OVERVIEW

1. The necessities

High-Utility Itemset Mining (HUIM) is a data mining task that aims to discover itemsets that have high utility in a transaction database. The utility of an itemset is typically measured by its total profit or weight. HUIM has been a popular research topic since its introduction in 2004, and there have been many different approaches proposed for mining high-utility itemsets.

Most of the existing HUIM algorithms are sequential or parallelized in a limited way. They also make some unrealistic assumptions in order to reduce their complexity. As a result, they may not be able to fully utilize the capabilities of modern multi-core processors.

In recent years, there has been growing interest in developing more efficient and scalable HUIM algorithms that can take advantage of the parallelism available in modern hardware. Some promising approaches have been proposed, but there is still much room for improvement.

In this dissertation, the author focuses on researching and proposing methods to improve the mining time of high-utility itemsets by flexibly applying parallel processing models to further improve the efficiency of parallel processing on multi-core CPUs.

2. Research objectives

The dissertation focuses on researching and finding suitable data structures and strategies to effectively utilize more the processing power of multi-core CPUs to improve the mining time of high-utility itemsets. The research objectives of the dissertation can be summarized into the following key points:

- Proposing strategies and techniques to improve the efficiency of high-utility itemset mining while preserving the amount of knowledge extracted.
- Applying parallel processing models to improve the efficiency of high-utility itemset mining.
- Extending and scaling parallelization strategies to further utilize the multi-core CPUs.

3. Research scopes

The dissertation focuses on studying the following:

- **Traditional high-utility itemset mining algorithms:** This includes algorithms like Two-Phase, FHM, EFIM, IMEFIM, along with their associated efficient strategies.
- **Compact representation-based high-utility itemset mining algorithms:** This includes algorithms like CHUI-Miner, CLS-Miner, EFIM-Closed, and related techniques.
- **Hierarchical high-utility itemset mining algorithms:** This includes algorithms like MLHUI-Miner and CLH-Miner.
- **Parallel high-utility itemset mining algorithms:** This includes algorithms like pEFIM and MCH-Miner.
- **Static utility databases:** Databases where the utility of items remains constant.
- **Dynamic utility databases:** Databases where the utility of items can change over time.
- **Hierarchical databases:** Databases where items are organized in a hierarchical structure.
- **Standard and publicly available datasets:** These are commonly used datasets in scientific literatures.

4. Contributions of the dissertation

This dissertation focuses on developing new approaches to improve the efficiency of high-utility itemset mining algorithms.

The proposed approaches concentrate on the following topics:

- **Efficient mining techniques:** These techniques aim to efficiently discover high-utility itemsets using compact and lossless representations to preserve the discovered insights.
- **Parallel mining models:** These models utilize multi-core processing architectures to reduce the execution time of high-utility itemset mining algorithms.

5. Research novelty

The results of this dissertation can be applied to the following problems:

- **Reducing the size of the result set:** This involves minimizing the number of results generated by association rule mining algorithms, particularly on large databases and hierarchical databases, while maintaining accuracy. The goal is to identify relevant items that have high utility in various applications, such as retail stores, supermarkets, and more.
- **Improving mining efficiency:** This involves enhancing existing association rule mining methods to leverage the power of multi-core processors, which are widely used today. The aim is to improve the efficiency of the algorithm while keeping costs low.

6. Structure of the dissertation

The dissertation is structured into five chapters. In which, the main contributions of the dissertation are presented in **Chapter 2**, **Chapter 3** and **Chapter 4**.

CHAPTER 1. PRELIMINARIES

1.1 History of the high-utility mining task

The high-utility itemset mining (HUIM) problem originates from the frequent itemset mining (FIM) problem. The purpose of FIM is to identify frequently occurring item combinations within a transaction database. The popularity of itemsets is determined based on a minimum popularity threshold ($minsup$) specified by the miner. The results of the problem are knowledge in the form of association rules, which support decision making. Because FIM only considers the occurrence of items in transactions and ignores important information about the profit of items, most of the itemsets found do not bring high profit value.

HUIM overcomes this problem by considering additional information for each item. Each item in the transaction will be supplemented with the quantity and unit price of the corresponding item. To generalize, this information is respectively called internal utility and external utility. The product of these two pieces of information indicates the profit (utility) obtained when selling the corresponding item. When an item or itemset has a utility that satisfies a given minimum utility threshold ($minutil$), they are called high-utility itemsets.

However, considering more information brings many difficulties and challenges to the HUIM problem. The main reason is that the utility measure does not satisfy the downward closure property (DCP). This leads to the computational cost and search space of the problem being significantly larger than FIM..

Definition 2.1 (The task of high-utility itemset mining).

Let $\mathbb{D}$ be a transaction database and a minimum utility threshold $minutil$, returns the complete set of all high-utility itemset from the given database.

- Input: transaction database $\mathbb{D}$, $minutil$ threshold.
- Output: the set $\mathbb{H}$ containing the full set of high-utility itemsets from $\mathbb{D}$.

Definition 2.2 (Utility of an item in transaction).

Let $i \in \mathbb{I}$ be an item appearing in transaction T_c, the utility of i in T_c is determined as $u(i, T_c) = iu(i, T_c) \times eu(i)$.

1.2 Related works

Some initial approaches to solve the HUIM problem include Two-Phase, UP-Growth, TWU-Mining, HUI-Miner, FHM, EFIM, HMiner, iMEFIM, etc. However, the pattern mining problem often returns a large number of itemsets and requires additional time to analyze these results. Methods for reducing the size of the result set have also been introduced, such as: CHUI-Miner, CLS-Miner, EFIM-Closed, HMiner-Closed, etc.

The above HUIM approaches only focus on single items and their combinations. These approaches only meet a part of the problem in practice. Typically, items are classified into groups of different types. This classification makes management easier and more convenient.

In order to utilize this information in the transaction database, the HUIM problem has been extended since 2017. The problem is known as generalized high-utility itemset mining (GHUIM). Some typical methods include: MLHUI-Miner, CLH-Miner, TKC, MLHMiner, FEACP. The GHUIM problem is a highly complex problem because the search space is significantly expanded and there are not many effective methods to solve it.

In order to reduce the mining time of the HUIM problem, a parallel processing model based on multi-core CPU is also considered. The effectiveness of applying the parallel processing model to the HUIM problem depends on many factors such as: partitioning of the search space, scheduling strategy of parallel tasks, database size, etc. Some typical parallel algorithms include: pEFIM, MCH-Miner, pCLH-Miner. These parallel algorithms all have the common feature that the parallelization model and scheduling strategy are still simple and have not fully utilized the computing power of multi-core CPUs.

Distributed models have also been applied to the HUIM problem to take advantage of the capabilities of multiple computers. Some methods include: DTWU-Mining, PHUI-Miner, P-FHM+, etc. However, the common point of these methods is that the mining process still takes place sequentially on the computers.

1.3 Opening problems in adopting parallel computing into HUIM.

In the process of research and survey, the researcher found that the application of the parallel processing model to the HUIM problem faces the following challenges, leading to the problem still having no solution or the solution is still not thorough:

- The cost of scanning the database is still high, especially for the HUIM mining problem with closed representation on large, low-density databases with a large number of items. In that case, the join and transaction projection techniques will no longer be effective.
- The parallelization model is only applied to the mining phase through static search space partitioning, applying a simple scheduling strategy. Therefore, it does not fully utilize the processing power of multi-core hardware.
- The GHUIM problem has a very large search space and there is still no efficient parallel method to speed up the mining process.

CHAPTER 2. EFFICIENTLY DISCOVER HIGH-UTILITY ITEMSET IN DATABASES WITH DYNAMIC UTILITIES

2.1 Overview of the problem

The HUIM problem has a large search space and high computational complexity. In addition, the results returned from the mining algorithms are also very large in number of itemsets. Besides, previous high-utility itemset mining algorithms all assume that the utility of items is static. This creates results that are inaccurate compared to reality and are not applicable when applied to transaction databases with changing utility. These problems have all been solved, however, only from independent perspectives and have not been combined to become a complete solution.

The problem is how to solve the problem of mining closed high-utility itemsets with dynamic utility databases in a thorough way. From there, it can be extended to apply optimization and parallel processing strategies to speed up the mining process of the problem..

2.2 Dynamic utility model

To apply the high-utility itemset mining approaches to the databases containing dynamic utility values, the core utility computation provided in Definition 2.2 must be completely revised. The following definition provide a more precise way to reflect the dynamic property of items in real-world databases.

Definition 3.1 (Utility of an item in transaction)

The utility value of an item $i \in \mathbb{I}$ in transaction T_c, denoted as $u(i, T_c)$, is now computed as follows: $u(i, T_c) = iu(i, T_c) \times eu(i, T_c)$. Whereas $eu(i, T_c)$ is the external utility of i in transaction T_c.

2.3 The iEFIM-Closed algorithm

The proposed algorithm namely iEFIM-Closed (Algorithm 2.1), aims to efficiently address the aforementioned mining task to extract high-utility itemset from dynamic utility databases. The characteristics of the algorithm are as follows.

- A single-phase approach, extending from the EFIM-Closed algorithm.
- Adopting the dynamic utility model from the iMEFIM algorithm.

- Significantly reducing the cost of database scans by employing the P-Set structure, which was previously introduced the iMEFIM algorithm.

Input: itemset α, projected database $\alpha\mathbb{D}$, $Primary(\alpha)$, $Secondary(\alpha)$, $minutil$ threshold, $P\text{-}set(\alpha, i)$.
Output: tập các tập mục hữu ích cao là mở rộng của α.

FOREACH $i \in Primary(\alpha)$ **DO**
  $\beta = \alpha \cup \{i\}$;
  Scan $\alpha\mathbb{D}$ using $P\text{-}set(\alpha, i)$ to calculate $u(\beta)$ $\beta\mathbb{D}$; // HTM.
  IF β has no backward extensions **THEN**
    Calculate $sup(\beta, z), su(\beta, z), lu(\beta, z), P\text{-}set(\beta, z)$ for each $z \in Secondary(\alpha)$ bys scanning $\beta\mathbb{D}$.
    IF $sup(\beta) = sup(\beta \cup \{z\}), \forall z \succ i \wedge z \in E(\alpha)$ **THEN**
      RETURN $\beta \cup \bigcup_{z \succ i \wedge z \in E(\alpha)}\{z\}$ nếu là HUI; // CJU
    ELSE
      $Primary(\beta) = \{z \in Secondary(\alpha) | su(\beta, z) \geq minutil\}$;
      $Secondary(\beta) = \{z \in Secondary(\alpha) | lu(\beta, z) \geq minutil\}$;
      $Search(\beta,\ \beta\mathbb{D},\ Primary(\beta),\ Secondary(\beta),\ minutil,\ P\text{-}set(\beta, z))$;
      IF β has no forward extension **AND** $u(\beta) \geq minutil$ **THEN**
        RETURN β;
    END ELSE
  END IF
END FOR

Algorithm 2.1. Pseudo-code of the iEFIM-Closed algorithm

2.4 Evaluation studies

The iEFIM-Closed algorithm was tested on a number of transaction databases to evaluate its performance. However, in order to properly assess its performance, all items in these database have their external utility values randomly generated within the range $[1, 10]$ using the standard distribution

The performance of iEFIM-Closed is evaluated against two same-class algorithms: EFIM-Closed and CLS-Miner. These algorithms were modified by adopting the Definition 3.1 to properly work on dynamic utility databases.

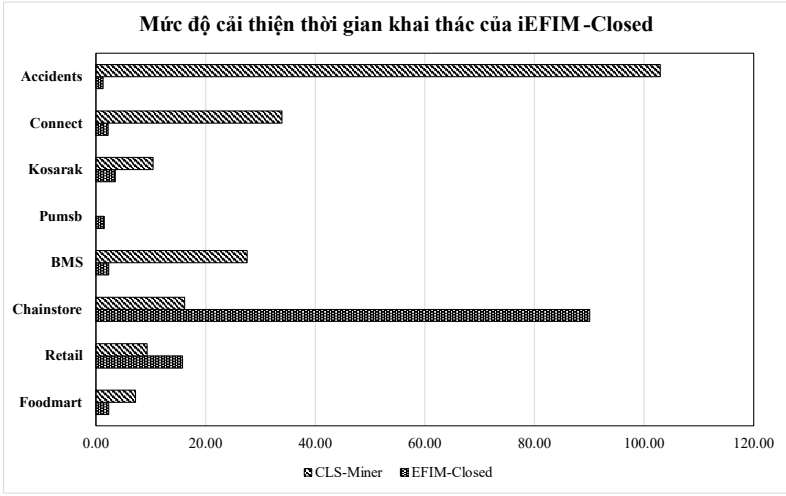


Figure 2.1. Speed-up factor of iEFIM-Closed against baseline approaches

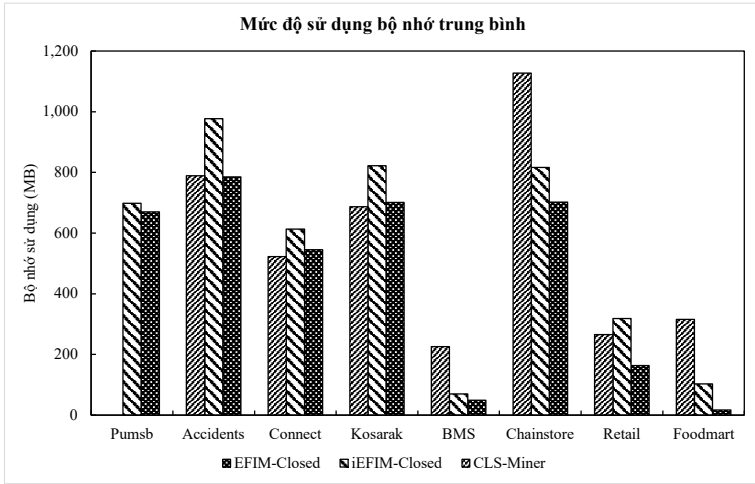


Figure 2.2. Comparison of average memory usages

Figure 2.1 presents the speed-up factor in term of mining time of iEFIM-Closed compared to that of EFIM-Closed and CLS-Miner. It can be seen that iEFIM-Closed has achieved superior performance improvement over both EFIM-Closed and CLS-Miner. The boost in mining time of iEFIM-Closed is over 90 times compared to EFIM-Closed

on the Chainstore database, and over 100 times faster than CLS-Miner of the Accidents database.

Table 2.1. Number of transactions scanned by both iEFIM-Closed and EFIM-Closed

Database	iEFIM-Closed	EFIM-Closed	Reduced	Scanned
Chainstore	10,021,074	6,141,636,120	**99.8%**	0.2%
Retail	861,798	134,978,290	**99.4%**	0.6%
Kosarak	8,101,676	872,102,126	**99.1%**	0.9%
Foodmart	75,484	6,716,739	**98.9%**	1.1%
BMS	175,270	10,002,024	**98.2%**	1.8%
Accidents	12,357,609	22,724,687	**45.6%**	54.4%
Connect	1,348,154	1,732,230	**22.2%**	77.8%
Pumsb	18,679,150	22,393,751	**16.6%**	83.4%

The average memory usage of the three algorithms is shown in Figure *2.2*. iEFIM-Closed always shows a higher average memory usage than EFIM-Closed to store the P-Set structure, however the difference is negligible compared to the performance that the algorithm brings.

Table *2.1* shows the database scan cost of both iEFIM-Closed and EFIM-Closed, which is determined based on the total number of transactions scanned during the entire mining process. In all databases, the number of transactions scanned by iEFIM-Closed is always lower than that of EFIM-Closed, and the improvement is up to 99.4%.

2.5 Conclusions

This first contribution proposes the iEFIM-Closed algorithm to serve the problem of mining closed high-utility itemsets and is capable of:

- Effectively mining all closed high-utility itemsets in a transaction database.
- Applicable to databases with changing profits.
- Applying a new technique to improve the performance of the algorithm, minimizing the cost of scanning the database multiple times when applied to real databases.

The algorithm has good mining performance on both dense and sparse database formats. The significant improvement in mining performance is shown on sparse databases.

The experiments showed that iEFIM-Closed solved the first research question of the dissertation. However, the mechanism for checking the closure of itemsets inherited from EFIM-Closed all impose data dependencies. Therefore, it is not possible to implement parallelization on iEFIM-Closed.

Therefore, the following development directions for the algorithm can be considered:

- Extending the algorithm to work on databases with item hierarchies, to serve the GHUIM problem.
- Studying the application of parallelization mechanisms at each level of a transaction database with item hierarchies.

CHAPTER 3. PARALLEL MODEL TO EXTRACT MULTI-LEVEL HIGH-UTILITY ITEMSETS

3.1 Overview of the problem

The problem of mining high-utility itemsets on hierarchical databases was proposed in 2017 and is an extension of the HUIM problem. This problem considers the hierarchy of items in the database through a taxonomy structure. Figure *3.1* illustrates a taxonomy, in which items a, b are grouped into a group called Y; similarly, c, d, e are grouped into group X. X, Y are called generalized items and a, b, c, d, e, f are called specialized items. Through the taxonomy, it is possible to see the emergence of knowledge that the traditional HUIM problem could not previously discover.

So far, no algorithm has been proposed to solve the problem of mining closed high-utility itemsets on hierarchical databases to overcome the problem of the number of result itemsets.

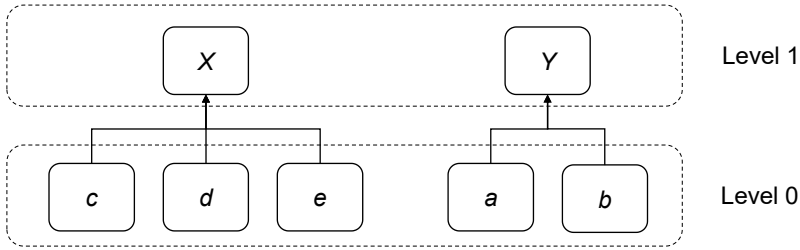


Figure 3.1. An example of a taxonomy structure of items in a database

3.2 Parallel Computing Model

To take advantage of the power of multi-core CPUs, algorithms need to be implemented in parallel. There are many methods to achieve parallelization, such as parallelization through microprocessor instruction sets (SIMD, MIMD), parallelization through multi-threaded processing mechanisms, etc. A wide range of applications have been implemented in parallel, taking advantage of the multi-threaded processing model to increase efficiency. In the field of data mining, many mining approaches have also been extended to parallelization.

For the problem of mining high-utility itemsets, some parallelization approaches have

also been proposed. However, they only stop at simple parallelization. For the problem of mining high-utility itemsets on hierarchical databases, there has been no proposed parallelization approach to reduce mining time. Thus, during the research process, the following properties are observed:

- High-utility itemset mining on hierarchical databases is mainly based on the taxonomy of the database.
- For the problem of mining multi-level high-utility itemsets, the discovered itemsets are strictly limited at their respective level on the taxonomy.
- The larger the size of the taxonomy (depth and width of each level), the mining time will increase significantly.

- **Input:** the database $\mathbb{D}$, taxonomy τ, $minutil$ threshold
- **Output:** set of all the discovered multi-level high utility itemsets $\mathcal{R}$.

1 $\mathcal{R} \leftarrow \emptyset$; $EUCS \leftarrow \emptyset$;
2 Scan $\mathbb{D}$ and τ to construct $\mathbb{I}$ and $\mathbb{G}$, compute TWU for all $i \in \mathbb{I}$.
3 Scan $\mathbb{D}$ and τ to compute TWU for all generalized items $g \in \mathbb{G}$.
4 Construct the set $\mathbb{I}^+ \leftarrow \{i \in \mathbb{I} \mid TWU(i) \geq minutil\}$.
5 Construct the set $\mathbb{G}^+ \leftarrow \{g \in \mathbb{G} \mid TWU(g) \geq minutil\}$.
6 Construct $EUCS$ for all items in $\mathbb{I}^+ \cup \mathbb{G}^+$
7 **FOR EACH** level $l \in \tau$ **DO PARALLEL**
8 $\mathbb{IG}^+ \leftarrow$ promising items from $\mathbb{I}^+ \cup \mathbb{G}^+$ at level l.
9 $\mathcal{R}_l \leftarrow$**FHM_Search** $(\emptyset, \mathbb{IG}^+, \emptyset, minutil, EUCS)$;
10 **END**
11 **RETURN** $\mathcal{R} \leftarrow \cup_{l \in \tau} \mathcal{R}_l$.

Algorithm 3.1. Pseudo-code of MCML-Miner algorithm

3.3 The algorithm MCML-Miner

The first algorithm proposed to solve this problem is the MCML-Miner algorithm (Algorithm 3.1), which is also the first algorithm to implement a parallel model for the GHUIM mining problem. The parallelization model chosen is the multi-core processing model. MCML-Miner is extended from the MLHUI-Miner algorithm. The main mining process of MLHUI-Miner on each level of the taxonomy is implemented in parallel. The deeper the taxonomy and the more cores the processor has, the more efficient the mining process will be.

Characteristics of the MCML-Miner algorithm are as follows.

- It is a single-phase algorithm extended from MLHUI-Miner.
- It parallelizes the mining process at each level of the taxonomy.

3.4 The algorithm MLC-Miner

To reduce the size while preserving the full insights of the result set from the mining process, closed representation is considered for application in the mining process. The MLC-Miner algorithm is proposed to solve this problem effectively. This provides the basis for applying the parallelization model to the problem.

Input:	the database $\mathbb{D}$, taxonomy τ, $minutil$ threshold, useEUCP flag
Output:	set of all discovered multi-level closed high utility itemsets
1	$\mathcal{R} \leftarrow \emptyset$; $EUCS \leftarrow \emptyset$;
2	Scan $\mathbb{D}$ and τ to construct $\mathbb{I}$ and $\mathbb{G}$, compute TWU for all $i \in \mathbb{I}$.
3	Compute TWU for all generalized items $g \in \mathbb{G}$.
4	Construct the set $\mathbb{I}^+ \leftarrow \{i \in \mathbb{I} \mid TWU(i) \geq minutil\}$.
5	Construct the set $\mathbb{G}^+ \leftarrow \{g \in \mathbb{G} \mid TWU(g) \geq minutil\}$.
6	Scan $\mathbb{D}$ to construct EU-List, $EUCS$ for all items in $\mathbb{I}^+ \cup \mathbb{G}^+$ if $useEUCP =$ **TRUE**.
7	**FOR EACH** level $l \in \tau$ **DO**
8	$\mathbb{IG}^+ \leftarrow$ các hạng mục tiềm năng thuộc $\mathbb{I}^+ \cup \mathbb{G}^+$ tại cấp l.
9	$\mathcal{R}_l \leftarrow$**Explore**$(\emptyset, \mathbb{IG}^+, \emptyset, minutil, EUCS)$;
10	**END**
11	**RETURN** $\mathcal{R} \leftarrow \cup_{l \in \tau} \mathcal{R}_l$.

Algorithm 3.2. Pseudo-code of the MLC-Miner algorithm

Throughout the process of mining multi-level closed high-utility itemsets, each time a candidate itemset is generated, the algorithm needs to check the closure of this itemset. The CHUI-Miner algorithm proposes a simple closure check by checking the closure of the itemsets. However, due to the high complexity of the check, the mining time will be significantly increased, and the performance of the whole algorithm will be reduced. Therefore, an optimization technique called Rapid Utility Closure Checker (RUCK) is proposed. This is the main contribution of the whole algorithm, which helps to improve the mining efficiency significantly. The RUCK strategy is proposed as follows:

- Convert the utility-lists from list format to hash map format.

- The elements of the hash table are a tuple $\langle TID,\ item \rangle$ in the utility-list.
- The TID field is not stored because it is already in the hash table: this reduces the memory cost.
- The closure check is performed by checking the $TidSet$ of the itemsets, which are actually the keys set of the hash map.

This can be seen as a combination of the advantages of both data structures, utility-list and P-Set. The MLC-Miner (Multi-Level Closed HUI Miner) algorithm is proposed to mine closed high-utility itemsets from hierarchical databases, with the following characteristics (Algorithm 3.2).

- Single-phase algorithm, inherited from CHUI-Miner.
- Applies the EUCP technique to reduce the search space.
- Applies the Rapid Utility Closure Checker (RUCK) strategy.

Input: the database $\mathbb{D}$, taxonomy τ, $minutil$ threshold
Output: set of all discovered multi-level closed high utility itemsets $\mathcal{R}$

1 $\mathcal{R} \leftarrow \emptyset$; $EUCS \leftarrow \emptyset$;
2 Scan $\mathbb{D}$ and τ to construct the $\mathbb{I}$ and $\mathbb{G}$, compute TWU for all $i \in \mathbb{I}$.
3 Compute TWU for all generalized items $g \in \mathbb{G}$.
4 Construct the set $\mathbb{I}^+ \leftarrow \{i \in \mathbb{I} \mid TWU(i) \geq minutil\}$.
5 Construct the set $\mathbb{G}^+ \leftarrow \{g \in \mathbb{G} \mid TWU(g) \geq minutil\}$.
6 Scan $\mathbb{D}$ to construct EU-List, $EUCS$ for all $i \in \mathbb{I}^+ \cup \mathbb{G}^+$.
7 **FOR EACH** level $l \in \tau$ **DO PARALLEL**
8 $\mathbb{IG}^+ \leftarrow$ promising items at $\mathbb{I}^+ \cup \mathbb{G}^+$ level l.
9 $\mathcal{R}_l \leftarrow$**Explore**$(\emptyset, \mathbb{IG}^+, \emptyset, minutil, EUCS)$;
10 **END**
11 **RETURN** $\mathcal{R} \leftarrow \cup_{l \in \tau}\ \mathcal{R}_l$.

Algorithm 3.3. Pseudo-code of the PMLC-Miner algorithm

3.5 The algorithm PMLC-Miner

MLC-Miner has effectively solved the problem of mining closed high-utility itemsets from hierarchical databases. However, MLC-Miner is still a sequential algorithm. Therefore, a parallel version called PMLC-Miner will be implemented to improve the mining time of the algorithm, using the same parallel model mentioned above (Algorithm 3.3). The PMLC-Miner algorithm has the following characteristics.

- It is a single-phase algorithm extended from the MLC-Miner algorithm..
- PMLC-Miner inherits all the characteristics of MLC-Miner.
- It shortens the mining time through the multi-core parallelization model.

3.6 Evaluation studies

The three proposed algorithms are compared with their corresponding baseline versions based on: mining time and memory usage. MCML-Miner will be compared with MLHUI-Miner. MLC-Miner and PMLC-Miner will be compared with CHUI-Miner* (extended version of the CHUI-Miner algorithm to work with hierarchical databases).

3.6.1 The algorithm MCML-Miner

Figure *3.2* shows the effectiveness of the multi-core parallelization model through the mining time of MCML-Miner: the efficiency is improved up to 5.6 times. However, the parallelization strategy depends on the number of levels of the taxonomy and the parallelization process only takes place during mining. Therefore, although the mining performance is improved, MCML-Miner still does not fully utilize the capabilities of multi-core processors.

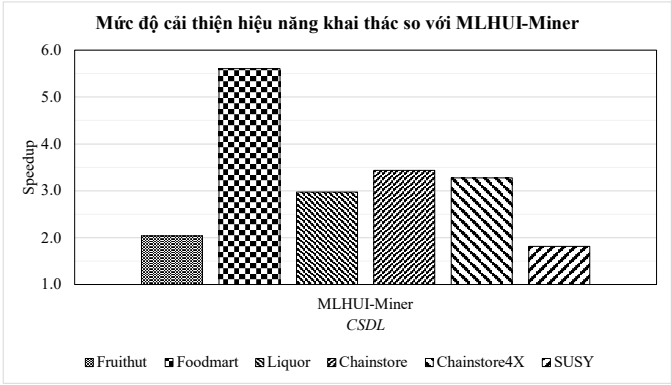


Figure 3.2. Speed-up factor of MCML-Miner against MLHUI-Miner

MCML-Miner also shows that its memory usages are always higher than that of MLHUI-Miner. The more parallel tasks, the more memory are allocated to maintain them, according to Figure *3.3*.

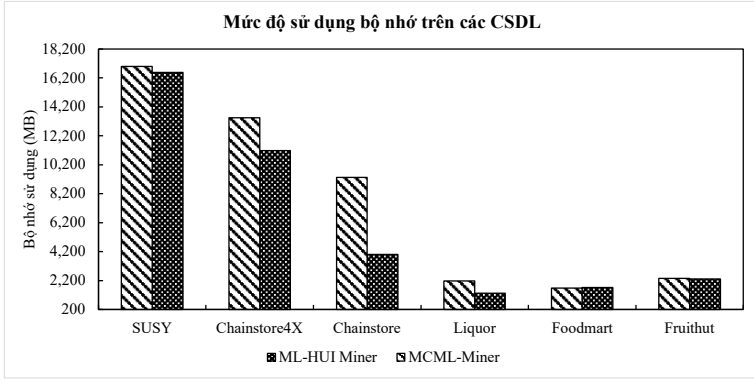


Figure 3.3. Average memory usages of MCML-Miner and MLHUI-Miner

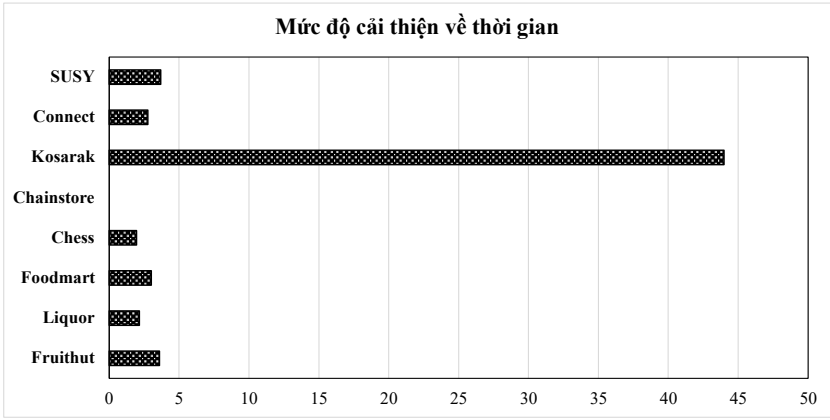


*Figure 3.4. Speed-up factor of MLC-Miner against CHUI-Miner**

3.6.2 The algorithm MLC-Miner

The speed-up factor of MLC-Miner against CHUI-Miner* is depicts on *Figure 3.4*. The time improvement of MLC-Miner is clearly shown on large databases. For the Chainstore database, while MLC-Miner completes in less than 150 seconds, CHUI-Miner* cannot complete at the 7200 second mark..

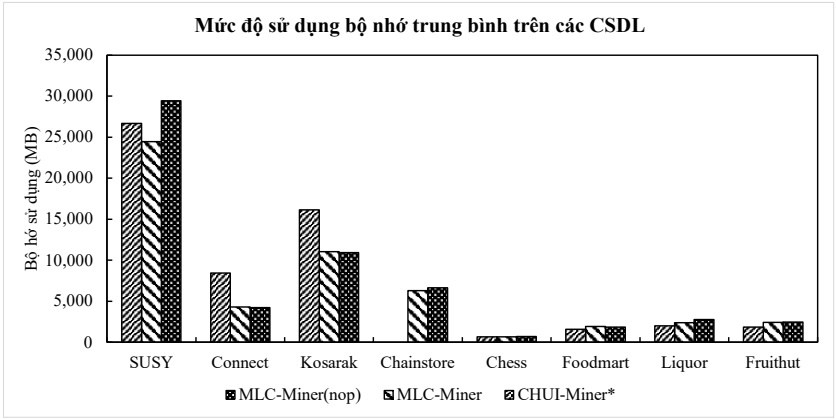


Figure 3.5. Peak memory usage between MLC-Miner and CHUI-Miner*

Figure *3.5* shows the efficiency in memory usage between MLC-Miner and CHUI-Miner. The memory usage of MLC-Miner is significantly lower than that of CHUI-Miner*, even the version without EUCP, MLC-Miner(nop), the amount of memory allocated to the algorithm is also much lower than CHUI-Miner*.

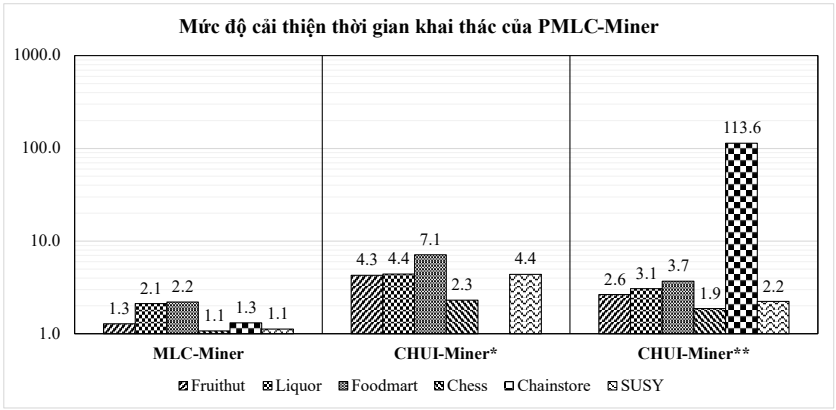


Figure 3.6. Speed up factor of PMLC-Miner against the baselines

3.6.3 The algorithm PMLC-Miner

For the parallelized version of MLC-Miner, PMLC-Miner, the performance improvement is significantly better than CHUI-Miner*, CHUI-Miner** and MLC-Miner (*Figure 3.6*). Specifically, PMLC-Miner, which is the parallelized version of CHUI-

Miner using the same parallel model, achieves a performance improvement of over 110 times on the Chainstore dataset over CHUI-Miner**.

In terms of memory usage, PMLC-Miner performs similarly to MCML-Miner. However, PMLC-Miner consistently requires more memory than the sequential version.

3.7 Conclusions

This research focuses on the problem of parallel mining of high-utility multi-level, multi-level closed itemsets in the hierarchical database format and proposes a series of methods to solve it. The proposed algorithms are named MCML-Miner, MLC-Miner and PMLC-Miner, respectively.

The second research content of the dissertation has been basically completed by proposing a parallelization model as presented. However, it can be seen that the parallel processing of the proposed model only takes place in the search space scanning phase of the algorithm. Besides, this parallelization model also does not fully utilize the idle processing cores of multi-core processors and still depends mainly on the depth of the taxonomy. If this parallelization model is applied to conventional databases, databases that do not contain taxonomy, the parallelization process will be completely disabled.

The next development direction of the dissertation will focus on further improving this parallelization model. From there, better utilize the parallel computing power of multi-core processors to serve the problem of mining high-utility itemsets more effectively

CHAPTER 4. IMPROVING THE EFFECTIVENESS OF THE PARALLEL MODEL TO DISCOVER MULTI-LEVEL HIGH-UTILITY ITEMSETS

4.1 Overview of the problem

The content of the second research contributed to the research field by proposing a parallelization model based on taxonomy to conduct mining. This model is applied to two algorithms, MCML-Miner and PMLC-Miner, to mine multi-level and multi-level closed high-utility itemsets, respectively. However, this model only focuses on parallelizing the mining process and depends on the depth of the taxonomy.

This third research task will focus on extending and further improving the aforementioned parallelization model to fully utilize multi-core processors.

4.2 The algorithm MCML+

The disadvantage of MCML-Miner is that it mines the search space of each level in the taxonomy independently. Typically, with CPUs with 2 to 4 cores, MCML-Miner performs well. If the CPU has more processing cores than the depth of the taxonomy, this strategy will become less effective because it does not utilize the idle CPU cores.

Therefore, a new approach is needed to address the problem of MCML-Miner to better utilize the processing power of CPUs. The parallel processing model can be applied at different stages of the algorithm to improve efficiency. The characteristics of the MCML+ algorithm include:

- Improved single-phase algorithm from MCML-Miner algorithm.
- Applying multi-core parallelization strategy at different stages improves the efficiency of the mining process.

4.3 The algorithm MCML++

The MCML+ approach (Algorithm 4.1) achieves the goal of improving mining time, especially on large databases. However, the main mining process of MCML+ is still based on MCML-Miner and is not yet effective. This drawback needs to be addressed

thoroughly: all CPU cores must be used for the problem space scanning process, instead of just using a few cores to scan each level of the taxonomy.

Input: database $\mathbb{D}$, taxonomy τ, $minutil$ threshold
Output: set of all discovered multi-level high utility itemsets $\mathcal{R}$

1 $\mathcal{R} \leftarrow \emptyset$; $EUCS \leftarrow \emptyset$;
2 Scan $\mathbb{D}$ andτ to construct $\mathbb{I}$ and $\mathbb{G}$, compute TWU for all $i \in \mathbb{I}$.
3 Compute TWU for all generalized items $g \in \mathbb{G}$ in parallel.
4 Construct the set in parallel $\mathbb{I}^+ \leftarrow \{i \in \mathbb{I} \mid TWU(i) \geq minutil\}$.
5 Construct the set in parallel $\mathbb{G}^+ \leftarrow \{g \in \mathbb{G} \mid TWU(g) \geq minutil\}$.
6 Scan $\mathbb{D}$ to construct in parallel the structures EU-List, $EUCS$ cho for all items in $\mathbb{I}^+ \cup \mathbb{G}^+$ using taxonomy τ.
7 Reordering the promising items using the order of SJF strategy.
8 Enqueue all items in $\mathbb{I}^+ \cup \mathbb{G}^+$ into a priority queue using the order of SJF.
9 **FOR EACH** $i \in \mathbb{I}^+ \cup \mathbb{G}^+$ **DO**$\mathbb{I}^+ \cup \mathbb{G}^+$ **PARALLEL**
10 $\mathbb{IG}_i \leftarrow$ promising items that can extend i.
11 $\mathcal{R} \leftarrow \mathcal{R} \cup$ **Explore**$(\emptyset, \mathbb{IG}_i, \emptyset, minutil, EUCS)$;
12 **END**
13 **RETURN** $\mathcal{R}$.

Algorithm 4.1. Pseudo-codeof the MCML++ algorithm

This issue can be solved by parallelizing the search space of each single item, which is similar to the pEFIM and MCH-Miner algorithms. The coordination strategy used in these algorithms is FCFS. Assigning search spaces randomly to CPU cores without considering their characteristics is also not effective. Consider the following heuristic: smaller search spaces should be executed first, followed by larger ones. The coordination strategy that uses this heuristic is called Shortest Jobs First.

Definition 5.1. (Order of processing)

Let $\mathcal{A}$ and $\mathcal{B}$ be two itemsets that need to be extended. The search space of $\mathcal{A}$ is considered smaller than the search space of if $|E(\mathcal{A})| < |E(\mathcal{B})|$. Whereas $E(\mathcal{A})$ and $E(\mathcal{B})$ are the two sets of items that can be used to extend $\mathcal{A}$ and $\mathcal{B}$, respectively. This order of processing can be implemented using a priority queue.

The MCML++ algorithm integrates all the parallelization techniques presented with the following characteristics.

- Single-phase algorithm using a multi-core parallel processing model.

- Eliminates taxonomy depth dependency to maximize multi-core CPU utilization.
- Applies the Shortest Jobs First coordination strategy.

4.4 Evaluation studies

A series of experiments were also implemented to evaluate the effectiveness of MCML+ and MCML++. The MCML-Miner algorithm is denoted as MCML(12). The MCML+ and MCML++ algorithms were tested with , denoted as MCML+(12) and MCML++(12). Figure *4.1* shows that MCML++ always has a better improvement rate than MCML+, MCML-Miner, and MLHUI-Miner.

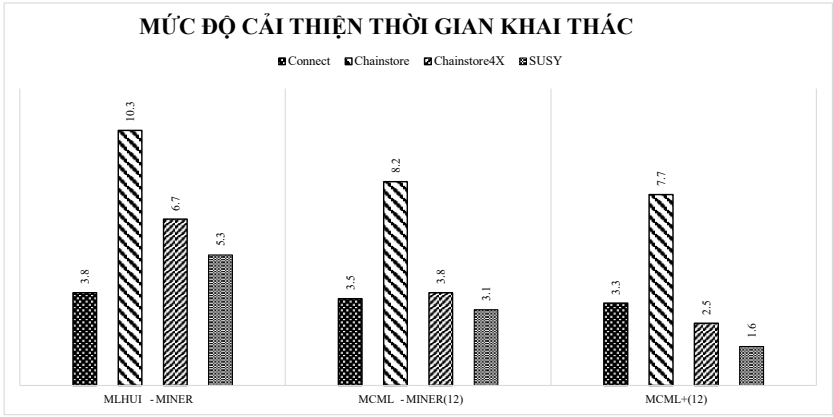


Figure 4.1. Speed-up factor of MCML++ against the baseline algorithms

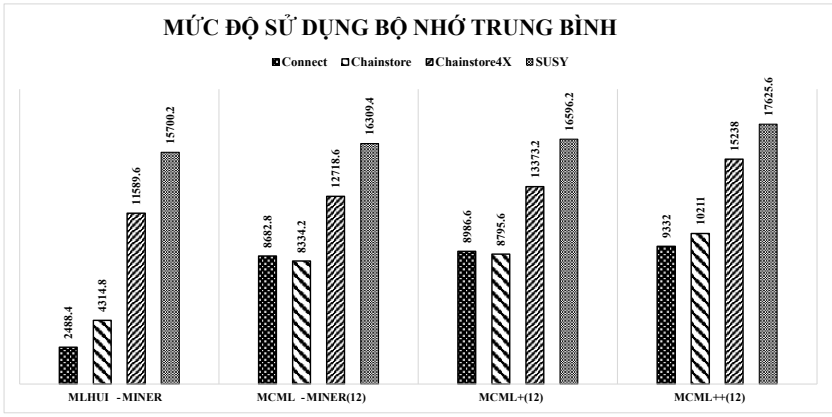


Figure 4.2. Average memory usage among algorithms

Figure *4.2* shows the average memory usage of the algorithms. MCML++ shows higher memory usage than the other algorithms. However, the increase in memory is negligible compared to the improved performance. For example, MCML++ with 12 cores on the SUSY dataset has a memory increase of only ~2.0GB while the performance improvement is more than 5 times that of MLHUI-Miner.

The effectiveness of Shortest Jobs First was also evaluated with CPU cores compared to the reverse Shortest Jobs First and First Come, First Served strategies. The results show that SJF is more effective than the other strategies..

4.5 Conclusions

The third research task studies the problem of parallel mining and high-level high-utility itemsets from hierarchical databases. The MCML+ and MCML++ algorithms are proposed based on MCML-Miner. The parallel processing model is applied to different stages of the algorithm. The Shortest Jobs First coordination strategy is also proposed to control the parallel mining process.

Experiments show that MCML+ and MCML++ are both significantly more efficient than MCML-Miner. The memory usage of MCML+ and MCML++ is higher than that of MCML-Miner, however, this difference is not significant.

Although applying a series of techniques to improve mining efficiency, the proposed methods still have the potential to be extended in the future.

- The partitioning of the search space of the algorithms is unbalanced. In most cases, mining tasks will end at different times. Therefore, approach to distribute the workload more evenly through load balancing strategies should be considered.
- The proposed algorithms only work on a single computing system with limited processing power and memory. Therefore, the future consideration should be to extend the algorithm to work on large databases.

CHAPTER 5. CONCLUSION AND RECOMMENDATIONS

5.1 Conclusions

In the research process, the author has achieved a number of results by proposing a series of algorithms for mining high-utility itemsets, generalized high-utility itemsets, and combining these problems with the multi-core parallel processing model. The main goal is to better utilize the computing power of multi-core processors at low cost. Experimental results show the performance improvement of the proposed methods. In particular, the approaches using the multi-core parallel computing model. The performance improvement of these methods compared to non-parallel approaches is significant.

In terms of methodology, the parallelization models have been proven through mathematical modeling and experimentation to be able to improve mining time by several times compared to methods that do not apply parallel computing models.

5.2 Recommendations

Although the proposed algorithms are very effective, they still have some drawbacks. A common feature of depth-based itemset mining algorithms is that the search space is unbalanced, leading to parallel mining tasks having different execution times. Therefore, more balanced load balancing techniques need to be considered.

In addition, the proposed parallel processing models are limited to a single computing system with limited processing and storage capacity. Therefore, the next development direction is to extend the algorithm to work on extremely large databases through a distributed environment