Knowledge-Based Systems 267 (2023) 110441

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

An efficient method for mining high occupancy itemsets based on equivalence class and early pruning

Loan T.T. Nguyen [a,b], Thang Mai [c], Giao-Huy Pham [a,b], Unil Yun [d], Bay Vo [e,*]

[a] School of Computer Science and Engineering, International University, Ho Chi Minh City, Viet Nam
[b] Vietnam National University, Ho Chi Minh City, Viet Nam
[c] IS Department, Aperia Solutions, Dallas, TX, USA
[d] Department of Computer Engineering, Sejong University, Seoul, Republic of Korea
[e] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Viet Nam

ARTICLE INFO

Article history:
Received 20 February 2022
Received in revised form 28 February 2023
Accepted 1 March 2023
Available online 5 March 2023

Keywords:
High occupancy itemset
Occupancy pattern
Data mining
Equivalence class
Pruning candidates

ABSTRACT

Many researchers have been investigating and applying a new trend of data mining, namely high occupancy itemset mining. Frequent itemset mining often returns a large set of itemsets, but businesses need a smaller set of inputs to investigate or send into a recommendation system to quickly make decisions. Applying an occupancy measure to a support-based mining framework will thus bring many benefits for decision support systems, while managers will benefit by having a new method to visualize reports and analyze data more efficiently. Similar to frequent itemset mining, mining high occupancy itemsets can be applied on any transaction database. In this research, we apply additional conditions to eliminate unqualified itemsets and integrate the property of equivalence class to reduce the runtime of the *k-itemsets* generation process. Moreover, a new theorem is stated and applied to a specific class of databases so that it is not necessary to calculate the upper-bound occupancy, and this speeds up the process as well as reduces memory requirements with regard to generating high occupancy itemsets. We develop two new algorithms, fast high occupancy itemset mining (FHOI) and depth first search (DFS) for high occupancy itemset mining (DFHOI) to solve the problem. Our new algorithms are examined experimentally using different databases to evaluate its performance in term of runtime and memory usage.

1. Introduction

Data mining is now widely used in a variety of fields, such as the national economy [1,2], business [3], engineering [4,5], medicine [6–8], etc. Frequent itemset mining is one of the most common research topics in data mining [9]. The task of mining frequent itemsets is very important in association rule mining [9–15], and thus many algorithms have been developed for solving frequent itemset mining problems. Apriori [9] was one of the earliest methods for mining frequent itemsets (FIs), with its methodology being to scan the database in a brute force way to carry out the task. The FP-Growth [16] method was then created to address some of the Apriori algorithm's shortcomings, such as scanning the database numerous times to find useful itemsets. The authors of FP-Growth also proposed the frequent pattern tree (FP-tree) concept for storing and compressing databases in local memory. The algorithm can accomplish the mining operation with only two database-scans thanks to this technique. Furthermore, the FP-tree structure aids in the reduction of computational costs and improves the performance of mining considerably. However, most algorithms use a support-based mining approach.

Recent frequent itemset mining studies have focused on when the frequency of each itemset is greater than a certain threshold. In 2012, Tang et al. [17] also researched mining FIs, analyzed some real-application requirements and then proposed a new measure called average occupancy (the average percentage of an itemset in the transactions that hold the itemset). Based on the support and average occupancy, a qualified itemset must follow this statement: let a minimum support threshold be α and a minimum occupancy threshold β, then if the support of itemset A is equal or greater than α and the average occupancy is equal or greater than β, it means that itemset A is qualified. However, it is difficult to find α and β, because there are a lot of qualifying itemsets. A new measure was thus proposed to solve this problem: (i) if the recommended measure of the itemset is considerable, the itemset should have surprisingly high

* Corresponding author.
E-mail addresses: nttloan@hcmiu.edu.vn (L.T.T. Nguyen), mhthang.it@gmail.com (T. Mai), phamghuy98@gmail.com (G.-H. Pham), yunei@sejong.ac.kr (U. Yun), vd.bay@hutech.edu.vn (B. Vo).

https://doi.org/10.1016/j.knosys.2023.110441
0950-7051/

occupancy and support; (ii) the suggested measure requires only a threshold.

Users may face difficulty in how to determine a pair of minimum support thresholds α and a minimum occupancy threshold β when they apply the approach proposed by Tang et al. [17], Deng proposed the HEP algorithm [18] using an occupancy measure to develop an approach for mining high occupancy itemsets.

1.1. Motivation

Deng proposed the HEP algorithm to mine high occupancy itemsets [18], and compared the new approach with the baseline algorithm of mining FIs with post-processing. This research also reported that the set of high occupancy itemsets is much smaller than the set of frequent itemsets. However, the HEP algorithm must perform a large number of candidates generation during the mining process. Moreover, while the upper bound was proposed to early prune candidates, this approach is not effective in pruning candidates in cases when all transactions in a database have the same length, and thus computing the upper bound will consume time.

1.2. Contributions

The primary contribution of this research is to propose efficient candidate elimination algorithms to mine high occupancy itemsets quickly and efficiently by applying two strategies to prune candidates and improve the search space.

- We index transaction length to save memory for storing the occupancy-list (OL).
- When $k \geq 2$, based on the OL generated after getting all candidates C_k, if the length of OL (contains k-itemsets) is not greater than or equal to a user-defined minimum occupancy threshold ξ, these itemsets will be eliminated before computing the upper-bound occupancy (UBO).
- Apply the equivalence class's property to speed up the candidate generation process.
- State and prove a proposition that the upper bound is redundant in cases when all transactions in a database have the same length. Computing the upper bound in this case will take time, and it has no effect on pruning candidates.
- We apply all the above strategies and techniques in the FHOI algorithm, then examine the performance and develop the DFHOI algorithm to prove that both the HEP and FHOI algorithms still consume a large amount of memory when mining HOIs from steps $k \geq 2$, especially with dense databases where items may appear in most of the transactions and each transaction has same number of items. The DFHOI algorithm utilizes the DFS technique to recursively mine HOIs efficiently, and also quickly release unnecessary memory.

The rest of this paper is organized as follows: In Section 2, we present some current works related to frequent itemset mining and high occupancy itemset mining. In Section 3, we introduce some basic definitions and the problem statement. The proposed FHOI and DFHOI algorithms for mining all high occupancy itemsets are developed in Section 4. A proposition in the case of all transactions in a database has the same length is developed in this section. By using this proposition, we do not need to compute the upper bound, and thus computing time is saved. The experimental results of the proposed algorithm and the existing algorithm (HEP) regarding both runtime and memory usage are also discussed in Section 5. Finally, the conclusions and some directions for future work are shown in Section 6.

2. Related work

2.1. Itemset mining

Itemset mining and its variant problems have attracted multiple researchers [16,19–29]. Frequent itemsets [19] are itemsets that appear frequently in a database. Frequent itemset mining is the task of discovering groups of items which are frequently purchased together by customers. Frequent itemset mining can be easily explained by introducing market basket analysis, a well-known typical usage. Market basket analysis aims to find the relationships or itemsets between different products (real or virtual) that are selected and placed in their basket by a particular buyer, and assign support and reliability for comparison. Cross-marketing and customer behavior analysis is at the heart of this.

Apriori [9] is a well-known algorithm for mining FIs and association rules. It accomplishes this by detecting specific items that appear frequently in the transactional database and grouping them into increasingly larger itemsets (since these itemsets frequently appear enough in database). Apriori uses a "bottom-up" strategy, in which FIs are enlarged one by one (known as candidate generation), and then candidate sets are checked based on the database. If there are no successful expansions, the algorithm stops. Apriori uses breadth-first search and a hash-tree structure to efficiently discover all potential candidates, and so generate a *k-itemset* from the *(k-1)-itemset*. It then trims candidates with a few sub-itemsets. Through the downward closure property, the itemset includes all frequent k-length items. Finally, it will scan the database to find FIs among candidates.

After the release of Apriori, several efficient approaches were discussed, and several efficient algorithms were proposed [23,24,30,31]. Han et al. proposed the FP-Growth algorithm [30], a DFS algorithm, for mining frequent itemsets using a new data structure called FP-Tree. The algorithm was found to be much better than Apriori with regard to speed and having no candidate generation process. Another DFS algorithm was proposed by Zaki et al. [31], called Eclat, although the authors used a linked list data structure to organize data. Vo et al. [23] created the N-list and Subsume-based algorithm for mining frequent itemsets (NSFI algorithm), which uses the N-list structure and the subsume concept to optimize execution time and memory usage. Aryabarzana et al. investigated some data structures based on sets of nodes in a prefix tree and proposed the NegFIN algorithm [24] to mine FIs using a new, efficient data structure, NegNodeset, a novel encoding model for nodes in a prefix tree based on a bitmap representation of sets.

Besides frequent itemset mining, many researchers have also focused on investigating its extension problems. Discovering frequent closed itemsets (FCIs) [32–36], maximal frequent itemsets [37,38] and sequential pattern mining [21,36,39] are also key topics which are important for recommendation and decision support systems. Similarly, other key methods such as weighted clickstream patterns [40], class association rule mining [10,17], high utility itemset mining [41–46] and association rule mining [9,11–15,47–51], help to find interesting associations between large itemsets. Such rules demonstrate how frequently an application appears in the transaction. A common example is based on market research. Association rules are advantageous when analyzing databases. In supermarkets, for example, the barcode scanners used when paying are used to collect data on shoppers' purchases. As a result, managers can know whether certain groups of items are bought together and use this data to adjust the layout, cross-sales, and promotions based on the collected information.

2.2. Occupancy itemsets

The occupancy measure [17] is motivated by some real-world model recommendation applications that require that any interesting itemset A is in a certain number of transactions. More precisely, for any support transaction t of itemset A, the number of items in A must be close to the total items in t. Higher occupancy in these itemsets can lead to a higher recall in these itemsets, while itemsets with a greater frequency have higher precision.

Recently, occupancy has been investigated and applied into many studies. Gan et al. [52] investigated frequent pattern mining and found that frequent patterns usually do not contain a large share of the desired patterns. The authors developed a novel method by applying a frequency-weight tree and two data structures called weight-list and frequency-weight table to exploit highly qualified patterns with frequency and weight occupancy, with the results indicating possible highly qualified patterns. Chen et al. [53] studied high utility itemset mining and extended the application of occupancy patterns. The authors developed a new algorithm, High-Utility-Occupancy Pattern Mining in Uncertain databases, to mine the Potential High Utility Occupancy Patterns. Zhang et [54] developed high correlated occupancy mining by applying correlated occupancy into a support-based framework to mine the top-k qualified results based on a combination of occupancy and correlation.

Deng also investigated a support-based framework and applied occupancy into the proposed HEP algorithm to mine high occupancy itemsets [18]. The HEP algorithm introduced a new data structure, called an occupancy-list, to carry the occupancy information of the itemset. This is considered as the very first approach to mine the high occupancy itemsets. Deng developed two algorithms and compared them with each other. The first was based on mining all frequent itemsets and then finding the high occupancy itemsets. The second algorithm (HEP) applied a new data structure and theorems and generated the high occupancy itemsets directly. The HEP algorithm employed two techniques during mining: (i) sorting transactions by length in ascending order and generating the occupancy lists of 1-itemsets; (ii) all items in an itemset are sorted by support in ascending order in order to generate *k-itemsets* based on interesting (*k-1*)-itemsets. Deng opened a new strategy as well as a new trend for managers to analyze the occupancy of itemsets (products) and to have a new support channel for decision making.

Datta et al. also researched and defined a new algorithm, HOIMTO [55], to mine occupancy itemsets. The approach presented another definition of a high occupancy itemset, which is based on two minimum thresholds, minimum support and minimum item occupancy.

Kim et al. also investigated high occupancy itemset mining [56]. The authors studied the real-world databases and reported that the state-of-the-art approach to high occupancy pattern mining cannot work on incremental databases. The authors thus developed the HOMI algorithm (High Occupancy pattern Mining on Incremental databases) to mine high occupancy itemsets from incremental databases.

3. Definitions and problem statement

3.1. Definitions

Definition 1. Given a set of distinct items, $I = \{i_1, i_2, \ldots, i_m\}$, a transaction database contains a set of transactions, denoted as $DB = \{T_1, T_2, \ldots, T_n\}$, in which $T_k (1 \leq k \leq n)$ is a transaction containing set of items in I.

Table 1
A sample transaction database.

T_{id}	Items
T_1	{a, c, d}
T_2	{a, b, d}
T_3	{b, c, d, e}
T_4	{a, d}
T_5	{c, d, e}
T_6	{a, b, c, d, e}

Definition 2. The support of an itemset P ($Sup(P)$) is the number of transactions that contain P.

Definition 3 (*[18]*)**.** The Support Transaction Set is the supporting transaction set of an itemset P, denoted as $STSet(P)$, is defined as the set of transactions which contain P.

Definition 4 (*[9]*)**.** A non-empty set of items is called an itemset. An itemset which contains a list of k items is called a *k-itemset*.

Definition 5 (*[18]*)**.** The occupancy of itemset P ($O(P)$) is defined as:

$$O(P) = \sum_{T \in STSet(P)} \frac{|P|}{|T|}$$

Definition 6 (*[18]*)**.** Given a user-defined minimum occupancy threshold ξ, an itemset P is called a high occupancy itemset (HO) if $O(P) \geq \xi$.

Definition 7 (*[18]*)**.** The Occupancy-List (OL) of itemset P is a list structure with two fields for each item: 'tid', 'tsize', in which the field 'tid' is the ID of transaction T that contains P; and the field 'tsize' is the length of T. Below is an example of the occupancy-list of 1-itemsets from the sample database (Table 1).

Example 1.

$$OL(\{a\}) = \{(T_1, 3), (T_2, 3), (T_4, 2), (T_6, 5)\}$$
$$OL(\{b\}) = \{(T_2, 3), (T_3, 4), (T_6, 5)\}$$
$$OL(\{c\}) = \{(T_1, 3), (T_3, 4), (T_5, 3), (T_6, 5)\}$$
$$OL(\{d\}) = \{(T_1, 3), (T_2, 3), (T_3, 4), (T_4, 2), (T_5, 3), (T_6, 5)\}$$
$$OL(\{e\}) = \{(T_3, 4), (T_5, 3), (T_6, 5)\}$$

After obtaining the list of OLs for 1-itemsets, the list of OLs for 2-itemsets can be built without examining the *DB*. For instance, the $OL(\{e, a\})$ is generated by intersecting $OL(\{e\})$ and $OL(\{a\})$, then $OL(\{e, a\}) = \{(T6, 5)\}$. Below is the list of all OLs for 2-itemsets:

$$OL(\{e, a\}) = \{(T_6, 5)\}$$
$$OL(\{e, b\}) = \{(T_3, 4), (T_6, 5)\}$$
$$OL(\{e, c\}) = \{(T_3, 4), (T_5, 3), (T_6, 5)\}$$
$$OL(\{e, d\}) = \{(T_3, 4), (T_5, 3), (T_6, 5)\}$$

Definition 8 (*[18]*)**.** Upper-Bound Occupancy (UBO)**:** Suppose the transactions with itemset P have u different lengths. Furthermore, the number of transactions with the same length l_x is denoted as $n_x (1 \leq x \leq u)$, and $\sum_{i=x}^{u} n_i \times \frac{l_x}{l_i}$ is denoted by UBO_P^x. The UBO of P ($UBO(P)$), is stated as:

$$UBO(P) = \max_{1 \leq x \leq u} UBO_P^x.$$

Definition 9 (*Equivalence Class [11]*)**.** Let P be a set. An equivalence relation on P is a binary relation $\equiv$ such that for all $X, Y, Z \in P$, the relation is:

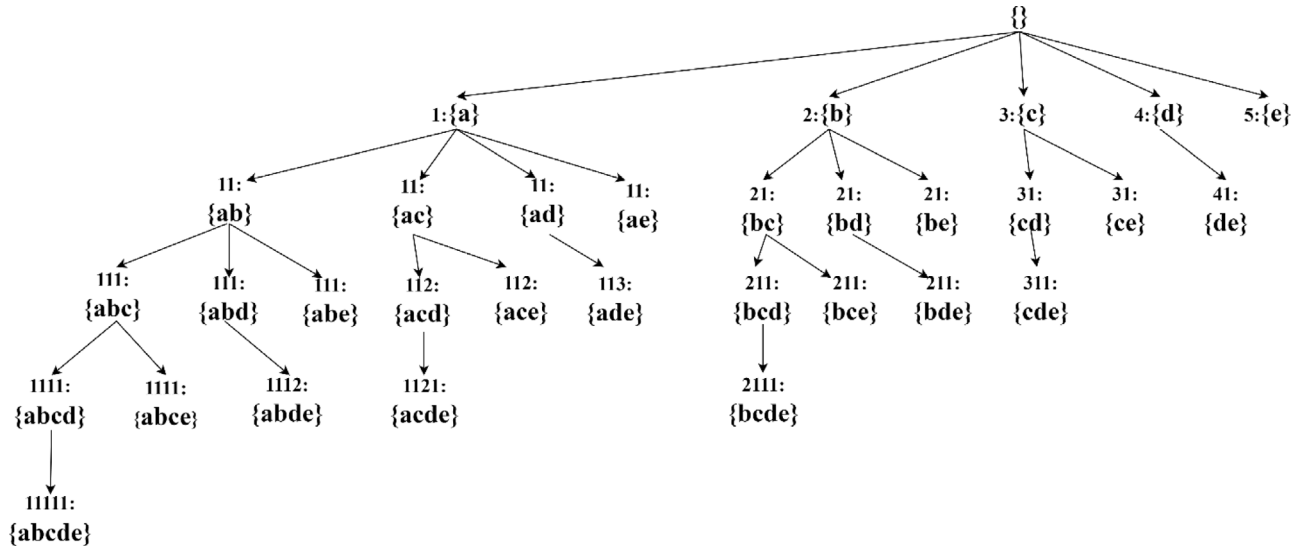

Fig. 1. An example of applying equivalence class using breadth first search.

- Reflexive: $X \equiv X$.
- Symmetric: $X \equiv Y$ implies $Y \equiv X$.
- Transitive: $X \equiv Y$ and $Y \equiv Z$, implies $X \equiv Z$

The equivalence relation partitions the set P into disjoint subsets called equivalence classes. The equivalence class of an element $X \in P$ is given as $[X] = \{Y \in P \mid X \equiv Y\}$.

3.2. Problem statement

Given a transaction database *DB* and a user-specified minimum occupancy threshold ξ, the problem of mining high occupancy itemsets is to find all high occupancy itemsets (HO) whose occupancies are no less than ξ.

$HO = \{P \in DB | O(P) \geq \xi\}$

Theorem 1 (*[18]*)**.** *For any itemset P, we always have O(P)* $\leq$ *Sup(P).*

Theorem 2 (*[18]*)**.** *For itemset P, UBO(P)* $\leq$ *Sup(P) holds.*

4. Proposed method

Definition 10 (*Index of Transaction Length*)**.** Let $DB = \{T_1, T_2, \ldots, T_n\}$ be a transaction database, an array L is defined as the index of length in *DB* as follows

$$\boldsymbol{L_i} = |\boldsymbol{T_i}|$$

Array L is used to save memory usage for storing occupancy-lists. We use the transactions in Table 1 to illustrate the use of this definition.

Example 2 (*Index of Transaction Length*)**.** From Table 1, we have $L = \{3, 3, 4, 2, 3, 5\}$ and now the OL of each item only needs to store the *STSet* of this item, i.e., $OL(P) = STSet(P)$.

Therefore, we have

$STSet(\{a\}) = \{T_1, T_2, T_4, T_6\}$
$STSet(\{b\}) = \{T_2, T_3, T_6\}$
$STSet(\{c\}) = \{T_1, T_3, T_5, T_6\}$
$STSet(\{d\}) = \{T_1, T_2, T_3, T_4, T_5, T_6\}$
$STSet(\{c\}) = \{T_3, T_5, T_6\}$.

Assume that each TID is 4 bytes and each transaction length is 4 bytes, using OL as [18] takes $4 \times 8 + 3 \times 8 + 4 \times 8 + 6 \times 8 + 3 \times 8 = 160$ bytes while using index of length L and *STSet* takes $6 \times 4 + 4 \times 4 + 3 \times 4 + 4 \times 4 + 6 \times 4 + 3 \times 4 = 104$ bytes.

4.1. Algorithms

A new condition can be applied to prune all unpromising *k-itemsets*. If the support of a *k-itemset* is not greater than the user-defined minimum occupancy threshold (ξ), then this itemset will be eliminated before computing UBO to check this candidate. Since UBO is the max function of the one-dimensional array, the computation time will be long. As the number of itemsets that need to calculate UBO decreases, the time needed to calculate UBO will also decrease.

We then investigated equivalence class [11], as our new approach can also speed up the process of high occupancy itemset mining. First, after we obtain all 1-itemsets from the *DB*, we assign an index to each 1-itemset, and these 1-itemsets are also seen as parent nodes in the tree structure. Then we will create all 2-itemsets by combining two 1-itemsets from left to right, and these 2-itemsets will inherit the index of the parent nodes. Two itemsets from *k-itemsets* will be joinable if they have the same parent nodes (or the same equivalence class).

From Fig. 1, we can see that when $k = 2$ this method groups all itemsets that have the same equivalence class into the same group (using a number). To generate candidates with $k > 2$, two itemsets can join to create a new candidate if they have the same group (or the same number), the complexity to check this is $O(1)$. In contrast, the HEP algorithm must traverse these two itemsets to check if they have the same prefix, the complexity is $O(k)$ where k is the length of these two itemsets.

Below is the proposed FHOI algorithm for mining high occupancy itemsets.

Theorem 3. *If all transactions in a database DB have the same length then UBO(P) = Sup(P).*

Proof. l is the length of a transaction and n is the number of transactions in DB, given an itemset P in DB, we have $UBO(P) = \max_{1\le x\le u} UBO_P^x = \max_{1\le x\le u} \sum_{i=x}^{u} n_i \times \frac{l_x}{l_i} = \sum_{i=1}^{u} n_i \times \frac{l}{l} = \sum_{i=1}^{u} n_i = Sup(P)$.

Based on the above ideas, we propose an efficient algorithm for mining HOIs, called FHOI. We use the search tree as in Fig. 1 to generate and prune candidates.

Algorithm 1: Fast High Occupancy Itemset Mining - FHOI

Input: database D and user-defined minimum occupancy threshold ξ

Output: $HOIs$

FHOI()

1. Scan D to generate a set of 1-itemset S, index of length L and tidset of each 1-itemset, determine if D has the same length (*hastheSameLengh*).
2. $HOIs \leftarrow \emptyset$;
3. $C_1 \leftarrow \emptyset$; $HOI_1 \leftarrow \emptyset$;
4. $\{C_1, HOI_1\} \leftarrow$ Mine-HOI-1Itemset(S);
5. $HOIs \leftarrow HOIs \cup HOI_1$;
6. $k = 2$;
7. **While** ($C_{k-1} \neq \emptyset$) **do**
8. $\{C_k, HOI_k\} \leftarrow$ Mine-HOI-KItemset (C_{k-1});
9. $HOIs \leftarrow HOIs \cup HOI_k$;
10. $k++$;
11. Return $HOIs$;

Mine-HOI-1Itemset(S)

12. **for each** 1 − itemset P in S **do**
13. **if** $Sup(P) \geq \xi$ **then**
14. **if** *hastheSameLengh* = *False* **then**
15. Scan P.STSet to calculate $UBO(P)$ based on L;
16. **if** $UBO(P) \geq \xi$ **then**
17. $C_1 \leftarrow C_1 \cup \{P\}$;
18. **if** $O(P) \geq \xi$ ***then***
19. $HO_1 \leftarrow HO_1 \cup \{P\}$;
20. **else**
21. $C_1 \leftarrow C_1 \cup \{P\}$;
22. **if** $O(P) \geq \xi$ **then**
23. $HO_1 \leftarrow HO_1 \cup \{P\}$;
24. Return $\{C_1, HO_1\}$;

Mine-HOI-KItemset (C_{k-1})

25. $C_k \leftarrow \emptyset$; // Dictionary of EquivalenceClass and list of candidates
26. $HO_k \leftarrow \emptyset$;
27. **While** $|C_{k-1}| > 0$ **do**
28. $P_1 = C_{k-1}[0]$;
29. **for** *each* itemset $P_2 \in C_{k-1}$ **do**
30. **if** IsSameEquivalenceClass(P_1, P_2) **then**
31. $P \leftarrow P_1 \cup P_2$;
32. P.STSet $\leftarrow P_1$.STSet $\cap$ P_2.STSet;
33. **if** $|P.STSet| \geq \xi$ **then**
34. **if** *hastheSameLengh* = *False* **then**
35. Scan $P.STSet$ to calculate $UBO(P)$ based on *P.STSet;*
36. **if** $UBO(P) \geq \xi$ **then**
37. $P_{tmp} \leftarrow \{P_0, P_1, \ldots. P_{P.length-1}\}$
38. $C_k \leftarrow C_k \cup \{P\}$ & IndexEquivalenceClass(P, P_{tmp})
39. **else**
40. $C_k \leftarrow C_k \cup \{P\}$;
41. $C_{k-1} = C_{k-1} \setminus C_{k-1}[0]$;
42. $HO_k \leftarrow \{P \,|P \in C_k \wedge O(P) \geq \xi\}$;
43. Return $\{C_k, HO_k\}$;

First, the FHOI algorithm scans the database to generate the set of 1-itemset (S) and stores the list of transaction id (id of the transactions that contain the 1-itemset in S). The algorithm also indexes the length of each transaction and determines whether every transaction has the same length (line 1). It then calls the *Mine-HOI-1Itemset* method (line 4) to mine high occupancy itemsets by looping each item P in S where the support of P is greater than or equal to ξ. There are two scenarios that this method needs to determine:

(i) Lines 16–19, if every transaction in the database does not have the same length, it will calculate the UBO. If the UBO is greater than or equal to ξ, P is a high occupancy itemset. Itemset P will be also added into a set of candidates for the next round of processing on *k-itemsets* ($k \geq 2$).
(ii) Lines 20–23, if all transactions in the database have the same length, P is a high occupancy itemset. Itemset P will also be added into a set of candidates for the next round of processing on *k-itemsets* ($k \geq 2$).

The algorithm will then loop to process until no more candidates are available (line 7). The set of candidates, C_{k-1}, returned by the *Mine-HOI-1Itemset* method will then be sent to the *Mine-HOI-KItemset* method (line 8). In the *Mine-HOI-KItemset* method, each candidate is denoted as an itemset P_i, it will be looped and compared with the other itemset (P_{i+1}) in a set of candidates C_{k-1}. If they are same equivalence class, the algorithm considers itemset P which is the union of P_i and P_{i+1} and also compiles the set of transactions containing P (lines 31, 32). Then if the presence of itemset P in the database is greater than or equal to ξ, the *Mine-HOI-KItemset* method will determine if P is a high occupancy itemset and also add P into the list of candidates for the next processing step (lines 33–40). The algorithm then takes the result of this method (lines 41–43) into processing at line 7.

While investigating the HEP algorithm, we learned that it requires a large amount of memory when mining HOIs from large or dense databases. This is because it must deal with a large number of candidates after processing each step $k \geq 2$.

Our new FHOI algorithm eliminates the unnecessary candidates and thus does not require a lot of memory resources in the way HEP does. However, we recognize that for each step $k \geq 2$, the FHOI algorithm still has to carry the list of candidates when it finishes the ***Mine − HOI − KItemset*** method. We thus investigate the characteristics of different databases. In dense databases, the item and itemset may appear in most of the transactions. As such, after finishing each step $k \geq 2$, ***Mine − HOI − KItemset*** returns a set of candidates in which each candidate may have a large number of items, and the length of the occupancy list for each candidate may also be large. This increases the amount of memory allocated to carry the list of candidates and their relevant information. By reviewing the breadth first search approach (Fig. 1) that FHOI is uses, we develop the DFHOI algorithm so that instead of carrying a list of candidates after each step k, we recursively mine HOIs using DFS and help the machine to release memory resources immediately after mining. Moreover, the DFHOI algorithm does not need to carry large numbers of candidates at any time.

Below is our state-of-the-art algorithm on mining HOIs using the DFS approach.

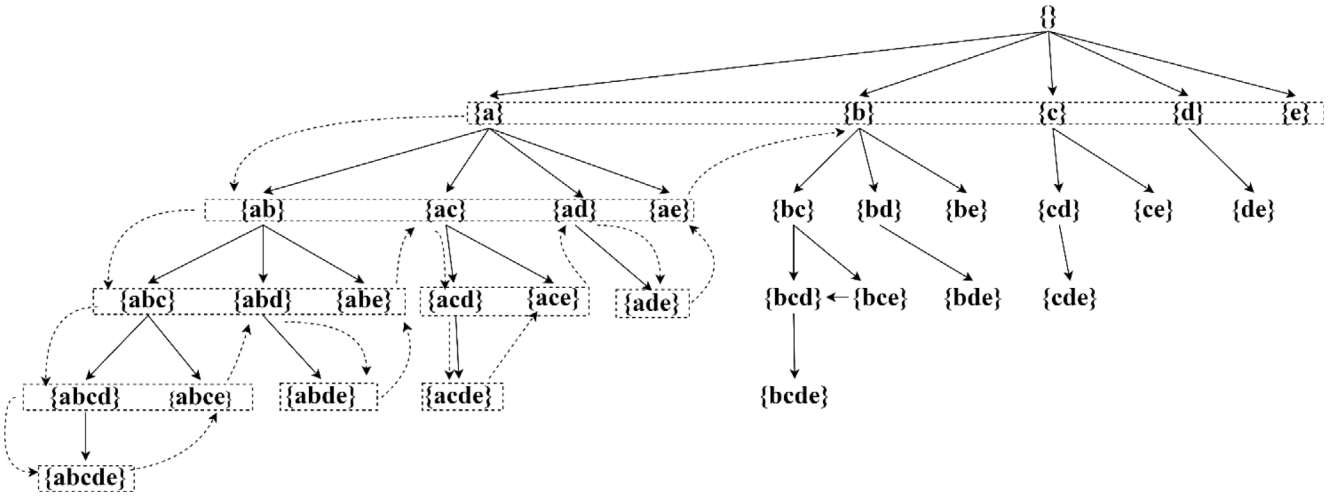

Fig. 2. An example of applying equivalence class using depth first search.

Algorithm 2: DFS for High Occupancy Itemset Mining - DFHOI

Input: database D and user-defined minimum occupancy threshold ξ

Output: $HOIs$

DFHOI()

1. Scan D to generate a set of 1-itemset S, with an index of length L and tidset of each 1-itemset, and determine if D has the same length (*hastheSameLengh*).
2. $HOIs \leftarrow \emptyset$;
3. $C_1 \leftarrow \emptyset; HOI_1 \leftarrow \emptyset$;
4. $\{C_1, HOI_1\} \leftarrow$ Mine-HOI-1Itemset(S);
5. $HOIs \leftarrow HOIs \cup HOI_1$;
6. **If** $(C_1 \neq \emptyset)$ **do**
7. $HOIs \leftarrow HOIs \cup$ Mine-Depth-HOIs (C_1);
8. Return $HOIs$;

Mine-Depth-HOIs (C)

9. **for** *each* itemset $P_1 \in C$ **do**
10. $C_l \leftarrow \emptyset$;
11. **for** *each* itemset $P_2 \in C$ with P_2 following P_1 **do**
12. $P \leftarrow P_1 \cup P_2$;
13. $P.STSet \leftarrow P_1.STSet \cap P_2.STSet$;
14. **if** $|P.STSet| \geq \xi$ **then**
15. **if** *hastheSameLengh* = *False* **then**
16. Scan $P.STSet$ to calculate $UBO(P)$ based on L;
17. **if** $UBO(P) \geq \xi$ **then**
18. $C_l \leftarrow C_l \cup \{P\}$
19. **else**
20. $C_l \leftarrow C_l \cup \{P\}$;
21. **Mine-Depth-HOIs** (C_l);
22. $HOIs \leftarrow HOIs \cup \{P \mid P \in C_l \wedge O(P) \geq \xi\}$;

In the DFHOI algorithm, following idea described in Fig. 2 and Algorithm 1, after processing line 4, the algorithm has a set of HOIs where each itemset has 1 item, and a list of candidates C_1. Instead of processing multiple k levels like FHOI, DFHOI will start processing recursively on set of C_1 at line 7 and jump right away to line 9 to perform Mine-Depth-HOIs method recursively. After computing the equivalence class for each itemset to create candidates, Mine-Depth-HOIs is invoked at line 21 to continue the mining process deeply until no more candidates are generated.

4.2. Illustrations

4.2.1. FHOI algorithm

This section illustrates how the proposed FHOI algorithm works to mine high occupancy itemsets from the database given in Table 1 by applying candidate elimination strategies.

First, we suppose ξ is 25%, then we have:

$$\xi = 25\% \times |T| = 25\% \times 6 = 1.5$$

We illustrate the process of computing the support, *UBO* and *O* of 1-itemsets as follows (the results are shown in Table 2).

With item {a}:

- $Sup(\{a\}) = |STSet(\{a\})| = 4$
- $O(\{a\}) = \sum_{T \in T1,T2,T4,T6} \frac{1}{|T|} = \frac{1}{3} + \frac{1}{3} + \frac{1}{2} + \frac{1}{5} = 1.36$
- To compute $UBO(\{a\})$, first we must compute $l(\{a\})$ and $n(\{a\})$, based on Example 2, we have $l(\{a\}) = \{2, 3, 5\}$ and $n(\{a\}) = \{1, 2, 1\}$. From L, $l(\{a\})$ and $n(\{a\})$, we can compute UBO as follows:

$$UBO(\{a\}) = \max_{1 \leq x \leq u} UBO^x_{\{(a)\}} = \max_{1 \leq x \leq 3} \sum_{i=x}^{3} n_i(\{a\}) \times \frac{l_x(\{a\})}{l_i(\{a\})}$$
$$= \max(1 \times \frac{2}{2} + 2 \times \frac{2}{3} + 1 \times \frac{2}{5}, 2 \times \frac{3}{3} + 1 \times \frac{3}{5}, 1 \times \frac{5}{5}) = 2.73.$$

Similarly, we can compute the information of item {b} as follows:

- $Sup(\{b\}) = |STSet(\{b\})| = 3$
- $O(\{b\}) = \sum_{T \in \{T2,T3,T6\}} \frac{1}{|T|} = \frac{1}{3} + \frac{1}{4} + \frac{1}{5} = 1.36$
- To compute $UBO(\{b\})$, we have $l(\{b\}) = \{3, 4, 5\}$ and $n(\{a\}) = \{1, 1, 1\}$. From L, $l(\{b\})$ and $n(\{b\})$, we can compute $UBO(\{b\})$ as follows:

$$UBO(\{b\}) = \max_{1 \leq x \leq u} UBO^x_{\{b\}} = \max_{1 \leq x \leq 3} \sum_{i=x}^{3} n_i(\{b\}) \times \frac{l_x(\{b\})}{l_i(\{b\})}$$
$$= \max(1 \times \frac{3}{3} + 1 \times \frac{3}{4} + 1 \times \frac{3}{5}, 1 \times \frac{4}{4} + 1 \times \frac{4}{5}, 1 \times \frac{5}{5}) = 2.35.$$

Table 2
The index, support, occupancy and *UBO* of each 1-itemset.

Index	1-itemset	$Sup(P)$	$UBO(P)$	$O(P)$
1	{a}	4	2.73	1.36
2	{b}	3	2.35	0.78
3	{c}	4	3.35	1.12
4	{d}	6	4.35	1.95
5	{e}	3	2.35	0.78

Table 3
The length of *OL*, occupancy, and upper-bound occupancy of each 2-itemset.

Index	2-itemsets	$\|OL(P)\|$	$UBO(P)$	$O(P)$
1	{ab}	2	1.6	1.07
1	{ac}	2	1.6	1.07
1	{ad}	4	2.73	2.73
1	{ae}	1	2.6	0.4
2	{bc}	2	1.8	0.9
2	{bd}	3	2.35	1.57
2	{be}	2	1.8	0.9
3	{cd}	4	2.35	2.23
3	{ce}	3	2.35	1.57
4	{de}	3	2.35	1.56

From the results shown in Table 2, and comparing them to ξ, we have:

- C_1: {a}, {b}, {c}, {d}, {e}
- HO_1: {d}

Based on C_1, we continue to find the STSet of all 2-itemsets by intersecting the STSet of two 1-itemsets, and these 2-itemsets also inherit the index of the parent node, we thus have:

1: *STSet*({*ab*}): $\{T_1, T_6\}$
1: *STSet* ({*ad*}) : $\{T_1, T_2, T_4, T_6\}$
1: *STSet* ({*ae*}) : $\{T_6\}$
2: *STSet* ({*bc*}) : $\{T_3, T_6\}$
2: *STSet* ({*bd*}) : $\{T_2, T_3, T_6\}$
2: *STSet* ({*be*}) : $\{T_3, T_6\}$
2: *STSet* ({*cd*}) : $\{T_1, T_3, T_5, T_6\}$
3: *STSet* ({*ce*}) : $\{T_3, T_5, T_6\}$
3: *STSet* ({*de*}) : $\{T_3, T_5, T_6\}$

When $k \geq 2$, we will have to calculate the length of *OL*, so we will obtain the length of *OL*, occupancy, upper-bound occupancy of each 2-itemset.

Based on the results in Table 3, we compare the length of *OL* to ξ, then comparing UBO(P) and O(P) to ξ, we have:

- C_2: {ab} , {ac} , {ad} , {bc} , {bd} , {be} , {cd} , {ce} , {de}.
- HO_2 : {ad}, {bd}, {cd}, {ce}, {de}.

In Table 3, we can see that $UBO\{ae\} = 2.6 > \xi$, and $O\{ae\} = 0.4 < \xi$, when using the improvement methods, we have $|OL\{ae\}| = 1 < \xi$, we can eliminate {15} immediately without calculating $UBO\{ae\}$.

Based on C_2 and the index of each 2-itemset, we can easily generate all 3-itemsets:

1: *STSet* ({*abc*}) : $\{T_6\}$
1: *STSet* ({*abd*}) : $\{T_2, T_6\}$
1: *STSet* ({*acd*}) : $\{T_1, T_6\}$
2: *STSet* ({*bcd*}) : $\{T_3, T_6\}$
2: *STSet* ({*bce*}) : $\{T_3, T_6\}$
2: *STSet* ({*bde*}) : $\{T_3, T_6\}$

Table 4
The length of OL, occupancy, and upper-bound occupancy of each 2-itemset.

Index	3-itemsets	$\|OL(P)\|$	$UBO(P)$	$O(P)$
1	{abc}	1	1	0.6
1	{abd}	2	1.6	1.6
1	{acd}	2	1.6	1.6
2	{bcd}	2	1.8	1.35
2	{bce}	2	1.8	1.35
2	{bde}	2	1.8	1.35
3	{cde}	3	2.35	2.35

3: *STSet* ({*cde*}) : $\{T_3, T_5, T_6\}$

Again, we calculate the length of OL, occupancy, and UBO of each 3-itemset.

Comparing the occupancy of each itemset in Table 4 to ξ, we have:

- C_3: {abd} , {acd} , {bcd}, {bce}, {bde}, {cde}
- HO_3: {abd} , {acd} , {*cde*}.

We continue to join two 3-itemsets that have the same index to look for all 4-itemsets from all the candidates in C_3, and then have:

1: *STSet* ({*abcd*}) : $\{T_6\}$
2: *STSet* ({*bcde*}) : $\{T_3, T_6\}$

We only obtain two 4-itemsets and they do not have the same equivalence class, so they cannot create a 5-itemset, and we only calculate the occupancy:

$$O(abcd) = \frac{5}{5} = 1 < \xi$$

$$O(bcde) = \frac{4}{4} + \frac{4}{5} = 1.8 > \xi$$

So, at $k = 4$ we obtain only one HO:
HO_4: {*bcde*}
Then, let $HO = HO_1 \cup HO_2 \cup HO_3 \cup HO_4$, and the FHOI algorithm returns HO as the list of high occupancy itemsets from the given sample database, as below:

HO = {{*d*} , {*ad*} , {*bd*} , {*cd*} , {*ce*} , {*de*} , {abd} , {acd} , {*cde*} , {*bcde*}}

4.2.2. DFHOI algorithm

Similar to the FHOI algorithm, in the first step the DFHOI algorithm also extracts the same information as shown in Table 2.

- C_1: {*a*}, {*b*}, {*c*}, {*d*}, {*e*}
- *HOIs*: {*d*}

From the above result, the DFHOI algorithm performs *Mine-Depth-HOIs* recursively as follows:

-***Mine-Depth-HOIs***({*a*}, {*b*}, {*c*}, {*d*}, {*e*}), result:

- C_l: {*ab*}, {*ac*}, {*ad*}
- *HOIs*: {*d*}, {ad}
- ***Mine-Depth-HOIs***({*ab*}, {*ac*}, {*ad*}), result:
  - C_l: {*abd*}
  - *HOIs*: {*d*}, {ad}, {*abd*}
  - ***Mine-Depth-HOIs***({*abd*}), result:
    - C_l: {}
    - *HOIs*: {*d*}, {ad}, {*abd*}
  - C_l: {*acd*}
  - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*}

- ***Mine-Depth-HOIs***({*acd*}), result:
  - C_l: {}
  - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*}

- C_l: {*bc*}, {*bd*}, {*be*}
- *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*}
- ***Mine-Depth-HOIs***({*bc*}, {*bd*}, {*be*}), result:
  - C_l: {*bcd*}, {*bce*}
  - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*}
  - ***Mine-Depth-HOIs***({*bcd*}, {*bce*}), result:
    - C_l: {*bcde*}
    - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*}
    - ***Mine-Depth-HOIs***({*bcde*}), result:
      - C_l: {}
      - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*}
    - ***Mine-Depth-HOIs***({*bce*}), result:
      - C_l: {}
      - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*}
  - ***Mine-Depth-HOIs***({*bde*}), result:
    - C_l: {}
    - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*}
- ***Mine-Depth-HOIs***({*cd*}, {*ce*}), result:
  - C_l: {*cde*}
  - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*} , {*cd*} , {*ce*}
  - ***Mine-Depth-HOIs***({*cde*}), result:
    - C_l: {}
    - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*} , {*cd*} , {*ce*} , {*cde*}
- ***Mine-Depth-HOIs***({*de*}), result:
  - C_l: {}
  - *HOIs*: {*d*}, {ad}, {*abd*} , {*acd*} , {*bd*} , {*bcde*} , {*cd*} , {*ce*} , {*cde*} , {*de*}

5. Experiments

In this section, we evaluated the performance of the FHOI algorithm and DFHOI algorithm compared to the HEP algorithm in terms of runtime, memory usage and number of eliminated candidates. The algorithms were developed using C# programing language. The testing machine was a sixth-generation quad-core 64-bit Core-i7 processor, clocked at 2.5 GHz (6500U), 16 GB RAM, and running Windows 10.

The list of databases for evaluation is shown in Table 5. They were downloaded from the SPMF website [57] .

The FHOI and HEP algorithms were executed on different databases with different input minimum occupancy thresholds, which are specific percentage values of a database's size. The results for the numbers of high occupancy itemsets on each corresponding database are reported in Table 6, Tables 7 and 8.

5.1. Runtime

We executed the HEP, FHOI and DFHOI algorithms and compared the runtimes on different databases with different ξ inputs. The speed of the FHOI and DFHOI algorithm are much greater than that of the HEP algorithm.

Table 5
Experimental databases.

Database	Type	Transaction count	Item count	Average item count per transaction
Foodmart	Sparse	4,141	1,559	4.42
FruitHut	Sparse	181,970	1,265	3.58
Online Retail	Sparse	541,909	2603	4.37
Retail	Sparse	88,162	16,470	10.3
T10I4D100K	Sparse	100,000	870	10.1
T40I10D100K	Sparse	100,000	942	39.6
Mushroom	Dense	8,416	119	23
Chess	Dense	3,196	75	37
Pumsb	Dense	49,046	2113	74

Table 6
Number of high occupancy itemsets from Foodmart, FruitHut, Online Retail.

FruitHut		Foodmart		Online Retail	
ξ (%)	No. of HOIs	ξ (%)	No. of HOIs	ξ (%)	No. of HOIs
0.9	28	0.09	185	0.40	112
0.7	44	0.08	337	0.30	215
0.5	63	0.07	568	0.20	464
0.3	109	0.06	838	0.10	2,004
0.1	438	0.05	1,117	0.05	7,019

Table 7
Number of high occupancy itemsets from Retail, T10I4D100K, T40I10D100K.

Retail		T10I4D100K		T40I10D100K	
ξ (%)	No. of HOIs	ξ (%)	No. of HOIs	ξ (%)	No. of HOIs
0.12	276	0.10	4,363	0.55	1
0.10	395	0.08	7,872	0.5	3
0.08	561	0.06	12,640	0.45	6
0.06	870	0.04	20,895	0.4	13
0.04	1,662	0.02	38,588	0.35	20

Table 8
Number of high occupancy itemsets from Mushroom, Chess, Pumsb.

Mushroom		Chess		Pumsb	
ξ (%)	No. of HOIs	ξ (%)	No. of HOIs	ξ (%)	No. of HOIs
13	5	48	0	86	0
11	198	46	0	84	0
9	3,266	44	0	82	0
7	27,123	42	0	80	0
5	93,266	40	0	78	0

FruitHut is a database of customer transactions from a US retail store that sold fruits. It contains 181,970 transactions and 1,265 unique items. It has an average of 3.58 items per transaction and the largest transactions contain 36 items. We executed the HEP, FHOI and DFHOI algorithms on the FruitHut database with various values of ξ, decreasing from 0.9% to 0.1% of the number of transactions. The total time to execute the FHOI and DFHOI algorithms were much shorter than that to execute the HEP algorithm. Based on the experimental results using FruitHut, the average execution time of the DFHOI algorithm was 96% faster than that of HEP with the same input. For instance, with $\xi = 0.9\%$, the time to execute the HEP algorithm was 10.05 s while the FHOI algorithm only required 0.35 s and the DFHOI algorithm also required only 0.30 s. In another sample on this database with $\xi = 0.1\%$, the HEP algorithm needed 14.09 s while the FHOI algorithm only needed 0.88 s and DFHOI algorithm only needed 0.77s. The comparison details are shown in Fig. 3.

Foodmart is a common database containing customer transactions from a retail store. This database has been used to evaluate the performance of many data mining algorithms. In our experiment, we compared the runtimes for mining high occupancy itemsets from Foodmart with different ξ inputs using the HEP,

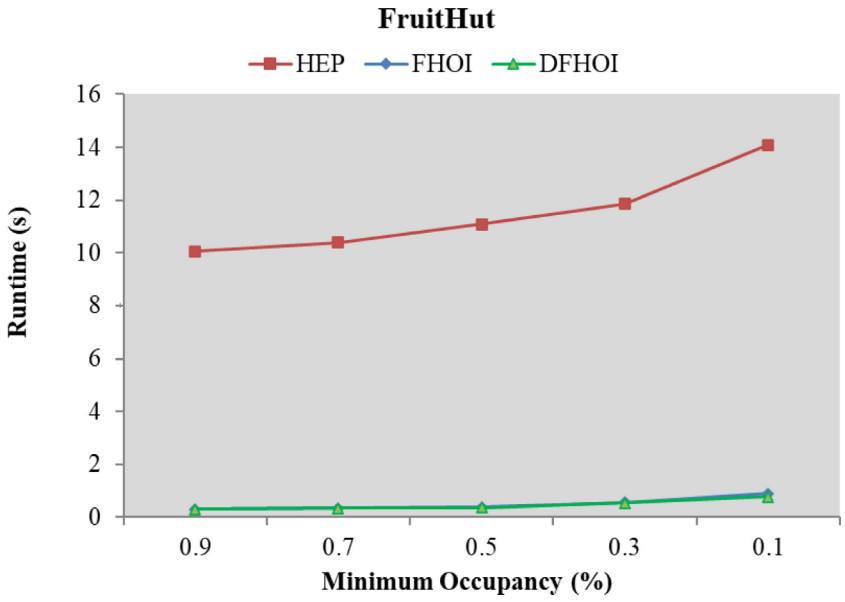

Fig. 3. Mining HOIs runtime on the FruitHut database.

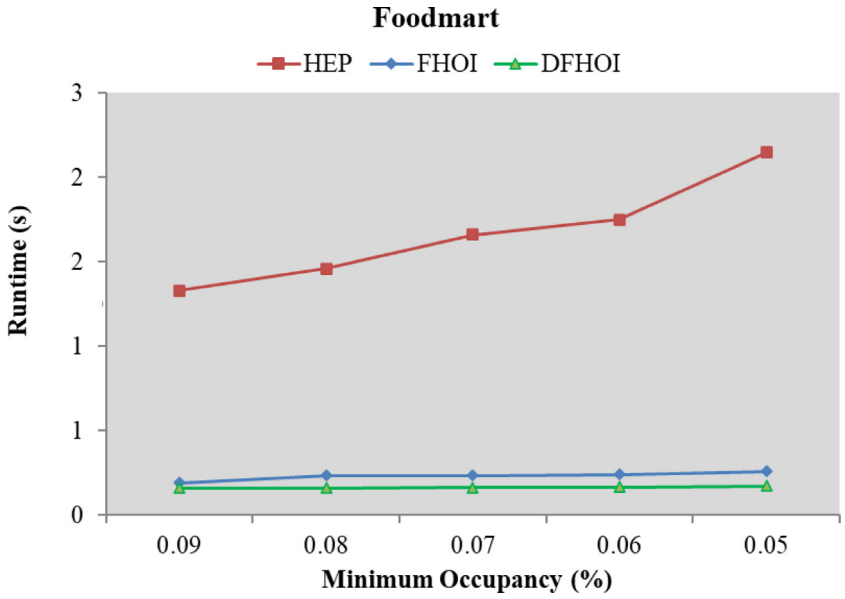

Fig. 4. Mining HOIs runtime on the Foodmart database.

FHOI and DFHOI algorithms. As shown in Fig. 4, the performance of the DFHOI algorithm on Foodmart in terms of runtime was much better than that of the HEP algorithm. It was also faster than FHOI. Based on the experimental results which tested both algorithms with $\xi \in \{0.09\%, 0.08\%, 0.07\%, 0.06\%, 0.05\%\}$, the average runtime of the DFHOI algorithm is 90.01% faster compared to that of the HEP algorithm and is 29% faster compared to the FHOI algorithm. For instance, to mine 1,117 high occupancy itemsets when $\xi = 0.05\%$, HEP algorithm required 2.15 s, FHOI required 0.26 s while DFHOI only required 0.17 s. As seen in Fig. 3, below, the processing time of the HEP algorithm tended to increase when the input ξ decreased, while the processing time of the FHOI and DFHOI algorithm on Foodmart database were much stabler and did not increase much when ξ decreased. The runtimes using the HEP algorithm with $\xi = 0.09\%$ and $\xi = 0.05\%$ were 1.33 s versus 2.15 s, a 0.82 s increase. Meanwhile, the runtimes using the DFHOI algorithm with same input ξ were 0.16 s versus 0.17 s, a difference of 0.01 s.

We also compared the runtimes of the HEP, FHOI and DFHOI algorithms on Online Retail, which has a larger number of transactions than that of FruitHut and Foodmart, in order to evaluate the effectiveness of our approach. These algorithms processed the same database with 541,909 transactions using same set of ξ input $\in\{0.4\%, 0.3\%, 0.2\%, 0.1\%, 0.055\%\}$, the corresponding runtimes of the HEP algorithm were $\in \{52.71s, 55.27s, 56.51s, 59.99s, 63.52s\}$ while the corresponding runtimes of the FHOI algorithm were $\in \{1.71s, 2.20s, 2.92s, 3.11s, 3.42s\}$ and the corresponding runtimes of the FHOI algorithm were $\in \{1.29s, 2.08s, 2.66s, 3.00s,$

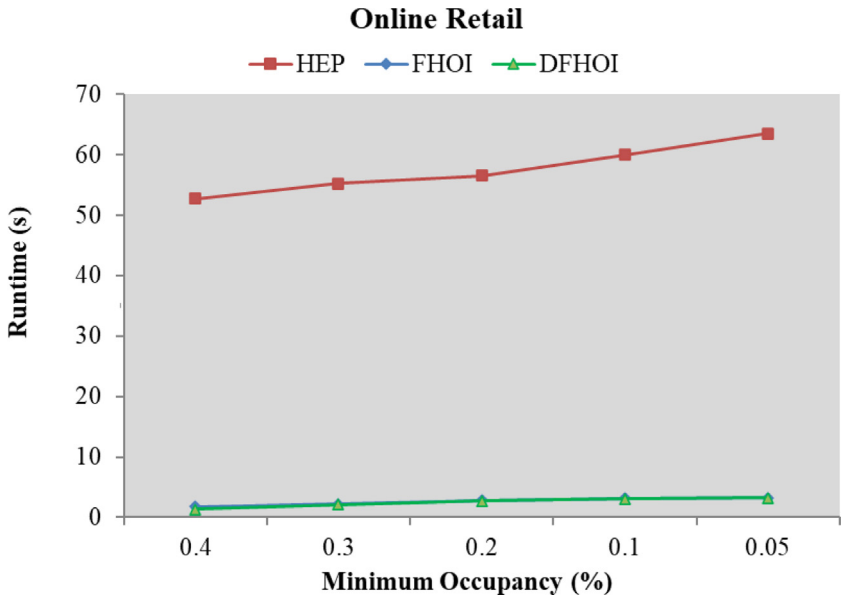

Fig. 5. Mining HOIs runtime on the Online Retail database.

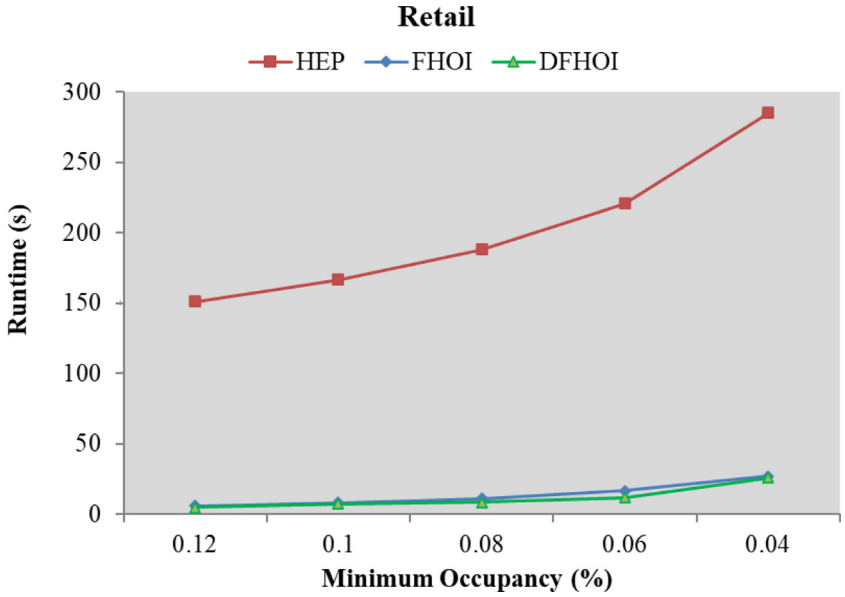

Fig. 6. Mining HOIs runtime on the Online Retail database.

3.20s}. The results of the comparison shown below in Fig. 5 indicate that the DFHOI algorithm is much faster (by an average of 95.8%) than the HEP algorithm in terms of runtime.

We also tried another database, Retail, which is the database containing customer transactions from an anonymous Belgian retail store, which has a large number of unique items (16,470 items). Fig. 6 shows that the DFHOI algorithm also ran faster than the FHOI algorithm and the HEP algorithm on Retail with different ξ input $\in${0.12%, 0.10%, 0.08%, 0.06%, 0.04%}. For instance, with $\xi = 0.12\%$, HEP needed 151.13 s, FHOI needed 6.19 s while DFHOI only needed 5.09 s; with $\xi = 0.04\%$, HEP ran in 285.20 s, FHOI needed 27.15 s, but DFHOI only needed 25.85 s.

Similarly, we also mined high occupancy itemsets using the HEP, FHOI and DFHOI algorithms on T10I4D100K, and the results indicated that the HEP algorithm ran much slower than our FHOI and DFHOI algorithms. There is not much difference in runtime between the FHOI and DFHOI algorithms. We modified the ξ input five times for T10I4D100K when evaluating the runtime. When the ξ was decreased, the number of HOIs increased dramatically, and thus the runtime also increased. When the threshold was changed from 0.1% to 0.02% on database T10I4D100K, the number of HOs increased from 4,363 to 12,640 (Table 7), and the runtime of the HEP algorithm rose from 17.89 s to 97.5 s while the FHOI algorithm was still much faster, its runtime also increased from 5.09 s to 10.2 s, similarly, while the DFHOI algorithm also performed faster its runtime also increased, from 5.08 s to 10.06 s. In particular, when we executed the HEP algorithm, the processing time to mine high occupancy itemsets with $\xi = 0.02\%$ was much higher than that with $\xi = 0.04\%$, with 97.50 s vs 36.65 s, an increase of 166%. Meanwhile, the DFHOI algorithm also used

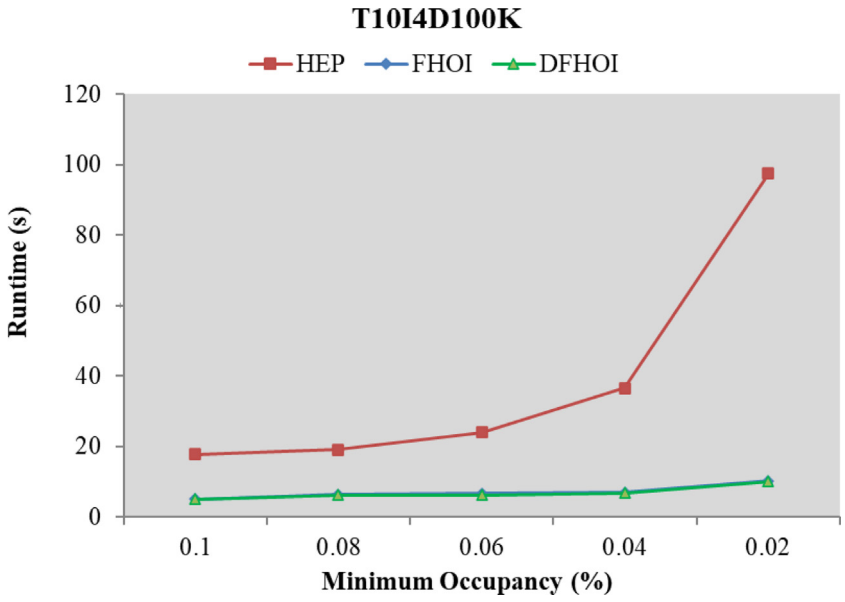

Fig. 7. Mining HOIs runtime on the T10I4D100K Database.

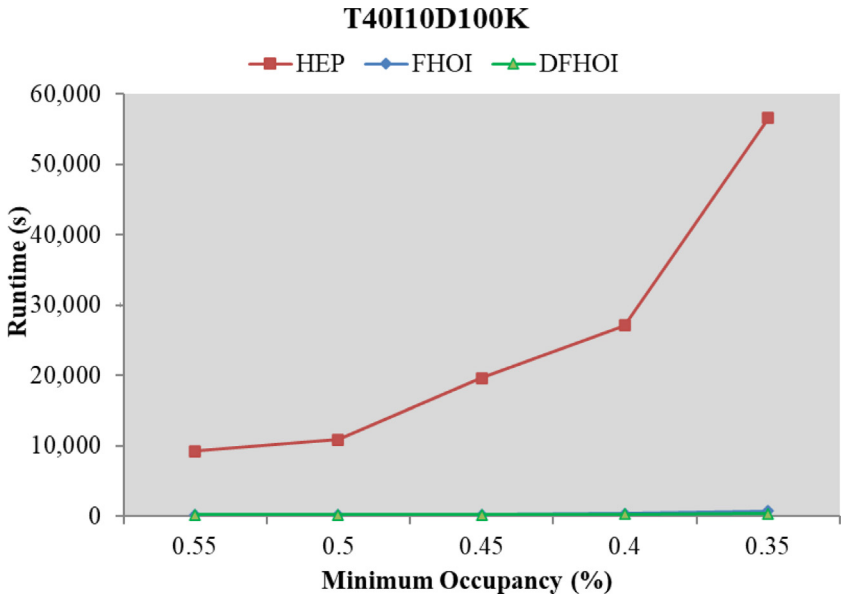

Fig. 8. Mining HOIs runtime on the T40I10D100K Database.

the same input ξ, but the processing times were 10.06 s vs 6.79 s, increasing by only 48.12%. Fig. 7 shows the runtime comparison among the HEP, the FHOI and DFHOI algorithms. In most of the testing scenarios, our DFHOI algorithm also performed extremely fast.

We then also ran test cases on another database, T40I10D100K, which has same number of transactions as the T10I4D100K database but the average number of items per transaction is larger (39.6 items). The DFHOI algorithm executed very much faster than HEP did. For instance, when $\xi = 4\%$, HEP algorithm performed mining in 27,116.72 s, while the DFHOI algorithm needed less than 98.9% of this, or just 306.83 s. It was also same when $\xi = 5\%$, as the DFHOI algorithm was much faster than HEP algorithm, at 56,633.46 s vs 420.02 s, or in 99.25% less time. The runtime performance report on this database is shown in Fig. 8.

In sparse databases, our proposed algorithms, FHOI and DFHOI, always bring more benefits to users in term of runtime, and the system will not have to wait for a long time to receive result when our approach is applied. This benefits the decision support systems as well as help the managers to make faster decision in their business activities.

We also evaluated the performance of the DFHOI and FHOI algorithms compared to the HEP algorithm in terms of runtime on the Mushroom database, which is a dense database with 119 unique items and each transaction has the same length, 23 items. During testing, we varied the ξ value from 5% to 13%, and Fig. 9 shows the comparison of runtimes among the algorithms. When ξ went down, the HEP algorithm ran slowly while the FHOI algorithm ran faster and DFHOI algorithm performed much faster. For instance, when $\xi = 5\%$, a total of 93,266 high occupancy itemsets were mined and the FHOI algorithm ran 90.07% faster than the HEP algorithm did, DFHOI algorithm ran 97.73% faster than the HEP algorithm. The DFHOI algorithm ran also 22.8% faster than the FHOI algorithm did. Overall of the experimental results for the

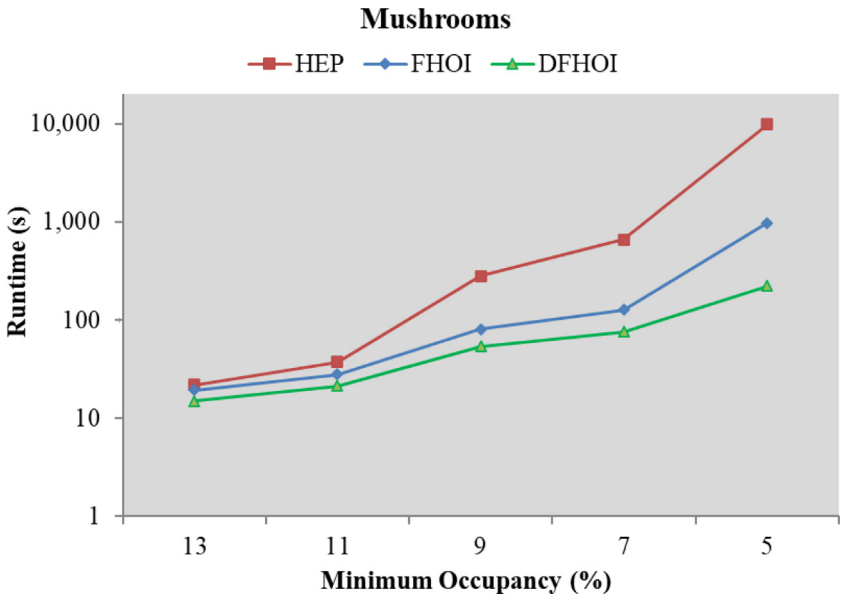

Fig. 9. Mining HOIs runtime on the Mushroom Database.

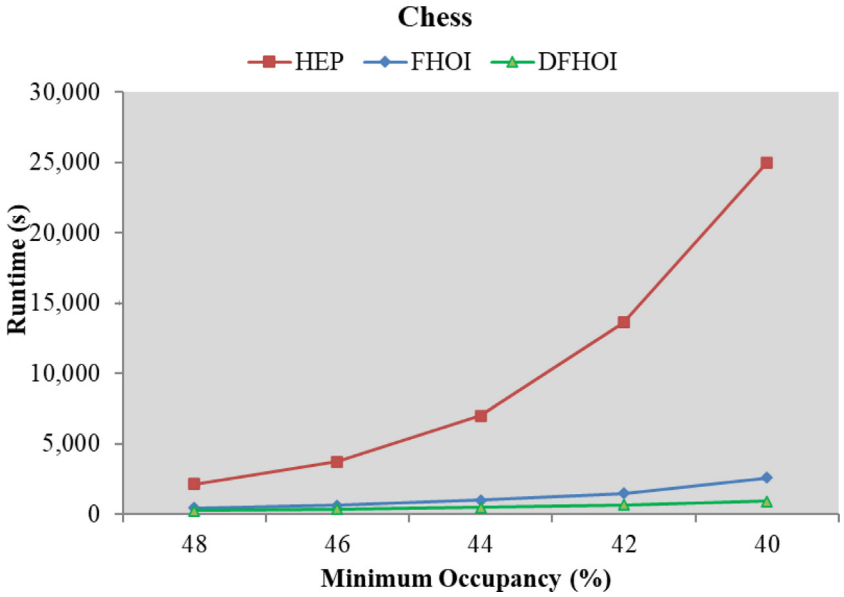

Fig. 10. Mining HOIs runtime on the Chess Database.

Mushroom database show that the DFHOI algorithm performed on average 96.42% faster than the HEP algorithm.

We then also compared the performance of the DFHOI, FHOI and HEP algorithms by executing more test cases for mining HOIs from the Chess and Pumsb databases. These databases are dense and have many transactions as well as many items in a transaction.

Fig. 10 compares the runtimes among the DFHOI, FHOI and HEP algorithms on the Chess database. We performed the task with different ξ input $\in$\{48%, 46%, 44%, 42%, 40%\}. We received no HOIs, but in term of runtime the DFHOI algorithm performed much faster than both FHOI and HEP did. When $\xi = 40\%$, DFHOI returned the result within $\sim\sim$917 s, FHOI returned the result within 2612s $\sim\sim$43 min $\sim\sim$0.7 h while HEP kept the user waiting 24,997 s $\sim\sim$417 min $\sim\sim$6.9 h. Fig. 10 indicates that our DFHOI algorithm is the best choice for mining high occupancy itemsets.

Similarly, on the Pumsb database we also tested the algorithms with different ξ values, and DFHOI always ran faster than FHOI and HEP did. The comparison is shown in Fig. 11.

The runtime of DFHOI algorithm is always faster in every testing scenario with the above sparse and dense databases. The results indicated our DFHOI algorithm is more efficient than the HEP algorithm in term of processing time. Table 9 shows the average runtime comparison for all testing scenarios on all testing databases.

5.2. Memory usage

This section discusses how DFHOI and FHOI used less memory than HEP did. In all the experimental databases, the total memory usage of the FHOI algorithm was lower than that of the HEP algorithm, although the total memory usage of the DFHOI algorithm was much lower than that of both the FHOI and HEP algorithms.

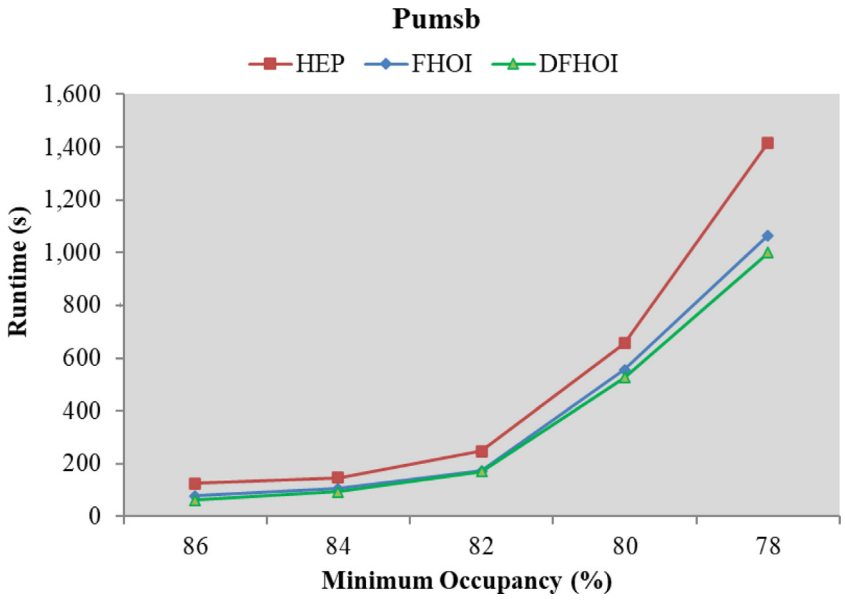

Fig. 11. Mining HOIs runtime on the Pumsb Database.

Table 9
Average runtime comparison.

Database	HEP Avg. Runtime (s)	FHOI Avg. Runtime (s)	DFHOI Avg. Runtime (s)	FHOI/HEP Ratio (%) $\frac{FHOI-Avg.Runtime(s)}{HEP-Avg.Runtime(s)}$%	DFHOI/HEP Ratio (%) $\frac{DFHOI-Avg.Runtime(s)}{HEP-Avg.Runtime(s)}$%
FruitHut	11.49	0.49	0.47	4.3	4.0
Foodmart	1.67	0.23	0.16	13.9	9.9
Online Retail	57.60	2.61	2.45	4.5	4.2
Retail	202.45	13.87	11.78	6.9	5.8
T10I4D100K	39.06	7.10	6.91	18.2	17.7
T40I10D100K	27, 713.93	373.74	283.10	1.5	1.1
Mushroom	2, 178.04	247.50	77.93	11.4	3.6
Chess	10, 314.43	1, 240.54	546.17	12.0	5.3
Pumsb	518.50	395.76	370.34	76.3	71.4

The DFHOI algorithm applied two strategies to prune candidates during high occupancy itemset mining. The DFHOI algorithm also did not need to carry large number of candidates at a time by recursively mining. Without having to deal with a large number of candidates, the DFHOI algorithm has better performance than the FHOI and HEP algorithms in term of not only runtime, as discussed in Section 5.1, but also memory usage.

In our experiment with the FruitHut database, HEP, FHOI and DFHOI algorithms processed with the ξ input $\in$ {0.9%, 0.7%, 0.5%, 0.3%, 0.1%}, however the FHOI algorithm did not require large memory usage to mine the high occupancy itemsets. In our testing, the HEP algorithm required a max 8.25MB to mine data with $\xi = 0.1\%$, the max memory usage that FHOI algorithm needed was 5.99MB, 2.26MB less (approximated 27.37%). However, the DFHOI algorithm only required 4.03MB when working with $\xi = 0.1\%$, 1.96MB (∼32.8%) less compared to the FHOI algorithm and 4.22MB, 4.22MB (∼51.2%) less compared to the HEP algorithm. The comparison details are shown in Fig. 12.

Similar to the experiment on the FruitHut database, the experiment on the Foodmart database also showed that both of the proposed algorithms required less memory usage than the HEP algorithm. In particular, the DFHOI algorithm always consumed much less memory than the FHOI and HEP algorithms. The average memory usage that HEP algorithm needed was 2.64MB, but the FHOI algorithm needed 1.49MB and DFHOI algorithm needed only 0.81MB. Fig. 13 shows how much better the memory usage performance of our proposed algorithms were. Even when we decreased the value of ξ to 0.05%, the HEP algorithm required 422% more memory allocation compared to the DFHOI algorithm.

The testing results with the Online Retail database, as shown in Fig. 14, also described the better performance in term of memory consumption. When the ξ was decreased from 0.4% to 0.05%, the number of high occupancy itemsets increased dramatically from 112 to 7,019. In these testing scenarios, the memory for the HEP algorithm was increased from 14MB to 37Mb. However, the FHOI algorithm needed from 3MB (approximately 4 times less than that of HEP) to 25MB (approximately 0.48 times less than that of HEP), and the DFHOI algorithm needed from 2.9MB (approximately 4.8 times less than that of HEP) to 18.43MB (approximately 2 times less than that of HEP).

The smaller the input value, the more memory the HEP algorithm needed to be compared to the FHOI and DFHOI algorithms, as shown in Fig. 15 for the Retail database. By applying the pruning candidate strategy, the FHOI algorithm consumed less memory than the HEP algorithm because the number of candidates that the FHOI algorithm generated was much lower than the number the HEP algorithm generated. The DFHOI algorithm showed its extreme effectiveness in terms of memory usage when applying both the pruning candidate strategy and DFS. For example, with ξ 0.04% the HEP algorithm generated a total of 89,614 candidates, but the FHOI and DFHOI algorithms only generated a total of 25,945 candidates, 63,669 candidates fewer. In that testing scenario ($\xi = 0.04\%$) HEP needed 32MB storage, while FHOI required 11.06MB, 20.94MB less (approximately 65.42% less) and DFHOI only required 7.01MB (approximately 78.1% less). As seen

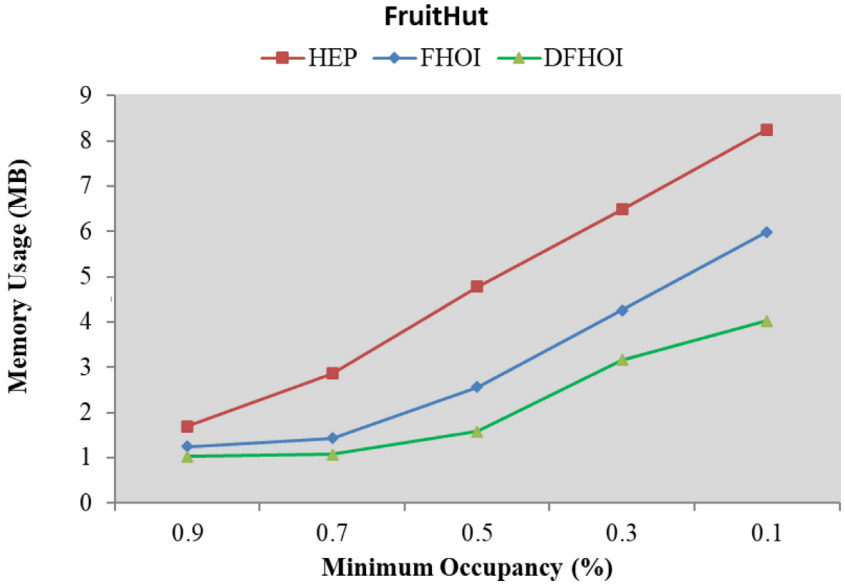

Fig. 12. Mining HOIs memory usage on the FruitHut database.

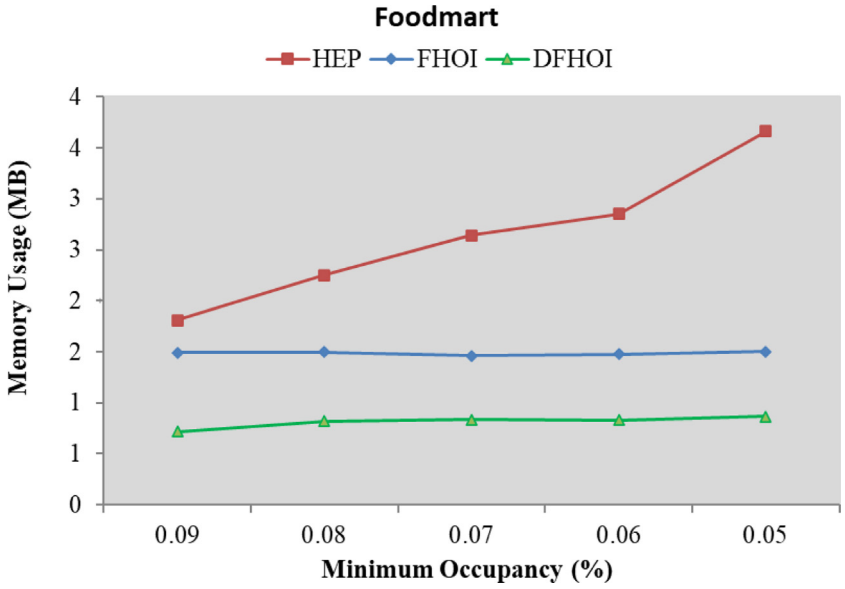

Fig. 13. Mining HOIs memory usage on the Foodmart database.

in Fig. 15, the performance on memory consumption of the DFHOI algorithm was much better than that of HEP when we varied the ξ from 0.08% to 0.04%, the difference was increased from 11.5MB (HEP: 16MB versus DFHOI: 4.5MB) to 24.99MB (HEP: 32MB versus DFHOI: 7.01MB).

Fig. 16, shown above, adds more evidence to prove the effectiveness of the DFHOI and FHOI algorithms compared to the HEP algorithm to mine high occupancy itemsets with $\xi \in \{0.1\%, 0.08\%, 0.06\%, 0.04\%, 0.02\%\}$. An average of 64.6MB of memory was consumed by the HEP algorithm, the FHOI algorithm consumed an average of 36.45MB, but the DFHOI algorithm consumed an average of 6.37MB. In Fig. 16, it can be seen that the DFHOI algorithm tended to consume much less memory than HEP did when ξ was decreased.

The T40I10D100K database has more items than T10I4D100K has, and the average number of items per transaction in T40I10D100K is also larger than that of T10I4D100K. The DFHOI algorithm also performed well to mine high occupancy itemsets. Details of the comparison are reported in Fig. 17.

We compared the memory usage among the HEP, FHOI and DFHOI algorithms on dense databases. In such databases an item may appear in most of the transactions, and thus the number of candidates may be high, the length of each candidate is large and the occupancy list for each candidate also contains many items. Our state-of-the-art algorithm, DFHOI, worked well with this kind of database and did not consume large amounts of memory. Figs. 18–20 show the comparison in terms of memory usage among the HEP, FHOI and DFHOI algorithms on the Mushrooms,

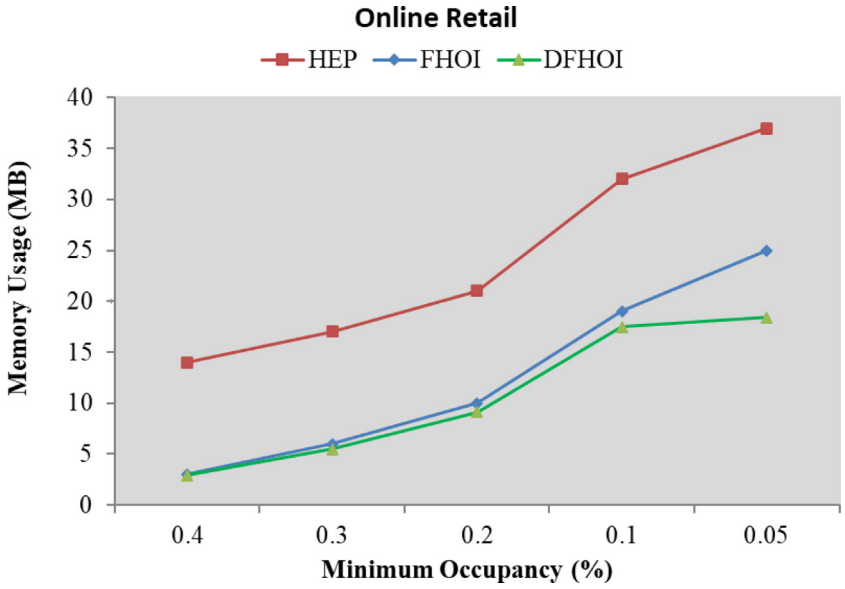

Fig. 14. Mining HOIs memory usage on the Online Retail database.

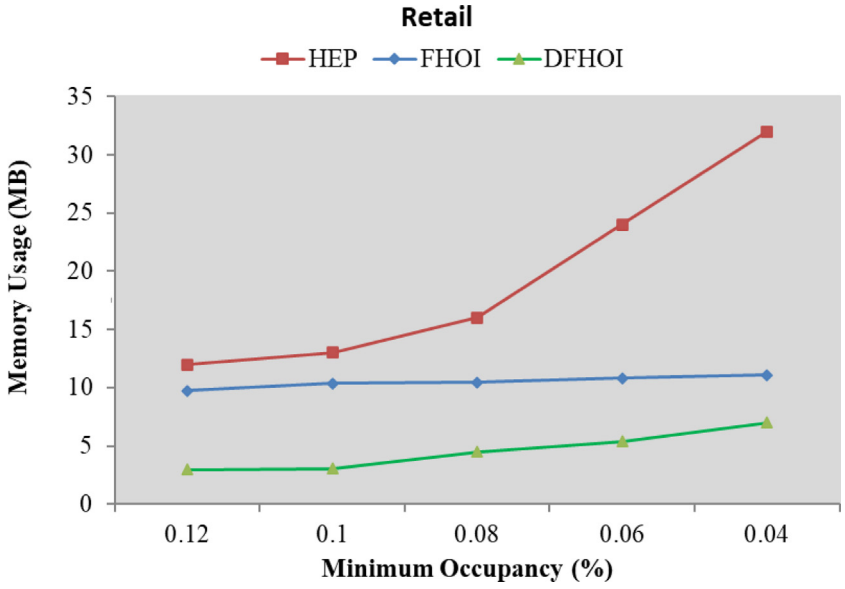

Fig. 15. Mining HOIs memory usage on the Retail database.

Chess and Pumsb databases. In all testing scenarios, the DFHOI algorithm always showed its superior ability with regard to not consuming large amounts of memory. Moreover, it did not tend to increase memory usage when we adjusted the input minimum occupancy threshold ξ.

Fig. 18 shows the results when we carried out an experiment to evaluate the memory usage using the Mushroom database. In this database, every transaction has the same length, and the number of candidates generated by the HEP, FHOI and DFHOI algorithms was same, although the FHOI algorithm indexes the transaction length to save memory with regard to storing the occupancy-list and eliminates the step of computing the upper bound. Therefore, the memory usage of FHOI is better than that of HEP. However, the performance of DFHOI algorithm was remarkable. For instance, when we input $\xi = 5\%$, HEP consumed 980.24MB, FHOI required 904.31MB but DFHOI only needed 23.99MB (97.6% less than that of HEP).

Fig. 19 also shows that the DFHOI algorithm is the best algorithm in terms of memory usage compared to the FHOI and HEP algorithms. In the testing scenarios with the Chess database, all three algorithms (HEP, FHOI and DFHOI) processed on same number of candidates which were generated in the whole process. However, the DFHOI algorithm performed mining recursively and did not have to hold a large number of candidates in the memory, and thus had the best performance. To further evaluate and prove the best ability of the DFHOI algorithm with regard to memory, we decreased $\xi = \{48\%, 46\%, 44\%, 42\%, 40\%\}$ and observed the memory consumption performance. The HEP

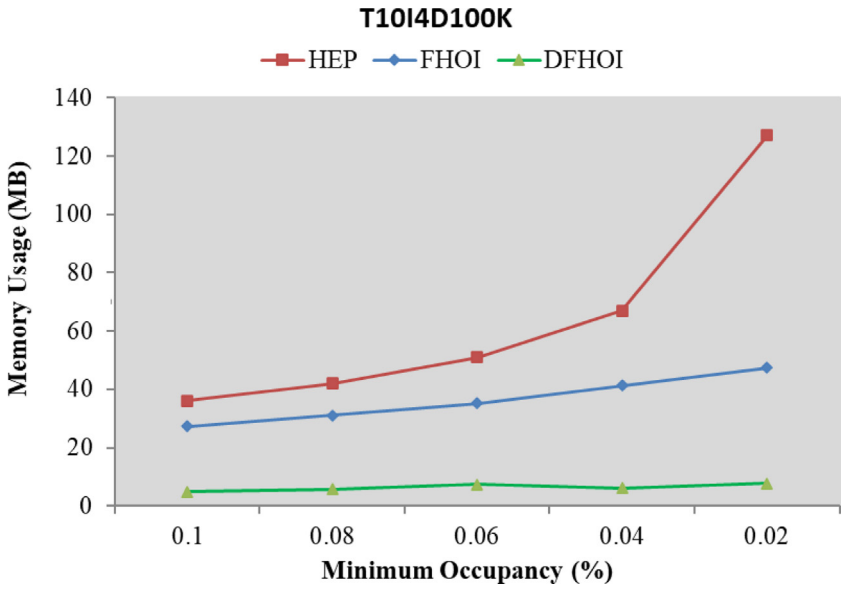

Fig. 16. Mining HOIs memory usage on the T10I4D100K database.

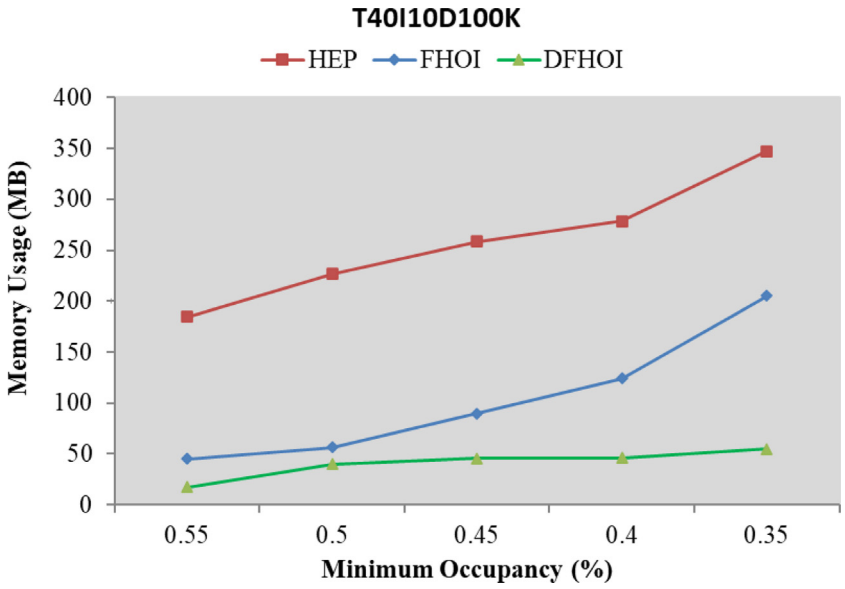

Fig. 17. Mining HOIs memory usage on the T40I10D100K database.

algorithm used an average of 2,443MB, the FHOI algorithm used an average of 894MB (1,550MB less than HEP), but the state-of-the-art DFHOI algorithm only needed 10.33MB for processing (99.6% less compared with HEP).

Fig. 20 compares the memory usage among the DFHOI, FHOI and HEP algorithms to mine HOIs from the Pumsb database. It shows that our proposed algorithms, especially the DFHOI algorithm, required less memory than the HEP algorithm. The DFHOI algorithm only needed an average of 52.04MB compared with 1,559.53MB for HEP. In particular, $at\xi = 78\%$ DFHOI only requested 99.63MB from the machine but the HEP algorithm needed 2,072.43MB.

The DFHOI algorithm performed well in our testing databases, with a fast runtime and less memory usage. The performance advantage obtained by eliminating unnecessary candidates will benefit the decision support and recommendation systems when it is applied. Table 10, below, shows the memory usage comparison in all testing scenarios on all testing databases which were examined in our experiments.

5.3. Candidate elimination

The experiments on the testing databases listed in Table 5 showed that both the DFHOI and FHOI algorithms are more efficient than the HEP algorithm in terms of runtime and memory usage. The DFHOI is optimized to perform mining extremely fast and also not require a large amount of memory allocation. The performance of the DFHOI algorithm is better due to applying the following pruning candidate strategies: i) checking the length of the occupancy-lists (OL) for each generated candidate and only processing candidates with a length greater than or equal to ξ; (ii) applying the equivalence class's property to the C_k generating process. Moreover, the DFHOI algorithm also performs the

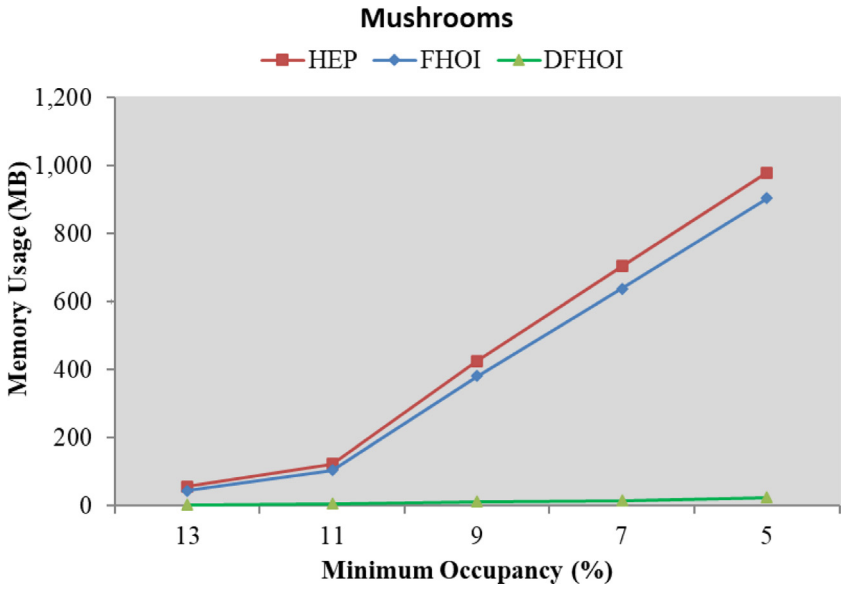

Fig. 18. Mining HOIs memory usage on the Mushrooms database.

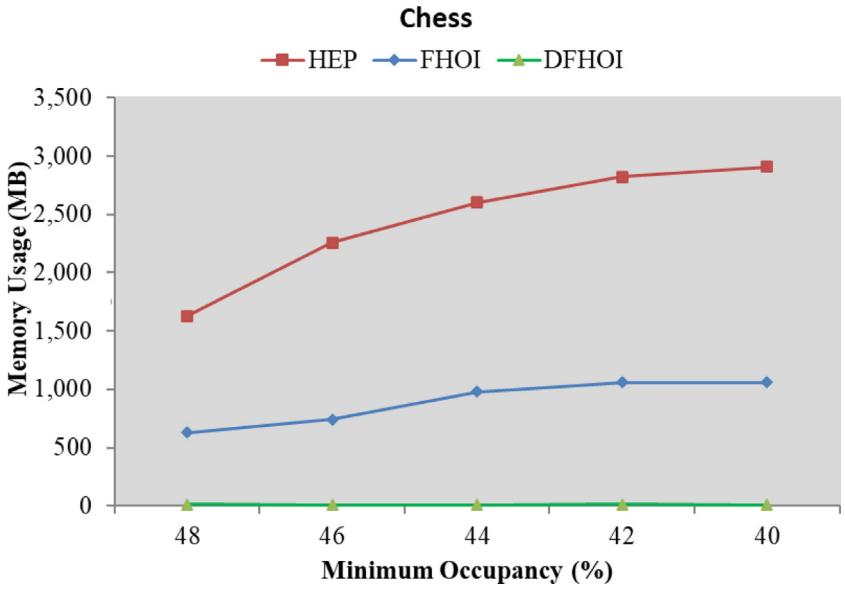

Fig. 19. Mining HOIs memory usage on the Chess database.

Table 10
Average memory usage comparison.

Database	HEP Avg. Memory Usage (MB)	FHOI Avg. Memory Usage (MB)	DFHOI Avg. Memory Usage (MB)	FHOI/HEP Ratio (%) $\frac{FHOI-Avg.MemoryUsage(MB)}{HEP-Avg.MemoryUsage(MB)}$ %	DFHOI/HEP Ratio (%) $\frac{DFHOI-Avg.MemoryUsage(MB)}{HEP-Avg.MemoryUsage(MB)}$ %
FruitHut	4.81	3.10	2.17	64.3	70.2
Foodmart	2.64	1.49	0.81	56.2	30.8
Online Retail	24.20	12.60	10.68	52.1	44.1
Retail	19.40	10.49	4.58	54.1	23.6
T10I4D100K	64.60	36.45	6.37	56.4	9.9
T40I10D100K	259.20	104.18	40.84	40.2	15.8
Mushroom	457.76	414.74	12.32	90.6	2.7
Chess	2, 443.47	893.51	10.33	36.6	0.4
Pumsb	1, 559.53	1, 372.54	52.04	88.0	3.3

whole process following DFS to avoid holding a large number of candidates at a time.

Table 11 is presented below to show the comparison of the number of candidates processed by the HEP and FHOI algorithms.

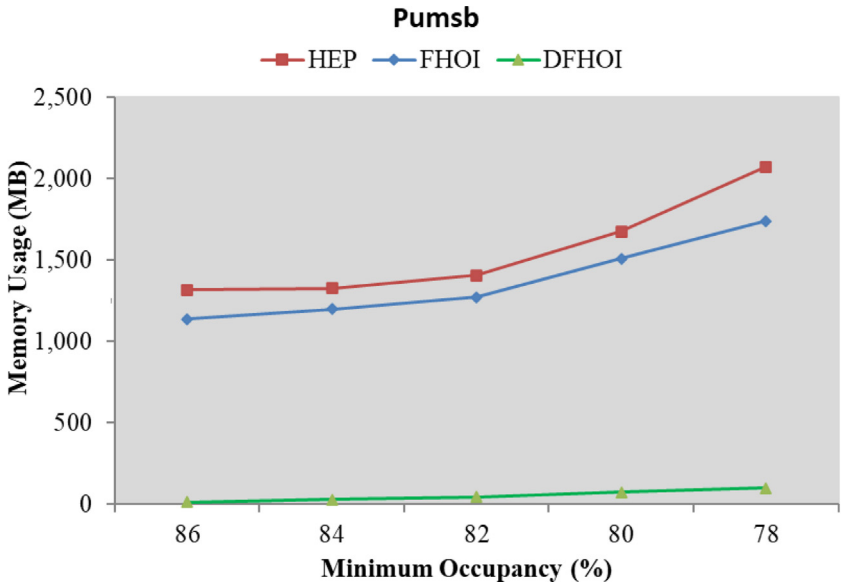

Fig. 20. Mining HOIs memory usage on the Pumsb database.

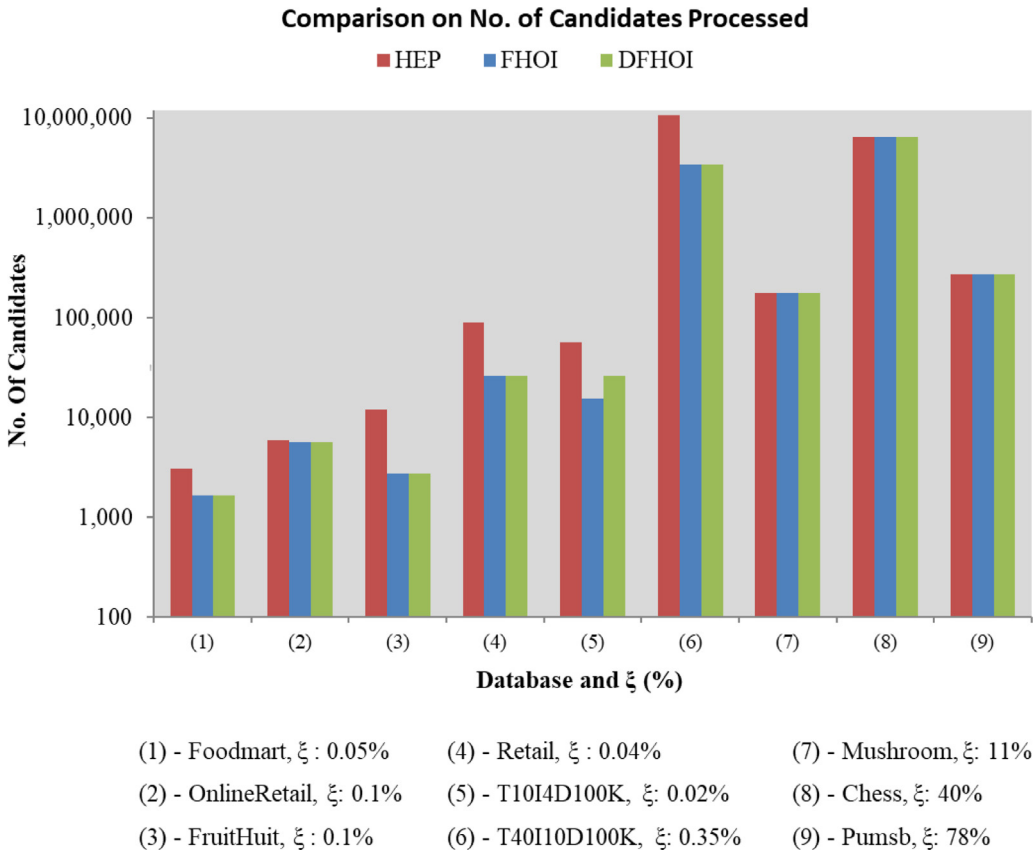

Fig. 21. Comparison of number of candidates processed.

Fig. 21 shows the number of candidates that the HEP algorithm and FHOI algorithm processed for each database with a corresponding representative user-defined ξ.

In sparse databases (Foodmart, Online Retail, FruitHut, Retail, T10I4D100K, and T40I10D100K), the FHOI algorithm always processes a smaller number of candidates than the HEP algorithm. In real applications, and in transactional databases, each transaction record from customers should always have different lengths of items that customers purchased. And thus, our algorithms, especially the DFHOI algorithm, should always demonstrate its superior performance and bring the most efficient decision making to managers.

In all experiments for spare databases, the total number of candidates which the proposed algorithms (DFHOI and FHOI) generated and processed was always less than the total number generated and processed by the HEP algorithm. For instance,

Table 11
Candidate Elimination Comparison.

Database	ξ (%)	Number of Candidates		Ratio(%)
		HEP Algorithm	FHOI/ DFHOI Algorithm	
FruitHut	0.9	912	133	14.58
	0.7	1,242	190	15.30
	0.5	1,868	281	15.04
	0.3	3,386	559	16.51
	0.1	11,938	2,763	23.14
Foodmart	0.09	1,630	1,557	95.52
	0.08	1,755	1,557	88.72
	0.07	1,900	1,644	86.53
	0.06	2,235	1,644	73.56
	0.05	3,056	1,644	53.80
Online Retail	0.4	663	540	81.45
	0.3	981	806	82.16
	0.2	1,963	1,755	89.40
	0.1	5,945	5,692	95.74
	0.05	14,134	13,938	98.61
Retail	0.12	26,958	5,951	22.08
	0.10	32,813	7,589	23.13
	0.08	41,521	10,356	24.94
	0.06	56,629	15,492	27.36
	0.04	89,614	25,945	28.95
T10I4D100K	0.1	47,993	27,532	57.37
	0.08	57,501	35,019	60.90
	0.06	70,132	44,583	63.57
	0.04	99,878	62,864	62.94
	0.02	235,653	129,875	55.11
T40I10D100K	0.55	4,057,797	1,135,088	27.97
	0.5	4,500,356	1,290,275	28.67
	0.45	6,064,745	1,489,025	24.55
	0.4	7,344,444	1,979,350	26.95
	0.35	10,685,757	3,448,216	32.27
Mushroom	13	114,179	114,179	100
	11	175,991	175,991	100
	9	618,189	618,189	100
	7	970,105	970,105	100
	5	4,137,547	4,137,547	100
Chess	48	1,743,135	1,743,135	100
	46	2,405,046	2,405,046	100
	44	3,333,197	3,333,197	100
	42	4,627,809	4,627,809	100
	40	6,439,702	6,439,702	100
Pumsb	86	13,599	13,599	100
	84	30,482	30,482	100
	82	67,903	67,903	100
	80	142,156	142,156	100
	78	272,336	272,336	100

on the T10I4D100K database we tried several test cases with $\xi \in \{0.1\%, 0.08\%, 0.06\%, 0.04\%, 0.02\%\}$, and the DFHOI and FHOI algorithms only processed an average of 60% the number of candidates that the HEP algorithm did. The proposed approach, the DFHOI and FHOI algorithms, obtained extremely good performance on the FruitHut and Retail databases. In FruitHut, when $\xi = 0.5\%$ the HEP algorithm generated 1,868 candidates while the new algorithms only generated 281 candidates, or $\frac{281}{1868} \sim 15\%$. Similarly, on Retail, the proposed algorithms only processed 23.13% the number of candidates that HEP did.

In the Mushroom, Chess, and Pumsb databases, HEP, FHOI and DFHOI algorithms processed the same number of candidates. Each transaction in the database has the same number of items and thus the number of candidates generated during mining is the same. However, as shown in Sections 5.1 and 5.2, the FHOI algorithm applied Definition 10 and theorem 3, and thus it still had better performance than HEP in terms of runtime and memory usage. Moreover, DFHOI is much more efficient than the FHOI and HEP algorithms due to applying DFS strategies to avoid allocating memory resources for storing a large number of candidates. Therefore, in the experiments the DFHOI algorithm always has outstanding performance, in terms of not only runtime but also memory usage.

6. Conclusion

In this research, we presented an approach to apply candidate elimination strategies to speed up the process of mining high occupancy itemsets, then proposed two new algorithms, FHOI and DFHOI. Our new algorithms prune the space search by checking the length of the occupancy-list as well as applying equivalence classes. In particular, the DFHOI algorithm applies the DFS strategy when processing candidates. As a result, both algorithms need less runtime and memory usage compared to the HEP algorithm. We tested and proved the effectiveness of the new algorithms by performing a high occupancy itemset mining task on a variety of databases using HEP, FHOI and DFHOI algorithms. By applying such a state-of-the-art and efficient algorithm, namely the DFHOI, recommendation and decision support systems will benefit from the rapidly generated results so that managers can quickly review and define strategies. Moreover, the

computer systems will also not require a large amount memory for mining process.

In future work we plan to research applying an approach for generating high occupancy association rules and applying the results to machine learning algorithms so that it can speed up and automate the process of decision making and prediction. We also intend to study the combination of occupancy and utility pattern mining [41] to develop an efficient approach for mining high utility occupancy itemsets.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] C.J.C. Jabbour, A.B.L.D.S. Jabbour, J. Sarkis, M.G. Filho, Unlocking the circular economy through new business models based on large-scale data: An integrative framework and research agenda, Technol. Forecast. Soc. Change 144 (2019) 546–552.

[2] G. Wang, J. Ma, S. Yang, An improved boosting based on feature selection for corporate bankruptcy prediction, Expert Syst. Appl. 41 (5) (2014) 2353–2361.

[3] I. Abbas, Role of data mining techniques in business, Indian J. Sci. Technol. 14 (5) (2021) 508–518.

[4] A. Ziegenbein, P. Stanula, J. Metternich, E. Abele, Machine learning algorithms in machining: A guideline for efficient algorithm selection, in: R. Schmitt, G. Schuh (Eds.), Advances in Production Research, in: WGP 2018, 2018, pp. 288–299.

[5] J.C. Feng, A. Kusiak, Data mining applications in engineering design, manufacturing and logistics, Int. J. Prod. Res. 44 (14) (2006) 2689–2694.

[6] S. Mainali, M.E. Darsie, K.S. Smetana, Machine learning in action: Stroke diagnosis and outcome prediction, Front. Neurol. 12 (2021).

[7] A.K. Arslan, C. Colak, M.E. Sarihan, Different medical data mining approaches based prediction of ischemic, Comput. Methods Progr. Biomed. 130 (2016) 87–92.

[8] A. Mumtaz, H.S. Le, K. Mohsin, T.T. Nguyen, Segmentation of dental X-ray images in medical imaging using neutrosophic orthogonal matrices, Expert Syst. Appl. 91 (2018) 434–441.

[9] R. Agrawal, R. Srikant, Fast Algorithms for Mining Sssociation Rules in Large Databases, VLDB, 1994, pp. 487–499.

[10] L.T.T. Nguyen, B. Vo, T. Mai, T.-L. Nguyen, A weighted approach for class association rules, in: Modern Approaches for Intelligent Information and Database Systems, 2018, pp. 213–222.

[11] M.J. Zaki, Scalable algorithms for association mining, IEEE Trans. Knowl. Data Eng. 12 (3) (2000) 372–390.

[12] Y. You, J. Zhang, Z. Yang, G. Liu, Mining top-k fault tolerant association rules, in: Proc. 2010 Intern. Conf. Intelligent Computing and Cognitive Informatics, 2010, pp. 470–473.

[13] Y. Djenouri, A. Belhadi, P. Fournier-Viger, H. Fujita, Mining diversified association rules in big datasets: A cluster/GPU/Genetic approach, Inform. Sci. 459 (2018) 117–134.

[14] O. Zaiane, M. El-Hajj, P. Lu, Fast parallel association rule mining without, in: First IEEE International Conference on Data Mining, 2001, pp. 665–668.

[15] D. Cheung, Y. Xiao, Effect of data distribution in parallel mining of associations, Data Min. Knowl. Discov. 3 (3) (1999) 291–314.

[16] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, SIGMOD Rec. (ACM Spec. Interes. Gr. Manag. Data) 29 (2000) 1–12.

[17] L. Tang, L. Zhang, P. Luo, M. Wang, Incorporating occupancy into frequent pattern mining for high quality pattern recommendation, in: ACM International Conference Proceeding Series, 2012, pp. 75–84.

[18] Z.H. Deng, Mining high occupancy itemsets, Future Gener. Comput. Syst. 102 (2020) 222–229.

[19] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: Current status and future directions, Data Min. Knowl. Discov. 15 (1) (2007) 55–86.

[20] R. Agrawal, C. Faloutsos, A.N. Swami, Efficient similarity search in sequence databases, in: Foundations of Data Organization and Algorithms, 1993, pp. 69–84.

[21] P. Fournier-Viger, J.C. Lin, R.U. Kiran, Y.S. Koh, R. Thomas, A survey of sequential pattern mining, Data Sci. Pattern Recognit. (DSPR) 1 (1) (2017) 54–77.

[22] J. Pei, J. Han, H. Lut, S. Nisho, S. Tang, D. Yang, H-mine: Fast and space-preserving frequent pattern mining in large databases, IIE Trans. 39 (6) (2007) 593–605.

[23] B. Vo, T. Le, F. Coenen, T.P. Hong, Mining frequent itemsets using the N-list and subsume concepts, Int. J. Mach. Learn. Cybern. 7 (2) (2016) 253–265.

[24] N. Aryabarzan, B. Minaei-Bidgoli, M. Teshnehlab, negFIN: An efficient algorithm for fast mining frequent itemsets, Expert Syst. Appl. 105 (2018) 129–143.

[25] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. Yu, Direct mining of discriminative and essential frequent patterns via model-based search tree, in: KDD '08: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 230–238.

[26] P. Fournier-Viger, C. Lin, B. Vo, T.C. Truong, J. Zhang, B.H. Le, A survey of itemset mining, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 7 (4) (2017).

[27] G. Grahne, J. Zhu, Fast algorithms for frequent itemset mining using FP-trees, IEEE Trans. Knowl. Data Eng. 17 (10) (2005) 1347–1362.

[28] Z.H. Deng, S.L. Lv, Fast mining frequent itemsets using nodesets, Expert Syst. Appl. 41 (2014) 4505–4512.

[29] Y. Wang, T. Xu, S. Xue, Y. Shen, D2P-Apriori: A deep parallel frequent itemset mining algorithm with dynamic queue, in: 2018 Tenth International Conference on Advanced Computational Intelligence, ICACI, 2018, pp. 649–654, http://dx.doi.org/10.1109/ICACI.2018.8377536.

[30] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, Data Min. Knowl. Discov. 8 (1) (2004) 53–87.

[31] M.J. Zaki, K. Gouda, Fast vertical mining using diffsets, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 326–335.

[32] T. Le, B. Vo, An N-list-based algorithm for mining frequent closed patterns, Expert Syst. Appl. 42 (19) (2015) 6648–6657.

[33] N. Aryabarzan, B. Minaei-Bidgoli, NEclatClosed: A vertical algorithm for mining frequent closed itemsets, Expert Syst. Appl. 174 (2021) 114738.

[34] T.D.D. Nguyen, L.T.T. Nguyen, L. Vu, B. Vo, W. Pedrycz, Efficient algorithms for mining closed high utility itemsets in dynamic profit databases, Expert Syst. Appl. 186 (2021) 115741.

[35] M. Zaki, C.-J. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, IEEE Trans. Knowl. Data Eng. 17 (4) (2005) 462–478.

[36] B. Huynh, B. Vo, V. Snasel, An efficient parallel method for mining frequent closed sequential patterns, in: IEEE Access, Vol. 5, 2017, pp. 17392–17402, http://dx.doi.org/10.1109/ACCESS.2017.2739749.

[37] K. Gouda, M.J. Zaki, GenMax: An efficient algorithm for mining maximal frequent itemsets, Data Min. Knowl. Discov. 11 (3) (2005) 223–242.

[38] S. Pham, T. Le, Z. Deng, B. Vo, A novel approach for mining maximal frequent patterns, Expert Syst. Appl. 73 (2017) 178–186.

[39] B. Huynh, C. Trinh, H. Huynh, T. Van, B. Vo, V. Snasel, An efficient approach for mining sequential patterns using multiple threads on very large databases, Eng. Appl. Artif. Intell. 74 (2018) 242–251.

[40] H. Huynh, L. Nguyen, B. Vo, Z.K. Oplatková, P. Fournier-Viger, U. Yun, An efficient parallel algorithm for mining weighted clickstream patterns, Inform. Sci. 582 (2022) 349–368.

[41] P. Fournier-Viger, C.J. Lin, R. Nkambou, B. Vo, V.S. Tseng, High-Utility Pattern Mining: Theory, Algorithms and Applications, Springer, ISBN: 978-3-030-04921-8, 2019.

[42] H. Nguyen, T. Le, M. Nguyen, P. Fournier-Viger, S.V. Tseng, B. Vo, Mining frequent weighted utility itemsets in hierarchical quantitative databases, Knowl.-Based Syst. 237 (2022) 107709.

[43] L.T.T. Nguyen, V.V. Vu, H.M.T. Lam, M.T.T. Duong, T.L. Manh, T.T.T. Nguyen, B. Vo, H. Fujita, An efficient method for mining high utility closed itemsets, Inform. Sci. 495 (2019) 78–99.

[44] L.T.T. Nguyen, P. Nguyen, T.D.D. Nguyen, B. Vo, P. Fournier-Viger, V.S. Tseng, Mining high-utility itemsets in dynamic profit databases, Knowl.-Based Syst. 175 (1) (2019) 130–144.

[45] N.T. Tung, L.T.T. Nguyen, T.D.D. Nguyen, B. Vo, An efficient method for mining multi-level high utility itemsets, Appl. Intell. (2021) http://dx.doi.org/10.1007/s10489-021-02681-z.

[46] T.N. Tung, L.T.T. Nguyen, T.D.D. Nguyen, P. Fourier-Viger, T.N. Nguyen, B. Vo, Efficient mining of cross-level high-utility itemsets in taxonomy quantitative databases, Inform. Sci. 587 (2022) 1–62.

[47] T. Mai, B. Vo, L.T.T. Nguyen, A lattice-based approach for mining high utility association rules, Inform. Sci. 399 (2017) 81–97.

[48] T. Mai, L.T.T. Nguyen, B. Vo, U. Yun, T.P. Hong, Efficient algorithm for mining non-redundant high-utility association rules, Sensors 4 (2020) 1078.

[49] R. Agrawal, T. Imieinski, A.N. Swami, Mining association rules between sets of items in large databases, in: Proceedings of The 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, 1993, pp. 207–216.
[50] L.T.T. Nguyen, T. Mai, B. Vo, High utility association rule mining, in: High-Utility Pattern Mining, Springer, 2019.
[51] Z. Tan, H. Yu, W. Wei, J. Liu, Top-k interesting preference rules mining based on MaxClique, Expert Syst. Appl. 143 (2020) 113043.
[52] W. Gan, C.J. Lin, P. Fournier-Viger, H.C.Z.J. Chao, J. Zhang, Exploiting highly qualified pattern with frequency and weight occupancy, Knowl. Inf. Syst. 56 (1) (2018) 165–196.
[53] C.-M. Chen, L. Chen, W. Gan, L. Qiu, W. Ding, Discovering high utility-occupancy patterns from uncertain data, Inform. Sci. 546 (2021) 1208–1229.
[54] K. Zhang, Y. Zhang, Z. Wang, Frequent pattern mining based on occupation and correlation, in: 2020 IEEE 3rd International Conference on Electronic Information and Communication Technology, ICEICT, 2020, pp. 161–166, http://dx.doi.org/10.1109/ICEICT51264.2020.9334367.
[55] S. Data, K. Mali, U. Ghosh, High occupancy itemset mining with consideration of transaction occupancy, Arab. J. Sci. Eng. 47 (2022) 2061–2075.
[56] H. Kim, T. Ryu, C. Lee, K. H, T. Truong, P. Fournier-Viger, W. Pedrycz, U. Yun, Mining high occupancy patterns to analyze incremental data in intelligent systems, ISA Trans. 131 (2022) 460–475.
[57] P. Fournier-Viger, A. Gomariz, A. Soltani, T. Gueniche, SPMF: Open-source data mining library, J. Mach. Learn. Res. 15 (1) 3389–3393.