Future Generation Computer Systems 102 (2020) 222–229

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Mining high occupancy itemsets

Zhi-Hong Deng

Key Laboratory of Machine Perception (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China

HIGHLIGHTS

- We present a new task called high occupancy mining.
- We propose a new algorithm to discovery high occupancy itemsets.
- Extensive experiments show that our proposed approach is efficient and effective.

ARTICLE INFO

Article history:
Received 18 February 2017
Received in revised form 14 February 2019
Accepted 16 July 2019
Available online xxxx

Keywords:
Data mining
High occupancy itemset
Algorithm

ABSTRACT

Frequent itemset mining has been extensively studied in data mining for over the last two decades because of its numerous applications. However, the classic support-based mining framework used by most previous studies is not suitable for some real-world applications, such as the travel landscapes recommendation, where *occupancy* besides *support* also plays a key role in evaluating the interestingness of an itemset. In this paper, we propose a new kind of tasks based on *occupancy*, namely high occupancy mining, by introducing *occupancy* into the support-based mining framework. An efficient algorithm, HEP (abbreviation for High Efficient algorithm for mining high occupancy itemsets), is developed to discover all high occupancy itemsets. HEP use a structure, named occupancy-list, to store the occupancy information about an itemset and employs an iterative level-wise approach to mine high occupancy itemset via a pruning strategy based on upper bound of occupancy. Substantial experiments on both synthetic and real datasets show that HEP is efficient for mining high occupancy itemsets and is at least one order of magnitude faster than the baseline algorithm.

1. Introduction

Frequent itemset (or pattern) mining is one of the most intensively investigated problems in terms of computational and algorithmic development [1]. Numerous algorithms [2–14] have been proposed to solve this problem for over the last two decades. Most of those algorithms are based on the support-based mining framework, where *support*, the number of times an itemset appears in a transaction database, is the key measure for itemset interestingness.

In 2012 year, Tang et al. [15] first pointed out that besides the frequent itemset, whose support is no less than a user-specified threshold, one also prefers the itemset that occupies a large portion of the transactions where it appears in some real-world applications, such as the travel landscapes recommendation, print-area recommendation and investment portfolio recommendation. Motivated by the real-application requirement, they propose a new measure, *average occupancy*, which is the average ratio that an itemset occupies the items in the transactions containing the itemset. By considering both *support* and *average occupancy*, they define qualified itemset as follows: For a given minimum support threshold α and a minimum occupancy threshold β, itemset X is said to be *qualified* if its support is no less than α and its average occupancy is no less than β. In recent years, there are a lot of works focus on occupancy, such as the works of Zhang et al. [16] and Gan et al. [17].

However, it is difficult for users to come up with an appropriate pair of α and β synchronously since an improper pair of α and β may cause either too many qualified itemsets or no answers. This issue motivates us to explore a new measure which has the features as follows: (1) The itemset should have a relatively large occupancy and support if the proposed measure of the itemset is large; (2) The proposed measure needs only one threshold. Certainly, only *support* or *average occupancy* defined by Tang et al. cannot satisfy these features. Consider the database in Table 1. If only *support* is used as the measure, the support of itemset $\{c\}$ is 4, which is large since it is 66.7% of the number of all transaction. But its average occupancy is just 0.28 ($= \frac{1}{4} \times (\frac{1}{3} + \frac{1}{4} + \frac{1}{3} + \frac{1}{5})$), which is not big enough. If only *average occupancy* is used as the measure, the average occupancy of itemset $\{a, d\}$ is 0.68 ($= \frac{1}{4} \times (\frac{2}{3} + \frac{2}{3} + \frac{2}{2} + \frac{2}{5})$) while that of itemset $\{a, b, c, d, e\}$ is 1($= \frac{1}{1} \times \frac{5}{5}$).

E-mail addresses: denny@cis.pku.edu.cn, zhdeng@pku.edu.cn.

https://doi.org/10.1016/j.future.2019.07.039
0167-739X/

Table 1
An example database.

TID	Length	Items
T_1	3	a, c, d
T_2	3	a, b, d
T_3	4	b, c, d, e
T_4	2	a, d
T_5	3	c, d, e
T_6	5	a, b, c, d, e

Transaction
Các transactions có trùng phần tử hay không? Ví dụ: T1={A,B}, T2={A,B}
Transaction database DB

However, compared with itemset $\{a, d\}$, itemset $\{a, b, c, d, e\}$ is more undesirable since its support is as low as 1, which means it may be an accidental.

After thorough investigation and research, we find that a desirable measure is the sum of the ratios that an itemset occupies all the items in the transactions containing the itemset. For example, according to this definition, the measures of itemset $\{c\}$, $\{a, d\}$, and $\{a, b, c, d, e\}$ are 1.12 $(= \frac{1}{3} + \frac{1}{4} + \frac{1}{3} + \frac{1}{5})$, 2.93 $(= \frac{2}{3} + \frac{2}{3} + \frac{2}{2} + \frac{2}{5})$, and 1$(= \frac{5}{5})$. Itemset $\{a, d\}$ gains a higher interestingness value than itemset $\{c\}$, and $\{a, b, c, d, e\}$. This is desirable since, compared with itemset $\{c\}$, and $\{a, b, c, d, e\}$, the support and occupancy of itemset $\{a, d\}$ are both relatively large.

Based on the above finding, we define *occupancy* and formulate the problem of mining high occupancy itemset. However, the problem is very challenging since the search space grows exponentially with the number of items. For example, for a database with n items, there are totally $2^n - 1$ itemsets, each of which may be a high occupancy itemset. Even worse, the *downward closure* property [18] does not hold on occupancy. That is to say, the value of occupancy does not increase or decrease monotonically when one adds more items to an itemset. Since all algorithms for mining frequent itemsets are based on the *downward closure* property, they cannot be used to directly mine high occupancy itemsets. In addition, since sum is totally different from average, our *occupancy* does not share the properties held by the *average occupancy* and *quality* defined in [15,16]. Therefore, one also cannot use the algorithms proposed in [15,16] to solve the problem. Thus, we propose a new algorithm called HEP (mining High occupancy itemsets with Efficient Pruning strategy) for high occupancy itemset mining. HEP achieves high efficiency by employing the upper bound properties on occupancy to narrow the search space and using the occupancy-list structure to decrease the computational cost for occupancy computation. The contributions of our work can be summarized as follows:

(1) We proposed a new kind of mining tasks based on *occupancy*, high occupancy itemset mining, by introducing a new interestingness measure for evaluating itemsets. As far as we know, no such mining task has been explicitly proposed or studied.
(2) We propose an algorithm called HEP to solve the problem of high occupancy itemset mining. To achieve high efficiency, HEP employs the upper bound of occupancy, which is first found by us, as the pruning criterion to narrow the search space in the process of mining high occupancy itemsets.
(3) We conduct extensive experiments on various datasets to evaluate the performance of HEP. Experimental results demonstrate that HEP significantly outperforms the baseline method in terms of efficiency.

The rest of the paper is organized as follows. Section 2 formulates the problem of high occupancy itemset mining. Section 3 proposes our methods. Experimental results are presented in Section 4. Section 5 summarizes our work and outlines directions for future research.

2. Problem definition

In this section we first give some preliminaries and propose the definitions of related concepts. Then, we present the formal description of the problem of high occupancy itemset mining. Let $I = \{i_1, i_2, , i_m\}$ be the complete set of distinct items, and a transaction database $DB = \{T_1, T_2, , T_n\}$, where each $T_k (1 \le k \le n)$ is a transaction which contains a set of items in I. Any non-empty set of items is called an itemset and an itemset containing k items is also called k-itemset. Given an itemset P, a transaction T is said to contain P if and only if P is a subset of T. That is, $P \subseteq T$. The preliminary concepts are defined as follows.

Definition 1 (*Support Transaction Set*)**.** The supporting transaction set of an itemset P, denoted as $STSet(P)$, is defined as the set of transactions which contain P.

Definition 2 (*Occupancy*)**.** The *occupancy* of itemset P, denoted as $O(P)$, is defined as

$$O(P) = \sum_{T \in STSet(P)} \frac{|P|}{|T|}, \quad (1)$$

where $|.|$ is the cardinality of set.

Definition 3 (*High Occupancy Itemset*)**.** Given a minimum occupancy threshold ξ, an itemset P is called a high occupancy itemset if its occupancy, $O(P)$, is no less than ξ. $O(P) \ge \xi$

For better understanding the above concepts, Let us take the transaction database in Table 1 as an example. the occupancies of itemset $\{e\}$ and $\{c, e\}$ can be calculated as follows. The supporting transaction set of $\{e\}$ is $\{T_3, T_5, T_6\}$ while that of $\{c, e\}$ is also $\{T_3, T_5, T_6\}$. Therefore, according to Definition 2, the occupancy of $\{e\}$ is 0.783 $(= \frac{1}{4} + \frac{1}{3} + \frac{1}{5})$ and the occupancy of $\{c, e\}$ is 1.567$(= \frac{2}{4} + \frac{2}{3} + \frac{2}{5})$.

With the above definitions, we formulate the problem of mining high occupancy itemsets as follows.

Problem 1 (*Mining High Occupancy Itemsets*)**.** 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 whose occupancies are no less than ξ. $O(P) \ge \xi$

A key characteristic of high occupancy itemsets is that they do not have the downward closure property, which states that if an itemset is frequent, all its subsets must also be frequent. For example, if threshold ξ is set to 1, $\{c, e\}$ is a high occupancy itemset while $\{e\}$ is not. That is, for a high occupancy itemset, its subsets are not necessarily high occupancy itemsets. Although many algorithms have been proposed to efficiently mine frequent itemsets, they cannot be used to directly mine occupancy itemsets since they are all based on the downward closure property. Similarly, our *occupancy* does not share the properties held by the interestingness measures defined in [15,16], the algorithms proposed in [15,16] is also unable to solve the problem of mining high occupancy itemsets. Therefore, it is necessary to design new algorithms for high occupancy itemset mining.

A naive method to address this mining task is to enumerate all itemsets and calculate their occupancies from databases. Obviously, this method suffers from the problem of a search space with exponential growth. Hence, how to design efficient pruning strategies to narrow the search space is a crucial challenge in high occupancy itemset mining. In the next section, we will present our methods.

sơ khảo
tính có
tập P chứa trong tập T
absolute???
bản chất
n=3
2^3-1=7
đơn điệu
loại bỏ
$\xi = 1$
ngây thơ
liệt kê
cam chịu
mặc dù tập {c,e} là tập có tầng suất xuất hiện cao O({c,e})=1.567 >= 1 như tập con của nó là {e} không phải tập có tầng suất xuất hiện cao O({e}) = 0.783 < 1

3. Proposed methods

In this section, we present two algorithms to discover all high occupancy itemsets from a database. The first one is a straightforward solution which first generates all frequent itemsets, calculates the occupancy of each frequent itemset , and then find the high occupancy itemsets. The second one uses upper bound of occupancy as the pruning criterion to directly find the occupancy itemsets in the search space.

3.1. Baseline algorithm

Definition 4 (*Support*)**.** The support of itemset P, denoted as $Sup(P)$, is defined as the number of transactions containing P in DB.

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

Proof. According to Definition 1, for any $T(\in STSet(P))$, we have $|P| \leq |T|$. Combining this conclusion with Definition 2, we have

$$O(P) = \sum_{T \in STSet(P)} \frac{|P|}{|T|} \leq \sum_{T \in STSet(P)} 1 \tag{2}$$

According to Definition 4, We have

$$Sup(P) = |STSet(P)| = \sum_{T \in STSet(P)} 1. \tag{3}$$

Therefore, $O(P) \leq Sup(P)$ holds. □

Definition 5 (*Frequent Itemsets*)**.** Given minimum support threshold ζ, itemset P is called a frequent itemset if $Sup(P)$ is no less than ζ.

Based on Theorem 1 and Definition 5, we infer the following corollary.

Corollary 1. *If the minimum support threshold* ζ *is ~~equal to~~ the minimum occupancy threshold* ξ*, a high occupancy itemset must be a frequent itemset.* $O(P) \geq \xi$

According to Corollary 1, high occupancy itemsets can be mined as follows. First, by taking the minimum occupancy as the minimum support threshold, one can apply the existing algorithms to mine frequent itemsets. Then, high occupancy itemsets can be identified from these frequent itemsets. This is the framework of our baseline algorithm.

However, as observed in our experiments, the baseline algorithm is often confronted of the problem of generating a very large number of frequent itemsets. When the number of frequent itemsets is huge, the baseline algorithm becomes very inefficient since it must consume a large amount of running time to calculate the occupancies of these frequent itemsets. The key point is that support is so loose that it usually leads to a large number of frequent itemsets. Thus, it is necessary to explore proper upper bound of occupancy to enable the design of efficient algorithms. In the next subsection, we will first present a tight upper bound of occupancy, and then build an effective algorithm.

3.2. Algorithm based on pruning strategy

In the section, we first introduce an occupancy-list structure which maintains the information relevant to the occupancy of an itemset. Then, we present an upper bound of occupancy. Finally, taking the upper bound as pruning criterion, we build an efficient algorithm for mining high occupancy itemsets.

{a}'s Occupancy-list : {(T_1, 3), (T_2, 3), (T_4, 2), (T_6, 5)}
{b}'s Occupancy-list : {(T_2, 3), (T_3, 4), (T_6, 5)}
{c}'s Occupancy-list : {(T_1, 3), (T_3, 4), (T_5, 3), (T_6, 5)}
{d}'s Occupancy-list : {(T_1, 3), (T_2, 3), (T_3, 4), (T_4, 2), (T_5, 3), (T_6, 5)}
{e}'s Occupancy-list : {(T_3, 4), (T_5, 3), (T_6, 5)}

Fig. 1. Occupancy-lists of all 1-itemsets.

{ea}'s Occupancy-list : {(T_6, 5)}
{eb}'s Occupancy-list : {(T_3, 4), (T_6, 5)}
{ec}'s Occupancy-list : {(T_3, 4), (T_5, 3), (T_6, 5)}
{ed}'s Occupancy-list : {(T_3, 4), (T_5, 3), (T_6, 5)}

Fig. 2. Occupancy-lists of 2-itemsets containing item e.

3.2.1. Occupancy-lists

Definition 6 (*Occupancy-lists*)**.** The occupancy-list of itemset P is a list structure where each element contains two fields defined as follows:
(1) Field tid indicates a transaction, T, containing P;
(2) Field tsize indicates the length of T.

Let us take itemset {a} as an example to illustrate how to construct the occupancy-lists of 1-itemsets. Since T_1 contains {a}, element (T_1, 3), where 3 is the length of T_1, is in the occupancy-list of {a}. In the same way, element (T_2, 3), (T_4, 2), and (T_6, 5) are also in the occupancy-list of {a}. Fig. 1 shows the occupancy-lists of all 1-itemsets.

Without database scan, the occupancy-lists of 2-itemsets can be constructed by the intersection of the occupancy-lists of 1-itemsets. For example, to construct the occupancy-list of itemset {e, a}, we intersect the occupancy-list of itemset {e}, i.e., {(T_3, 4), (T_5, 3), (T_6, 5)}, and that of itemset {a}, i.e., {(T_1, 3), (T_2, 3), (T_4, 2), (T_6, 5)}, which results in {(T_6, 5)}. As one can observe from the database shown in Table 1, only T_6 contains itemset {e, a}. That is, {(T_6, 5)} is the occupancy-list of {e, a}. Fig. 2 shows the occupancy-lists of all 2-itemsets containing item e.

Based on set theory, the occupancy-lists of itemsets hold the following property.

Theorem 2. *Let P,* P_1*, and* P_2 *be three itemsets and their occupancy-lists are denoted as* OL_P*,* OL_{P1}*, and* OL_{P2} *respectively. If* $P = P_1 \cup P_2$*, then* $OL_P = OL_{P1} \cap OL_{P2}$.

Based on Theorem 2, we can construct the occupancy-list of a k-itemset by intersecting the occupancy-lists of two $(k-1)$-itemsets. Since all tids in the occupancy-list of a 1-itemset are ordered, the occupancy-list of a 2-itemset can be constructed by 2-way comparison with at most $(m+n)$ comparisons, where m and n are the lengths of the occupancy-lists of the corresponding 1-itemsets. Naturally, all tids in the occupancy-list of a 2-itemset are ordered. Therefore, by the same way, we can construct the occupancy-lists of 3-itemsets by intersecting the occupancy of 2-itemsets, and so on.

3.2.2. Upper bound of occupancy

The key problem of mining high occupancy itemsets is that the search space grows exponentially. To reduce the search space, we can exploit the properties of itemset occupancy. According to Definition 6 and Theorem 2, for a superset of itemset P, the occupancy-list of the superset is contained in P's occupancy-list. This property provides the key information about whether some itemsets should be pruned or not.

Definition 7 (*Upper-bound Occupancy*)**.** Assume the transactions containing itemset P have u different lengths. Further, we denote the number of transactions with length of l_x by n_x $(1 \le x \le u)$, and denote $\sum_{i=x}^{u} n_i \times \frac{l_x}{l_i}$ by UBO_P^x. the upper-bound occupancy of P, denoted as $UBO(P)$, is defined as:

$$UBO(P) = \max_{1 \le x \le u} UBO_P^x. \quad (4)$$

u biểu diễn số lượng độ dài 'phân biệt của transaction
lx biểu diễn độ dài của từng transaction
nx biểu diễn số lượng các transaction có cùng độ dài

Theorem 3. *For itemset P, $UBO(P) \le Sup(P)$ holds.*

Proof. Assume the transactions containing itemset P have u different lengths. For any x $(1 \le x \le u)$, we have

Số lượng độ dài phân biệt của transaction chứa P

$$UBO_P^x = \sum_{i=x}^{u} n_i \times \frac{l_x}{l_i} \le \sum_{i=x}^{u} n_i \times 1$$

Độ dài của transaction chứa P đã sắp xếp tăng dần

$$\le \sum_{i=1}^{u} n_i \times 1 = Sup(P). \quad (5)$$

According to Definition 7,

$$UBO(P) = \max_{1 \le x \le u} UBO_P^x \le \max_{1 \le x \le u} Sup(P) = Sup(P). \quad \square \quad (6)$$

Theorem 4. *For any superset of itemset P, denoted as P', $O(P') \le UBO(P)$ holds.*

Proof. Without loss of generation, assume $l_1 \le l_2 \le \cdots \le l_u$. We use n_x to denote the number of transactions with length of l_x. In terms of size, $|P'|$, the length of P', belongs to one of the following cases:

(1) $|P'| < l_1$;
(2) $\exists x (1 \le x \le u), |P'| = l_x$;
(3) $\exists x (1 \le x \le u-1), l_x < |P'| < l_{x+1}$;
(4) $|P'| > l_u$.

We then prove that $O(P') \le UBO(P)$ holds for each case. Note that, since $P \subseteq P'$, we have $STSet(P) \supseteq STSet(P')$.

Case(1). If $|P'| < l_1$, we have

Độ dài của tập cha chứa P

$$O(P') = \sum_{T \in STSet(P')} \frac{|P'|}{|T|} \le \sum_{T \in STSet(P)} \frac{|P'|}{|T|} \le \sum_{T \in STSet(P)} \frac{l_1}{|T|} = \sum_{i=1}^{u} n_i \times \frac{l_1}{l_i} = UBO_P^1 \le UBO(P). \quad (7)$$

Case(2). If $\exists x (1 \le x \le u), |P'| = l_x$, only transactions whose lengths are no less than l_x in $STSet(P)$ may contain P'. That is, $STSet(P') \subseteq \{A | A \in STSet(P) \wedge |A| \ge l_x\}$. Thus, we have

phép toán quan hệ AND
conjunction = Liên hợp

$$O(P_1) = \sum_{T \in STSet(P')} \frac{|P'|}{|T|} \le \sum_{T \in \{A | A \in STSet(P) \wedge |A| \ge l_x\}} \frac{|P'|}{|T|} = \sum_{i=x}^{u} n_i \times \frac{l_x}{l_i} = UBO_P^x \le UBO(P). \quad (8)$$

Case(3). If $\exists x (1 \le x \le u-1), l_x < |P'| < l_{x+1}$, only transactions whose lengths are no less than l_{x+1} in $STSet(P)$ may contain P'. That is, $|STSet(P')| \le \sum_{i=x+1}^{u} n_i$. Similar to Case (2), we have $O(P') \le UBO_P^{x+1} \le UBO(P)$.

Case(4). If $|P'| > l_u$, no transaction can contain itemset P', which means $O(P') = 0$. Therefore, we have $O(P') \le UBO(P)$. $\square$

Based on Theorem 4, we directly infer the following corollary, which is the key pruning strategy of our algorithm presented in the next subsection.

Corollary 2. *Given itemset P, if $UBO(P)$ is less than the user-specified minimum occupancy threshold, any superset of P cannot be a high occupancy itemset.*

3.2.3. HEP algorithm

The HEP algorithm employs an iterative level-wise approach, where $(k-1)$-itemsets are used to explore k-itemsets, to discover all high occupancy itemsets. Algorithm 1 shows the pseudo-code of HEP.

Line 1 of HEP generates the occupancy-lists of all 1-itemsets. In Line 3 to 8, C_1, the set of all candidate 1-itemsets, and HO_1, the set of all high occupancy 1-itemsets, are generated. Note that HEP uses the support of 1-itemset to filter unpromising 1-itemsets in Line 4. Since these unpromising 1-itemsets are not frequent itemsets, they and their supersets cannot be high occupancy itemsets according to *downward closure* property and Corollary 1. Next, in Line 9 to 19, C_{k-1} is used to generate C_k and thus C_k is used to generate HO_k, where C_k denotes the set of candidates with length of k and HO_k denotes the set of high occupancy itemsets with length of k. In Line 15, candidate k-itemset P is generated by combining two $(k-1)$-itemsets sharing $k-2$ items. Sequentially, the occupancy-list of P is constructed by intersecting the occupancy-lists of corresponding $(k-1)$-itemsets in Line 16.

Algorithm 1 : HEP

Input: database DB and minimum occupancy threshold ξ
Output: HO_Set

```
1: Scan DB to obtain the occupancy-list of each 1-itemset;
2: C_1 ← ∅, HO_1 ← ∅;
3: for each 1-itemset P do
4:    if Sup(P) ≥ ξ then
5:       if UBO(P) ≥ ξ then
6:          C_1 ← C_1 ∪ {P};
7:       if O(P) ≥ ξ then
8:          HO_1 ← HO_1 ∪ {P};
9: for (k = 2; C_{k-1} ≠ ∅; k++) do
10:   C_k ← ∅;
11:   for each itemset P_1 ∈ C_{k-1} do
12:      for each itemset P_2 ∈ C_{k-1} do
13:         if |P_1 ∩ P_2| = k − 2 then
14:            P ← P_1 ∪ P_2;
15:            P.occupancy − list ← P_1.occupancy − list ∩ P_2.Occupancy − list;
16:            Scan P.occupancy − list to calculate UBO(P);
17:            if UBO(P) ≥ ξ then
18:               C_k ← C_k ∪ {P};
19:   HO_k ← {P | P ∈ C_k ∧ O(P) ≥ ξ};
20: Return HO_Set ← ∪_k HO_k;
```

thuận lợi

To facilitate the mining process, we implement the HEP algorithm by adopting two techniques as follows. First, Database DB is sorted in transaction-length-ascending order and the occupancy-lists of 1-itemsets are generated by scanning the sorted database. Since the elements in the occupancy-list of a 1-itemset are sorted in transaction-length-ascending order, the elements in the occupancy-list of a $k(\ge 2)$-itemsets are also sorted in such order. The sorted occupancy-lists provide that UBO_P^x can be promptly computed during scanning the lists instead of after scanning the lists. Thus, once some UBO_P^x is no less threshold ξ, one can ensure

nhanh chóng

Table 2
An example database.

Dataset	#Trans	#Items	AvgLen	MaxLen
Mushroom	8,124	119	23	23
Retail	88,162	16 470	10.3	76
Accidents	340,183	468	33.8	51
Kosarak	990,002	41 270	8.1	2498
T10I4D100K	100,000	870	10.1	29
T40I10D100K	100,000	942	39.6	77

that $UBO(P)$ is no less than ξ without scanning the rest of lists. This helps to save the time for evaluating $UBO(P) \geq \xi$ in Line 5 and 17. Second, All the items in an itemset are sorted in item-support-ascending order, which helps to generate k-itemsets by intersecting $(k-1)$-itemsets.

4. Experimental evaluation

In this section we present the performance evaluation of the baseline algorithm and HEP over six various datasets. These two algorithms were implemented in C++. All experiments were performed on a desktop computer with Intel i5 CPU of 3.20 GHz, 8 GB memory and Ubuntu operating system.

Since FP-Growth [3] is the state-of-the-art algorithm for mining frequent itemsets [19,20], it is used as the baseline in our experiment. We first use FP-Growth to find all frequent itemsets and then select high occupancy itemsets from them. Note that, to achieve high performance, itemsets' occupancies are also calculated by occupancy-lists instead of highly time-consuming database scan in the baseline.

4.1. Datasets

Six datasets are used for performance evaluation. These datasets are all downloaded from FIMI Repository.[1] Dataset *mushroom*, *retail*, *accidents*, and *kosarak* are real datasets while dataset $T10I4D100K$ and $T40I10D100K$ are synthetic. Table 2 shows the statistical information about these datasets, including the number of transactions (#Trans), the number of distinct items (#Items), the average number of items in each transaction (AvgLen), and the maximal number of items in the longest transaction (MaxLen).

4.2. The runtime

Fig. 3 shows the time of two algorithms running on all six datasets. Here, runtime means the total execution time, which is the period between input (original databases) and output (all high occupancy itemsets). Once a mining task runs beyond 20,000 s, it will be terminated. The occupancy threshold is the percentage of the size of a dataset. When evaluating runtime, we varied the threshold for each dataset. As the threshold decrease, the number of high occupancy itemsets remarkably increase, and thus the runtime grows. For example, on dataset *mushroom*, when the threshold changes from 12% to 10%, the number of high occupancy itemsets increases from 157 to 2723, and the runtime of HEP grows from 1.35 s to 3.07 s.

In Fig. 3, we observe that HEP is one to several orders of magnitude faster than the baseline algorithm. In addition, HEP also scales better than the baseline algorithm. For dataset *mushroom*, when the threshold is set to 12%, the runtimes of HEP and the baseline algorithm are 1.35 s and 147.73 s respectively. as the threshold goes down, both of them run slowly. However, the runtime of HEP increase slower than that of the baseline algorithm. For example, for threshold 4%, the runtime of HEP is 14.22 s while the runtime of the baseline algorithm is as many as 4961.64 s. By varying the threshold from 12% to 4%, the runtime of HEP increase 10 times while that of the baseline algorithm increase 33 times. On other five datasets, our experimental results still show that the runtime of HEP increases slower than that of the baseline algorithm when the threshold decreases. This finding indicates that HEP has a better scalability than the baseline algorithm.

Table 3
The number of frequent itemsets vs. the number of high occupancy itemsets.

Mushroom	4%	8%	12%
Frequent itemset	4,376,741	658,107	127,053
High occupany itemset	331,704	15,645	157
Retail	0.01%	0.03%	0.05%
Frequent itemset	240,852	38,152	19,242
High occupany itemset	14,202	2,620	1,170
Accidents	10%	20%	30%
Frequent itemset	10,691,549	889,883	149,545
High occupany itemset	5,535	15	0
Kosarak	0.2%	0.6%	1%
Frequent itemset	39,466	1,137	383
High occupany itemset	265	71	31
T10I4D100K	0.02%	0.06%	0.1%
Frequent itemset	119,039	44,583	23,343
High occupany itemset	38,588	12,640	4,363
T40I10D100K	0.3%	0.4%	0.5%
Frequent itemset	5,034,866	1,966,371	1,282,470
High occupany itemset	4,150	17	5

To analyse the reason why HEP is much more efficient than the baseline algorithm, we conduct experiments to record the number of frequent itemsets generated by the baseline algorithm and the number of resultant high occupancy itemsets on each dataset with different thresholds. Table 3 shows the results.

In Table 3, we find that the number of frequent itemsets is one to several orders of magnitude larger than that of high occupancy itemsets on all datasets except dataset $T10I4D100K$. Although the difference is not so remarkable on dataset $T10I4D100K$, the number of frequent itemsets is still three to ten times of that of high occupancy itemsets. Using the occupancy-list structure and the pruning strategy based on upper bound of occupancy, HEP can discover all high occupancy itemsets without generating frequent itemsets. Since HEP avoids the costly occupany computation of huge numbers of frequent itemsets, its efficiency is higher than the baseline algorithm. For example, the baseline algorithm generates 127,053 frequent itemsets from dataset *mushroom* when the threshold is set to 12%. However, the number of resultant high occupancy itemsets is only 157. Although 126,896 (127,053–157) itemsets are finally discarded, the baseline algorithm have to calculate the occupancies of these itemsets on 8,124 transactions.

4.3. Memory consumption

Fig. 4 shows the peak memory consumption of HEP and the baseline algorithm on all six datasets. In the figure, HEP uses much lower amount of memory than the baseline algorithm on dataset *retail*, $T10I4D100K$, and $T40I10D100K$ for all thresholds. On dataset $T40I10D100K$, the memory consumed by HEP is 25% of the memory consumed by the baseline algorithm. The percents are 40% and 43% on dataset *retail* and $T10I4D100K$. However, the baseline algorithm performs better than HEP on *mushroom*, *accidents*, and *kosarak*. Although the baseline algorithm uses less

[1] http://fimi.ua.ac.be/data/.

thay đổi
bỏ đi
độ lớn
HEP cho kết quả runtime tốt hơn baseline. Khi giảm threshold thì runtime cả HEP và baseline đều tăng, tuy nhiên HEP tăng chậm hơn

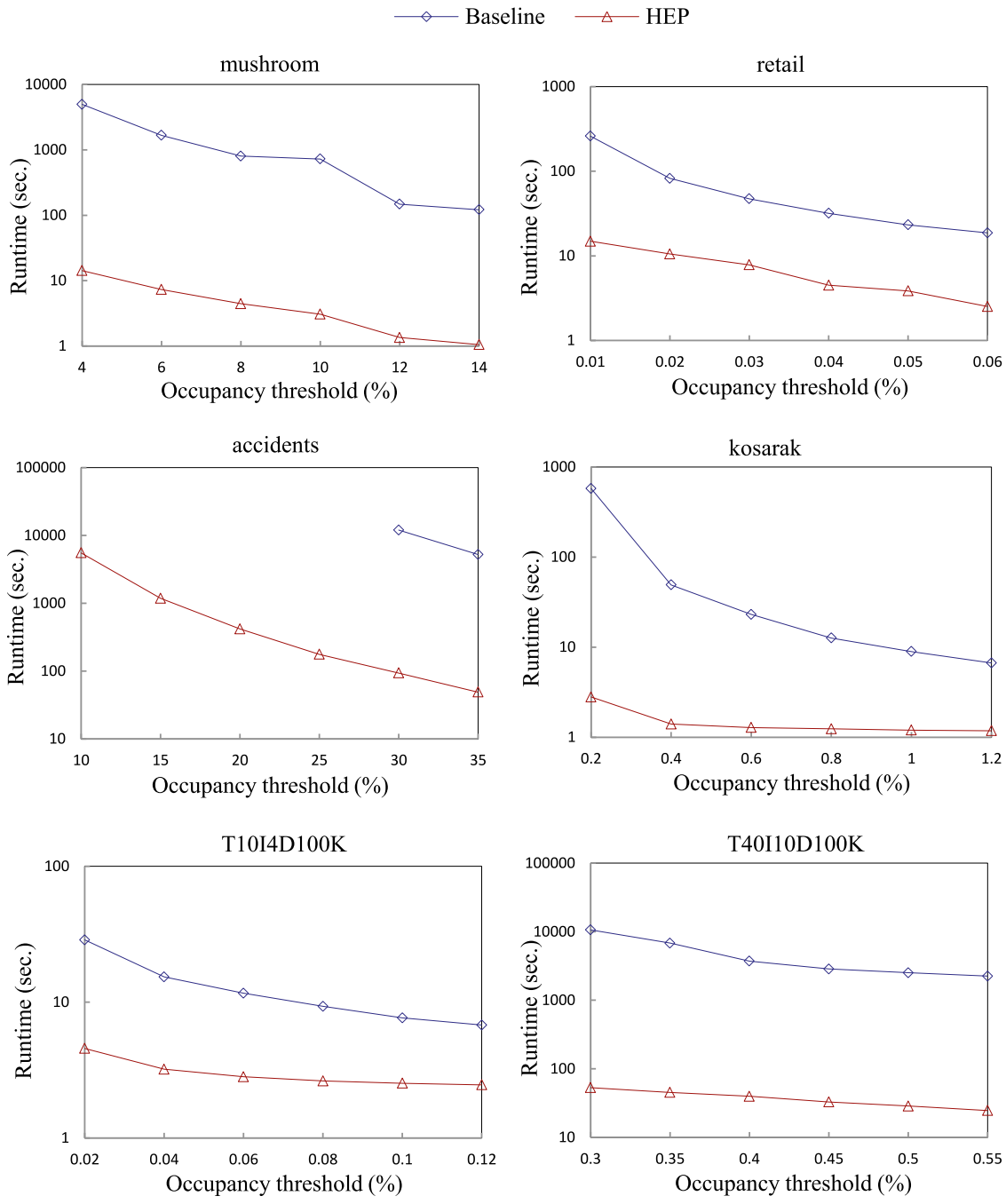

Fig. 3. Runtime comparison on all datasets.

memory than HEP on these datasets, the memory used by the baseline algorithm is at least 53% of the memory used by HEP.

The above observation can be explained as follows. As mentioned in [20], FP-growth used by the baseline algorithm adopts a prefix-tree structure named FP-tree to store the information about frequent itemsets. The FP-tree is suitable for dense datasets. For a dense dataset, its FP-tree is compact and smaller than the dataset. However, when a dataset is sparse, the corresponding FP-tree is usually wide and bushy, and thus relatively large. Based on studies on frequent itemset mining, dataset *mushroom*, *accidents*, and *kosarak* are dense datasets while dataset *retail*, *T*10*I*4*D*100*K*, and *T*40*I*10*D*100*K* are sparse. Therefore, the baseline algorithm performs better on the former than on the latter in terms of memory consumption.

5. Conclusion and future work

In this paper, we have proposed a new mining task, namely high occupancy itemset mining. To solve this problem, we first propose a data structure, occupancy-list, and then explore the upper bound property of occupancy. Finally, we develop an efficient algorithm, HEP, for high occupancy itemset mining. Without generating large numbers of frequent itemsets as the baseline algorithm does, HEP employs the upper bound of occupancy as pruning criterion to directly mine high occupancy itemset, which avoids costly occupancy computation of numerous frequent itemsets. Our experimental study shows that HEP gains significant

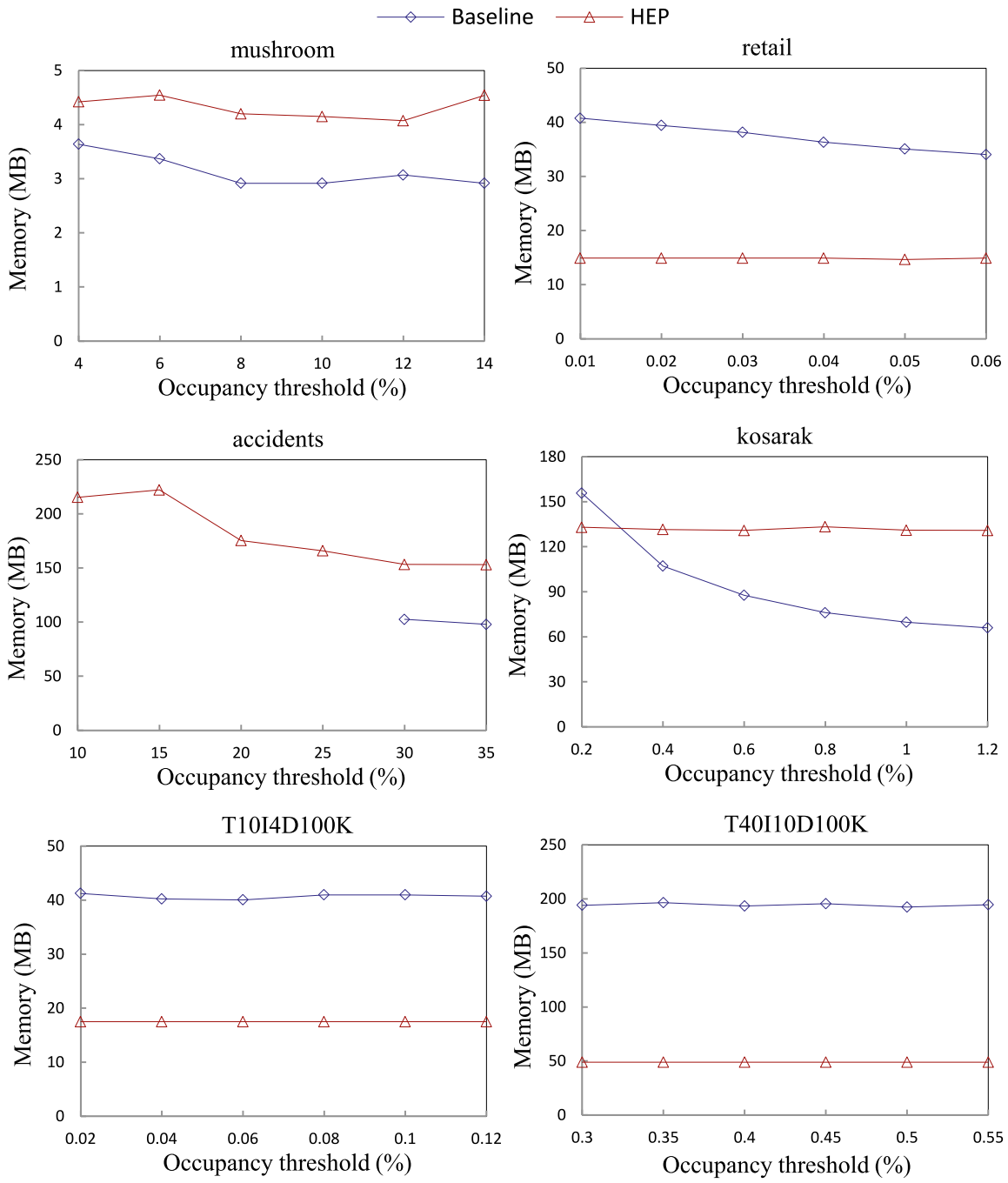

Fig. 4. Memory consumption on all datasets.

improvement over the baseline algorithm in terms of efficiency, and shows a comparable performance on memory consumption.

Since the concept of occupancy can be extended to complex pattern, such as sequential pattern [21,22] and subgraph pattern [23,24], we will focus on this challenging task of mining high occupancy complex patterns in the future.

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.

Acknowledgement

This work is partially supported by the National Natural Science Foundation of China (Grant No. 61170091).

References

[1] C.C. Aggarwal, J. Han (Eds.), Frequent Pattern Mining, Springer, 2014.

[2] R. Agrawal, T. Imielinski, 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, D.C., May 26-28, 1993, 1993, pp. 207–216.

[3] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA, 2000, pp. 1–12.
[4] M.J. Zaki, K. Gouda, Fast vertical mining using diffsets, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003, 2003, pp. 326–335.
[5] G. Grahne, J. Zhu, Fast algorithms for frequent itemset mining using fp-trees, IEEE Trans. Knowl. Data Eng. 17 (10) (2005) 1347–1362.
[6] C.K. Chui, B. Kao, E. Hung, Mining frequent itemsets from uncertain data, in: Advances in Knowledge Discovery and Data Mining, 11th Pacific-Asia Conference, PAKDD 2007, Nanjing, China, May 22-25, 2007, Proceedings, 2007, pp. 47–58.
[7] C.C. Aggarwal, Y. Li, J. Wang, J. Wang, Frequent pattern mining with uncertain data, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009, 2009, pp. 29–38.
[8] Z. Deng, Z. Wang, J. Jiang, A new algorithm for fast mining frequent itemsets using n-lists, Sci. China Inf. Sci. 55 (9) (2012) 2008–2030.
[9] C. Zeng, J.F. Naughton, J. Cai, On differentially private frequent itemset mining, PVLDB 6 (1) (2012) 25–36.
[10] B. Négrevergne, A. Dries, T. Guns, S. Nijssen, Dominance programming for itemset mining, in: 2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013, 2013, pp. 557–566.
[11] Z. Deng, S. Lv, Fast mining frequent itemsets using nodesets, Expert Syst. Appl. 41 (10) (2014) 4505–4512.
[12] Z. Deng, S. Lv, Prepost^{+}: An efficient n-lists-based algorithm for mining frequent itemsets via children-parent equivalence pruning, Expert Syst. Appl. 42 (13) (2015) 5424–5432.
[13] T. Le, B. Vo, An n-list-based algorithm for mining frequent closed patterns, Expert Syst. Appl. 42 (19) (2015) 6648–6657.
[14] Z. Deng, Diffnodesets: An efficient structure for fast mining frequent itemsets, Appl. Soft Comput. 41 (2016) 214–223.
[15] L. Tang, L. Zhang, P. Luo, M. Wang, Incorporating occupancy into frequent pattern mining for high quality pattern recommendation, in: 21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012, 2012, pp. 75–84.
[16] L. Zhang, P. Luo, L. Tang, E. Chen, Q. Liu, M. Wang, H. Xiong, Occupancy-based frequent pattern mining*, TKDD 10 (2) (2015) 14.
[17] W. Gan, J.C. Lin, P. Fournier-Viger, H. Chao, Exploiting high utility occupancy patterns, in: Proceedings of the First International Joint Conference, APWeb-WAIM 2017, July 7-9, 2017, Beijing, China, 2017, pp. 239–247.
[18] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, 1994, pp. 487–499.
[19] A. Ceglar, J.F. Roddick, Association mining, ACM Comput. Surv. 38 (2) (2006).
[20] 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.
[21] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan, 1995, pp. 3–14.
[22] C. Mooney, J.F. Roddick, Sequential pattern mining - approaches and algorithms, ACM Comput. Surv. 45 (2) (2013) 19.
[23] X. Yan, J. Han, Gspan: Graph-based substructure pattern mining, in: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan, 2002, pp. 721–724.
[24] C.C. Aggarwal, Y. Li, P.S. Yu, R. Jin, On dense pattern mining in graph streams, PVLDB 3 (1) (2010) 975–984.

Zhi-Hong Deng received the M.Sc. and Ph.D. degrees in computer science from Peking University, Beijing, China, in 2000 and 2003, respectively. He is currently a professor of Department of Machine Intelligence at Peking University, Beijing, China. He has more than 60 papers published in referred journals and international conferences. His research interests include pattern mining, machine learning, and natural language processing.