, and C. J. Butz, “A foundational approach to mining itemset utilities from databases,” in Proc. SIAM Int. Conf. Data Mining, vol. 4, Apr. 2004, pp. 482–486.
[8] Y. Liu, W. Liao, and A. Choudhary, “A two-phase algorithm for fast discovery of high utility itemsets,” in Proc. 9th Pacific-Asia Conf. Adv. Knowl. Discovery Data Mining, 2005, pp. 689–695.
[9] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y.-K. Lee, “Efficient tree structures for high utility pattern mining in incremental databases,” IEEE Trans. Knowl. Data Eng., vol. 21, no. 12, pp. 1708–1721, Dec. 2009.
[10] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, “UP-growth: An efficient algorithm for high utility itemset mining,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2010, pp. 252–262.
[11] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, “EFIM: A fast and memory efficient algorithm for high-utility itemset mining,” Knowl. Inf. Syst., vol. 51, no. 2, pp. 595–625, May 2017.
[12] L. T. T. Nguyen, P. Nguyen, T. D. D. Nguyen, B. Vo, P. Fournier-Viger, and V. S. Tseng, “Mining high-utility itemsets in dynamic profit databases,” Knowl.-Based Syst., vol. 175, pp. 130–144, Jul. 2019.
[13] H. Yao and H. J. Hamilton, “Mining itemset utilities from transaction databases,” Data Knowl. Eng., vol. 59, no. 3, pp. 603–626, Dec. 2006.
[14] J. M.-T. Wu, J. C.-W. Lin, and A. Tamrakar, “High-utility itemset mining with effective pruning strategies,” ACM Trans. Knowl. Discovery Data Anal., vol. 13, no. 6, pp. 1–22, Dec. 2019.
[15] U. Yun, H. Ryang, and K. H. Ryu, “High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates,” Expert Syst. Appl., vol. 41, no. 8, pp. 3861–3878, Jan. 2014.
[16] B. Le, H. Nguyen, T. A. Cao, and B. Vo, “A novel algorithm for mining high utility itemsets,” in Proc. 1st Asian Conf. Intell. Inf. Database Syst., 2009, pp. 13–17.
[17] B. Vo, H. Nguyen, T. B. Ho, and B. Le, “Parallel method for mining high utility itemsets from vertically partitioned distributed databases,” in Proc. 13th Int. Conf. Knowl.-Based Inf. Eng. Syst., 2009, pp. 251–260.
[18] B. Le, H. Nguyen, and B. Vo, “An efficient strategy for mining high utility itemsets,” Int. J. Intell. Inf. Database Syst., vol. 5, no. 2, pp. 164–176, 2011.
[19] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining frequent patterns without candidate generation: A frequent-pattern tree approach,” Data Mining Knowl. Discovery, vol. 8, no. 1, pp. 53–87, Jan. 2004.
[20] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, “FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning,” in Proc. Int. Symp. Methods, Tools, and Applications (MTA), Lecture Notes in Computer Science), vol. 8502, 2014, pp. 83–92.
[21] S. Krishnamoorthy, “Pruning strategies for mining high utility itemsets,” Expert Syst. Appl., vol. 42, no. 5, pp. 2371–2381, Apr. 2015.
[22] H. Nam, U. Yun, B. Vo, T. Truong, Z.-H. Deng, and E. Yoon, “Efficient approach for dumped window-based high utility pattern mining with list structure,” IEEE Access, vol. 8, pp. 50958–50968, 2020.
[23] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. Truong, B. Vo, and W. Pedrycz, “Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases,” Future Gener. Comput. Syst., vol. 103, pp. 58–78, Feb. 2020.
[24] A. Bai, P. S. Deshpande, and M. Dhabu, “Selective database projections based approach for mining high-utility itemsets,” IEEE Access, vol. 6, pp. 14389–14409, 2018.
[25] B. Vo, L. T. T. Nguyen, N. Bui, T. D. D. Nguyen, V.-N. Huynh, and T.-P. Hong, “An efficient method for mining closed potential high-utility itemsets,” IEEE Access, vol. 8, pp. 31813–31822, 2020.
[26] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, “HUOPM: High-utility occupancy pattern mining,” IEEE Trans. Cybern., vol. 50, no. 3, pp. 1195–1208, Mar. 2020.
[27] W. Gan, J. C.-W. Lin, H.-C. Chao, A. V. Vasilakos, and P. S. Yu, “Utility-driven data analytics on uncertain data,” IEEE Trans. Syst., vol. 23, Mar. 2020, doi: 10.1109/TSYST.2020.2979279.
[28] J. M. Luna, A. Cano, M. Pechenizkiy, and S. Ventura, “Speeding-up association rule mining with inverted index compression,” IEEE Trans. Cybern., vol. 46, no. 12, pp. 3059–3072, Dec. 2016.
[29] B. Vo and B. Le, “Mining traditional association rules using frequent itemsets lattice,” in Proc. Int. Conf. Comput. Ind. Eng., Jul. 2009, pp. 1401–1406.

85898 VOLUME 8, 2020
[30] B. Vo, T.-P. Hong, and B. Le, “A lattice-based approach for mining most generalization association rules,” Knowl.-Based Syst., vol. 45, pp. 20–30, Jun. 2013.

[31] T. Mai, B. Vo, and L. T. T. Nguyen, “A lattice-based approach for mining high utility association rules,” Inf. Sci., vol. 399, pp. 81–97, Aug. 2017.

[32] T. Mai, L. T. T. Nguyen, B. Vo, U. Yun, and T.-P. Hong, “Efficient algorithm for mining non-redundant high-utility association rules,” Sensors, vol. 20, no. 4, p. 1078, 2020.

[33] E. Li, L. Liu, Y. Zhang, and Z. Tang, “Optimization of frequent itemset mining on multiple-core processor,” in Proc. 33rd Int. Conf. Very Large Data Bases, 2007, pp. 1275–1285.

[34] S. Alias and N. M. Norwawi, “pSPADE: Mining sequential pattern using personalized support threshold value,” in Proc. Int. Symp. Inf. Technol., vol. 2, 2008, pp. 1–8.

[35] S. Cong, J. Han, and D. Padua, “Parallel mining of closed sequential patterns,” in Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), 2005, pp. 562–567.

[36] T. Zhu and S. Bai, “A parallel mining algorithm for closed sequential patterns,” in Proc. 21st Int. Conf. Adv. Inf. Netw. Appl. Workshops (AINAW), vol. 1, 2007, pp. 392–395.

[37] K.-M. Yu and S.-H. Wu, “An efficient load balancing multi-core frequent patterns mining algorithm,” in Proc. IEEE 10th Int. Conf. Trust, Secur. Privacy Comput. Commun., Nov. 2011, pp. 1408–1412.

[38] B. Negrevergne, A. Termier, J. Méhaut, and T. Uno, “Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses,” in Proc. Int. Conf. High Perform. Comput. Simulation, Jun. 2010, pp. 521–528.

[39] B. Schlegel, T. Karnagel, T. Kiefer, and W. Lehner, “Scalable frequent itemset mining on many-core processors,” in Proc. 9th Int. Workshop Data Manage. New Hardw. (DaMoN), 2013, p. 8.

[40] B. Negrevergne, A. Termier, M.-C. Rousset, and J.-F. Méhaut, “ParaMiner: A generic pattern mining algorithm for multi-core architectures,” Data Mining Knowl. Discovery, vol. 28, no. 3, pp. 593–633, 2014.

[41] A. Casali and C. Ernst, “Extracting correlated patterns on multicore architectures,” in Availability, Reliability, and Security in Information Systems and HCI, 2013, pp. 118–133.

[42] D. Nguyen, B. Vo, and B. Le, “Efficient strategies for parallel mining class association rules,” Expert Syst. Appl., vol. 41, no. 10, pp. 4716–4729, Aug. 2014.

[43] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, “A survey of parallel sequential pattern mining,” ACM Trans. Knowl. Discovery Data, vol. 13, no. 3, Jun. 2019, Art. no. 34.

[44] B. Huynh, D. Nguyen, and B. Vo, “Parallel frequent subgraph mining on multi-core processor systems,” ICIC Express Lett., vol. 10, no. 9, pp. 2105–2113, Sep. 2016.

[45] B. Huynh, B. Vo, and V. Snasel, “An efficient method for mining frequent sequential patterns using multi-core processors,” Int. J. Speech Technol., vol. 46, no. 3, pp. 703–716, Apr. 2017.

[46] Y. Chen and A. An, “Approximate parallel high utility itemset mining,” Big Data Res., vol. 6, pp. 26–42, Dec. 2016.

[47] T. D. D. Nguyen, L. T. T. Nguyen, and B. Vo, “A parallel algorithm for mining high utility itemsets,” in Proc. 39th Int. Conf. Int. Conf. Syst. Archit. Technol., vol. 853, 2019, pp. 286–295.

[48] M. J. Zaki, “Parallel and distributed association mining: A survey,” IEEE Concurrency, vol. 7, no. 4, pp. 14–25, Oct. 1999.

[49] P. Fournier-Viger et al., “The SPMF open-source data mining library version 2,” in Proc. ECML/PKDD, in Lecture Notes in Computer Science, vol. 9853, 2016, pp. 36–40.

BAY VO (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2002, 2005, and 2011, respectively. He is currently an Associate Professor and the Dean of Faculty of Information Technology, Ho Chi Minh City University of Technology, Vietnam, and a Visiting Professor with the College of Electronics and Information Engineering, Sejong University, Seoul, South Korea. His research interests include association rules, classification, mining in incremental database, distributed databases, and privacy preserving in data mining.

LOAN T.T. NGUYEN received the B.Sc. and M.Sc. degrees in computer science from Vietnam National University, Ho Chi Minh City, Vietnam, in 2002 and 2008, respectively, and the Ph.D. degree in computer science from the Wroclaw University of Technology, Poland, in 2015. From October 2016 to September 2017, she was an ERCIM Postdoctoral Researcher with the University of Warsaw, Poland. She was a Visiting Researcher with NTNU, Norway, from March 5, 2017 to March 19, 2019. Her research interests include association rules, classification, and mining in incremental databases.

TRINH D.D. NGUYEN received the B.Sc. and M.Sc. degrees in information technology and computer science from Military Technology Academy, Ho Chi Minh Open University, in 2001 and 2017, respectively. He is currently pursuing the Ph.D. degree with the University of Information Technology, VNU-HCM, Vietnam. His research interests include association rules, classification, high utility pattern mining, frequent pattern mining, machine learning, and distributed computing.

PHILIPPE FOURNIER-VIGER received the Ph.D. degree in computer science from the University of Quebec, Montreal, in 2010. He is currently a Professor with the Shenzhen Graduate School, Harbin Institute of Technology, China. He is also the Founder of the popular SPMF Open-Source Data Mining Library, which has been cited in over 430 research articles since 2010. He has published over 140 research articles in refereed international conferences and journals, which have received over 1300 citations. His research interests include data mining, pattern mining, sequence analysis and prediction, test mining, e-learning, and social network mining. He has received the title of Youth 1000 Talent from the National Science Foundation of China. He is the Editor-in-Chief of the Data Mining and Pattern Recognition Journal.

UNIL YUN received the M.S. degree in computer science and engineering from Korea University, Seoul, South Korea, in 1997, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2005. He was a Postdoctoral Associate for almost one year with the Computer Science Department, Texas A&M University. He was with the Multimedia Laboratory, Korea Telecom, from 1997 to 2002. He was a Senior Researcher with the Electronics and Telecommunications Research Institute. In March 2007, he joined the School of Electrical and Computer Engineering, Chungbuk National University, South Korea. Since August 2013, he has been with the Department of Computer Engineering, Sejong University, Seoul. His research interests include data mining, information retrieval, database systems, artificial intelligence, and digital libraries.

* * *