Jurnal Teknologi

Full Paper

HIGH CANDIDATES GENERATION: A NEW EFFICIENT METHOD FOR MINING SHARE-FREQUENT PATTERNS

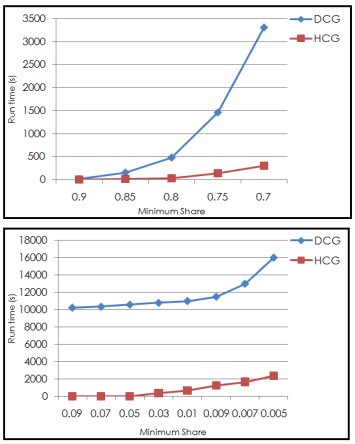
Chayanan Nawapornanan[a*], Sarun Intakosum[a], Veera Boonjing[b]

[a]Department of Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
[b]International College, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

Article history
Received
28 December 2016
Received in revised form
15 June 2017
Accepted
5 September 2017

*Corresponding author
56605008@kmitl.ac.th

Graphical abstract



Abstract

The share frequent patterns mining is more practical than the traditional frequent patternset mining because it can reflect useful knowledge such as total costs and profits of patterns. Mining share-frequent patterns becomes one of the most important research issue in the data mining. However, previous algorithms extract a large number of candidate and spend a lot of time to generate and test a large number of useless candidate in the mining process. This paper proposes a new efficient method for discovering share-frequent patterns. The new method reduces a number of candidates by generating candidates from only high transaction-measure-value patterns. The downward closure property of transaction-measure-value patterns assures correctness of the proposed method. Experimental results on dense and sparse datasets show that the proposed method is very efficient in terms of execution time. Also, it decreases the number of generated useless candidates in the mining process by at least 70%.

Keywords: Data mining, association rule mining, knowledge discovering, share-frequent patterns mining, frequent patterns mining, frequent itemsets mining

1.0 INTRODUCTION

Frequent patterns mining in a transaction database is an important task for knowledge discovery and data mining such as association rules [1], sequential patterns [3, 4, 21] and classification [14, 19]. Its goal is to find the complete set of patterns that appear with their frequencies in the transaction database above a certain threshold. Numerous methods were proposed to find frequent patterns such as level-wise algorithms [2, 6, 7, 20, 22] and pattern-growth approaches [9, 11, 12, 15]. These methods treated all patterns in a transaction database as a binary (0/1) value. That is, only considering the number of transactions in the database containing desired patterns. However, there could be a transaction with many units of an item. Therefore, discovering only traditional frequent patterns cannot reflect any other implicit factors, such as total costs and profits [13].

In dealing with this problem, Carter *et al*. [5] proposed a new share-frequent pattern mining, which extension of traditional frequent itemsets mining with consideration of the purchased quantities. Several mining methods were proposed for efficiently discovering share-frequent patterns. The Zero pruning (ZP) and the Zero subset pruning (ZSP) proposed by [8, 13] can find all share-frequent patterns using an exhaustive search method. However, both algorithms only prune the generated candidates which have zero local measure value. Some approaches have been proposed to find share-frequent itemsets with infrequent subsets but they cannot extract the complete set of share-frequent patterns such as SIP [8], CAC [10] and IAB [13]. After that, the Fast Share Measure (ShFSM)

method [16] was introduced to solve the weakness of the existing algorithms by using a level-closure property. However, the downward-closure property cannot be kept in this property and the ShFSM still produces too many candidates in each round.

[17] proposed the Direct Candidates Generation algorithm (DCG) to reduce the number of candidates by generating candidates directly without cutting back and joining steps in each iteration. The DCG developed to maintain the property of downward-closure by employing the transaction measure value of a pattern. The authors demonstrate that DCG method outperforms the previous algorithm both on the number of generated candidates and execution time. However, the DCG still extracts a huge number of candidates and consumes a lot of time to generate and test a big number of unwanted candidates in the mining process. This is because it treats all *k*-patterns (patterns with *k* items) as candidates for generating candidates in the later rounds. However, these *k*-patterns could be either high transaction-measure-value patterns or low transaction-measure-value patterns. According to the downward-closure property of transaction-measure-value patterns, we propose to exclude low transaction-measure-value *k*-patterns for generating candidates without losing any share-frequent patternsets.

The organization of this paper is as follows. The next section gives basic definitions. Proposed solution section describes details of our approach. An illustrative example section clarifies the proposed solution. Experimental results section reports performance evaluation of the proposed algorithm. We finally conclude the paper in conclusion section.

2.0 METHODOLOGY

2.1 Basic Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a finite set of items. A subset X of I is called an itemset or a patternset. Let $DB = \{T_1, T_2, .., T_i, .., T_n\}$ be a transaction database where T_i is a subset of I. Associated with each item in a transaction is its quantity sold in the transaction. Table 1 is an example of transaction database with I = {A, B, C, D, E, F, G, H} and $DB = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$.

Table 1 An example transaction database with quantity information

TID	Transaction
T_1	{A:1, B:1, C:1, D:1, G:1, H:1}
T_2	{F:4, H:3}
T_3	{B:4, C:3, D:3}
T_4	{C:4, E:1}
T_5	{B:3, D:2}
T_6	{B:3, C:2, D:1}
T_7	{B:3, C:4, D:1, E:2}
T_8	{A:4, F:1, G:1}
T_9	{C:2, E:1}

Definition 1. The measure value of an item i_p in transaction T_q is the quantity of item i_p in T_q denoted by,

$$mv(i_p, T_q) \tag{1}$$

The measure value of item C in T_1 of Table 1 is $mv(C,T_1) = 1$ which is its associated quantity sold.

Definition 2. The patternset measure value of a patternset X in transaction T_q can be defined as follows,

$$imv(X, T_q) = \sum_{i_p \in X} mv(i_p, T_q) \tag{2}$$

The patternset measure value of a patternset {CD} in transaction T_1 of Table 1 is $imv(\{CD\},T_1) = mv(C,T_1) + mv(D,T_1) = 1+1 = 2$.

Definition 3. The local measure value of a patternset X of a database DB is defined as follows,

$$lmv(X) = \sum_{T_q \in DB_X} \sum_{i_p \in X} imv(i_p, T_q) \tag{3}$$

where DB_X is the set of transaction database containing patternset X.

The local measure value of a patternset {CD} of Table 1 is $lmv(\{CD\}) = imv(\{CD\},T_1) + imv(\{CD\},T_3) + imv(\{CD\},T_6) + imv(\{CD\},T_7) = 2+6+3+5 = 16$.

Definition 4. The total measure value of database is the total measure value of all items in all transactions of a transaction database DB. It is defined by,

$$TMV(DB) = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q) \tag{4}$$

The total measure value of Table 1 database is $TMV(DB) = 58$.

Definition 5. The share value of a patternset X in a database DB is the ratio of the local measure value of X to the transaction measure value of DB. That is,

$$SH(X) = \frac{lmv(X)}{TMV(DB)} \tag{5}$$

The share value of a patternset {CD} in Table 1 database is $SH(\{CD\}) = lmv(\{CD\})/TMV(DB) = 16/58 = 0.27$.

Definition 6. Let δ be a minimum share threshold where $0 \leq \delta \leq 1$. The minimum local measure value of database DB with a threshold δ is defined as follows,

$$\min\_lmv = ceiling(\delta \times TMV(DB)) \tag{6}$$

The minimum local measure value of Table 1 database with $\delta = 0.25$ is $min\_lmv = \text{ceiling}(0.25 \times 58) = 15$.

Definition 7. The transaction measure value of transaction T_q is defined as follows,

$$tmv(T_q) = \sum_{i_p \in T_q} mv(ip, Tq) \quad (7)$$

From Table 1 database, the transaction measure value of T_1 is $tmv(T_1) = 6$.

Definition 8. The transaction measure value of a patternset X of a database DB is defined as follows,

$$tmv(X) = \sum_{X \subseteq T_q \in DB_X} tmv(Tq) \quad (8)$$

where DB_X is the set of transaction database containing patternset X.

The transaction measure value of patternset {F} of Table 1 database is $tmv(\{F\}) = tmv(T_2) + tmv(T_8) = 7 + 6 = 13$.

Lemma 1. The transaction measure value of a patternset X maintains the property of downward closure.

Proof. Let {A} be a share-frequent patternsets patternset in $DB_{\{A\}}$, where $DB_{\{A\}}$ is a set of all transaction database containing {A}. Let {B} be a super pattern of {A}, therefore {B} cannot be present in any transaction that {A} is absent. Thus, refer to definition 8, the maximum transaction measure value of {B} is $tmv(\{A\})$. Therefore, if $tmv(\{A\})$ is less than min_lmv, {B} cannot be a share-frequent patternsets and obviously cannot be a high transaction measure value pattern (including {A}).

By Lemma 1, if a patternset X is not high transaction measure value pattern, its supersets are not high transaction measure value patterns either. From Table 1 database with δ = 0.25, we have $min\_lmv = 15$ and $tmv(\{F\}) = 13 < min\_lmv$. Therefore, all supersets of {F} are not high transaction measure value patterns.

Definition 9. A patternset X of a database DB is a high transaction measure value patternset with respect to a threshold δ, if $tmv(X) \geq min\_lmv$.

2.2 Proposed Solution

In this section, at first, we present an efficient data structure called "a PSTable Knowledge" to maintain an incremental database. After that, the HCG algorithm is presented for mining share-frequent patterns from the PSTable Knowledge and it generates the candidate patterns from only the high transaction measure value patterns.

2.2.1 Maintenance of PSTable Knowledge

The PSTable is a set of patterns with their quantities. It is an improved version of a BitTable structure [18] by aggregating transactions having similar patterns. The maintaining process of the PSTable is described as below.

To maintain PSTable, each transaction from initial database or the newly inserted transaction is loaded and treated as a patternset. After that, it checks the patternset of the transaction whether it appears in the PSTable or not. If it does, the quantity of the transaction (total count) is summed with the new entries to gain new value of total count, and the patternset count is incremented by 1. Otherwise, it is the new patternset and then it will be inserted into the PSTable with the quantity of the transaction and set the patternset count to be 1. All the mentioned steps are repeated for all of the transaction in initial database or the newly inserted transactions. The pseudocode of the algorithm is shown in Algorithm 1 in Figure 1.

```
Algorithm 1 Maintenance the PSTable Knowledge
Input : The PSTable, group of original transaction database (DB) or
the newly inserted transactions (db+)
Output : The updated PSTable
1. For each transaction t in DB or db+
2.      set X be a patternset of t
3.      If X is not in PSTable then
4.          add X to the PSTable
5.          set a Total Count as the total of quantity of X
6.          set a Patternset Count as 1
7.      Else
8.          add total quantity of X to a Total Count
9.          increase a Patternset Count by 1
10.     End If
11.End For
```

Figure 1 Pseudocode for maintaining PSTable Knowledge

Example 1. To illustrate the maintaining process of the PSTable, we use a database shown in Table 1 as an example.

The first transaction T_1 ={A:1, B:1, C:1, D:1, G:1, H:1} is loaded into the algorithm, that is, {A,B,C,D,G,H} is treated as a new patternset. Next, the new patternset is checked in the PSTable. The new patternset is new, it will be added to the PSTable and set the total count as the sum of the quantity of transaction (=6) and also set the value 1 to the patternset count. The algorithm does the same steps to T_5 and the result is illustrated in Figure 2 (a). After that the next transaction T_6 = {B:3, C:2, D:1} is read to the algorithm, the new patternset of T_6 is {B,C,D} and it is examined in the PSTable. The new patternset exists, the total count is concluded with the new entries to gain new value of total count as (10+6 = 16) and also increases by one to the patternset count (1+1 = 2) as shown in Figure 2(b). This process is repeated for all transactions and the final PSTable Knowledge is presented in Figure 2(c).

PID	A	B	C	D	E	F	G	H	Total Count	Patternset Count
P_1	X	X	X	X			X	X	6	1
P_2						X			7	1
P_3		X	X	X					10	1
P_4			X		X				5	1
P_5		X		X					5	1

(a) The result of maintaining algorithm after transaction T1-T5 is added

PID	A	B	C	D	E	F	G	H	Total Count	Patternset Count
P_1	X	X	X	X			X	X	6	1
P_2						X			7	1
P_3		X	X	X					16	2
P_4			X		X				5	1
P_5		X		X					5	1

(b) The result of maintaining algorithm after transaction T6 is added

PID	A	B	C	D	E	F	G	H	Total Count	Patternset Count
P_1	X	X	X	X			X	X	6	1
P_2						X			7	1
P_3		X	X	X					16	2
P_4			X		X				8	2
P_5		X		X					5	1
P_6		X	X	X	X				10	1
P_7	X					X	X		6	1

(c) The result of the maintaining algorithm

Figure 2 PSTable knowledge extracted from Table 1 database

2.2.2 The HCG Algorithm

In this subsection, we present a new efficient algorithm for mining share-frequent patterns from the PSTable Knowledge. It generates the minimized number of candidate from only high transaction measure value patternsets. As mentioned in Lemma 1, the transaction measure value (tmv) contains the downward closure property, therefore, it is used to cutting back the useless candidate patternsets in each iteration. To determine whether the patternset is high or not, the value of *tmv* is calculated (by definition 8) - if the value of *tmv* of any patternsets is larger than or equal to the minimum local measure value (*min_lmv*), it is in the set of high transaction measure value. The details of the HCG algorithm are described below.

Input:
1. The PSTable Knowledge
2. A minimum share threshold δ

Output: High share-frequent patterns, C

Step 1: Calculates the total measure value of database from the PSTable using Eq. (4) denoted by *TMV*. After that, the *min_lmv* is computed by using Eq. (6).

Step 2: Each *1*-patternset from the PSTable *X* is loaded into the HCG algorithm and then calculates the transaction measure value of *X* using Eq. (8) denoted by *tmv*(*X*).

Step 3: Checks whether the value of *tmv*(*X*) is larger than or equal to the *min_lmv*. If *X* satisfies the above condition, put *X* in a set of high transaction measure value (abbreviated as *Htmv*).

Lemma 2. Each *1*-patternset from the PSTable *X* is an infrequent if *tmv*(*X*) <*min_lmv*

Proof. Let *X* be a high transaction measure value in the PSTable. According to definition 8, *tmv*(*X*) must be greater than or equal to *min_lmv*. Therefore, this opposes to the assumption *tmv*(*X*)<*min_lmv*. Thus, a patternset *X* is an infrequent.

In HCG algorithm, we calculate all *Htmv* in the PSTable by determining the original transaction measure value according to Eq. (8) which this value maintains the property of downward closure.

Step 4: Each *1*-patternset in the high transaction measure value (*Htmv*) *X* is read into the HCG algorithm. Then, the algorithm will show every pattern from the $PSTable_X$ where $PSTable_X$ is the set of patterns containing patternset *X*. The transaction measure value of each *1*-patternset in $PSTable_X$ is computed.

Step 5: Checks whether the value of *tmv* of each *1*-patternset in $PSTable_X$ is greater than or equal to the *min_lmv*. If *X* satisfies the above condition, put *X* in a set of high transaction measure value of *X*, $Htmv_X$.

Lemma 3. Each *1*-patternset from the $PSTable_X$ *Y* is an infrequent if *tmv*(*Y*) <*min_lmv*

Proof. Let *Y* be a high transaction measure value in the $PSTable_X$. According to definition 8, *tmv*(*Y*) must be greater than or equal to *min_lmv*. Therefore, this opposes to the assumption of *tmv*(*Y*) <*min_lmv*. Thus, a patternset *Y* is an infrequent.

In our algorithm, we calculate all $Htmv_X$ in the $PSTable_X$ by performing the original transaction measure value calculation according to Eq. (8) which this value maintains the property of downward closure.

Step 6: Mines all 2-patternset in $Htmv_X$ that prefixs with *X* and keep them in the C_2. After that, removes *X* from *Htmv* and sets *k* = 3, where *k* is used for recording the length of generated candidates in the mining process.

Step 6-1: The candidate *k*-patternsets are generated from C_{k-1} and also compute their transaction measure value (*tmv*(*k*-patternsets)) in the $PSTable_X$ for

checking against the *min_lmv*. Checks whether the value of *tmv*(*k*-patternset) is greater than or equal to the *min_lmv*. If it satisfies the above condition, stores it to C_k. Increases *k* by one and this step is then repeated until no candidate patternset is generated.

Step 7: Repeats Steps 4 – 6 until the *Htmv* is empty and returns the *C* that stores all high transaction-measure-value pattern at the end of the HCG algorithm.

2.3 An Illustrative Example

In this section, we use the Table 1 database to illustrate the HCG algorithm step by step. Also, it is assumed that the minimum share threshold is 25% of total quantity.

Input:
1. The PSTable is shown in Figure 2 (c), which constructed from the example transaction database in Table 1.
2. The minimum share threshold (δ) is set as 0.25.

Output: High share-frequent patterns, *C*

Step 1: The HCG algorithm calculates the initial variables at first i.e. *TMV*(=58) and *min_lmv* (58*0.25), which is 15, respectively.

Step 2: The *1*-patternset *X* in the PSTable for {A}, {B}, {C}, {D}, {E}, {F}, {G} and {H} are read into the algorithm and then calculates their transaction measure value (*tmv*(*X*)). Takes *1*-patternset {A} as an example to illustrate the process. The *1*-patternset {A} appears in the PSTable P_1 and P_7 and then the transaction measure value of the *1*-patternset {A} is calculated as (6+6 = 12). The other *1*-itemsets are calculated in the same way and the result is shown in Figure 3.

1-Patternset *X*	*tmv*(*X*)
A	12
B	37
C	40
D	37
E	18
F	13
G	12
H	13

Figure 3 *tmv*s of *1*-patternsets

Step 3: The *tmv* values of the *1*-patternset are checked against the *min_lmv*. In this example, the four item {B}, {C}, {D} and {E} satisfy the condition and put them in both the set of high transaction measure value (*Htmv*) and C_1. Thus, *Htmv* = {{B}, {C}, {D}, {E}}.

Step 4: Each *1*-patternset *X* in the *Htmv* is then read into the algorithm. Takes $Htmv_B$ as an example, the {B} appears in Pattern P_1, P_3, P_5 and P_6 in the PSTable ($PStable_B$). Then, the HCG algorithm computes the *tmv* value of all *1*-patternset in $PStable_B$ as {{B: 6+16+5+10 (=37)}, {C: 6+16+10 (=32)}, {D: 6+16+5+10 (=37)} and {E: 10}}, respectively, is shown in Figure 4 (a).

Step 5: Checks whether the *tmv* value of the *1*-patternset in the $PStable_X$ is larger than or equal to the *min_lmv*. Then, put them in the set of high transaction measure value of *X*, $Htmv_X$. In this example, the *1*-patternsets in the $PStable_B$ satisfy the condition which are as follows {B}, {C} and {D}. Thus, $Htmv_B$ = {B,C,D}. The other $Htmv_X$ are then processed in the same way. The results are shown in Figure 4 (b)-(d) respectively.

PID	B	C	D	E	TotalCount
P_1	X	X	X		6
P_3	X	X	X		16
P_5	X		X		5
P_6	X	X	X	X	10
	37	**32**	**37**	**10**	

(a) The process of {B} in Htmv

PID	C	D	E	TotalCount
P_1	X	X		6
P_3	X	X		16
P_4	X		X	8
P_7	X	X	X	10
	40	**32**	**18**	

(b) The process of {C} in Htmv

PID	D	E	TotalCount
P_1	X		6
P_3	X		16
P_5	X		5
P_6	X	X	10
	37	**10**	

(c) The process of {D} in Htmv

PID	E	TotalCount
P_4	X	8
P_6	X	10
	18	

(d) The process of {E} in Htmv

Figure 4 The process of 1-patterns in *Htmv*

Step 6: Mines all 2-patternset in $Htmv_X$ that prefix with *X* and then removes *X* from *Htmv*. Takes $Htmv_B$ as an example to illustrate the process. The HCG generates candidate 2-patternset of the $Htmv_B$ as {B,C} and

{B,D} respectively, which is shown in Figure 5, and also keeps them in C_2. Then, *k* is set to 3.

Step 6-1: The candidate *k*-patternset are generated from C_{k-1} and also compute their transaction measure value for checking against the *min_lmv*. From the result of candidate 2-patternset of the $Htmv_B$, the HCG generates the candidate *k*-patternset which is {B,C,D} as shown in Figure 6 and it satisfies the condition, saves it to C_k. This sub step of pattern "B" can be terminated in the third round because no candidate patternset is generated.

2-Patternset *X*	*tmv*(*X*)
{BC}	32
{BD}	37

Figure 5 *tmv*s of 2-patternsets with prefix "B"

3-Patternset *X*	*tmv*(*X*)
{BCD}	32

Figure 6 *tmv*s of 3-patternsets with prefix "B"

Step 7: Steps 4 – 6 are then repeated until no member of *Htmv*. After the algorithm works complete, we get *C* = {{B}, {C}, {D}, {E}, {B,C}, {B,D}, {B,C,D}, {C,D}, {C,E}}. The share-frequent patterns mined from *C* are {C}, {B,C}, {B,D}, {C,D} and {B,C,D} respectively. All generated candidate are shown in Figure 7.

No.	Candidate Patterns	tmv(*X*)	lmv(*X*)	Sh(*X*)	Share-frequent patterns
1	B	37	14	0.2413	NO
2	C	40	18	0.2758	YES
3	D	37	7	0.1379	NO
4	E	18	5	0.0689	NO
5	BC	32	21	0.3620	YES
6	BD	37	22	0.3793	YES
7	BCD	32	27	0.4655	YES
8	CD	32	16	0.2758	YES
9	CE	18	14	0.2413	NO

Figure 7 The generated candidates by the HCG algorithm

Observation 1. The DCG algorithm adopts the level-wise candidate generation-and-test methodology. Firstly, it reads all *1*-patternset from a transaction database and treats them as *1*-candidate for generating all the candidate for length 2, that is, all *1*-candidate contain high *tmv* value and low *tmv* value. For example, if the number of atomic patterns is 500 and they are treated as *1*-candidate and then the algorithm tests for $\binom{500}{2}$ 2-patternset candidate patterns. For 2-patternset, a patternset will be considered for selection as a candidate by the definition 8. Therefore, a large number of candidates are generated.

Lemma 4. If M_1 is the number of generated candidates by HCG algorithm and M_2 is the number of generated candidates by DCG algorithm, then $M_1 \subseteq M_2$.

Proof. Let $Z\{z_1,z_2, \ldots ,z_n\}$ be a candidate patternset if all of its subset of length *n*-1 are candidate patternset (high transaction measure value) in the DCG algorithm. So, *Z* could have low *tmv* value to become a candidate of length *1*. In HCG algorithm, if *Z* is a low *tmv* then it cannot appear as a candidate. In addition, HCG prunes *Z* immediately after determining it is low *tmv* value. In conclusion, the generated candidate patternsets of HCG stores only the high transaction measure value, therefore, M_1 is less than or equals to M_2.

3.0 RESULTS AND DISCUSSION

3.1 Experimental Environment and Datasets

In this section, the experimental evaluation of the compared algorithms: the HCG solution and the DCG method [17] are conducted. All algorithms were implemented in Microsoft C# 2012 and running on a 2.60 GHz Intel(R) Core i7-4720HQ with 12 GB main memory on the Windows 8.1 operating system. Similar to the performance evaluation of the previous share-frequent patterns mining algorithm [17], the count information was assigned to each item in each transaction, ranging from 1 to 10. Four datasets are used in the experiments and they are acquired from FIMI Repository Page [23-26]. The characteristics of the datasets are shown in Table 2, |D| is the total number of transaction in a dataset, |I| is the number of distinct pattern in the dataset and T_{avg} is the average size of transaction. The dataset is dense since the number of atomic pattern is small and each transaction has a lot of distinct pattern. By considering the mushroom dataset in Table 2, its atomic pattern is 119 and its transaction size is 23. Therefore, the ratio of its distinct pattern presented in every transaction is 20% ((23/119)*100).

Table 2 Characteristics of the experiment datasets

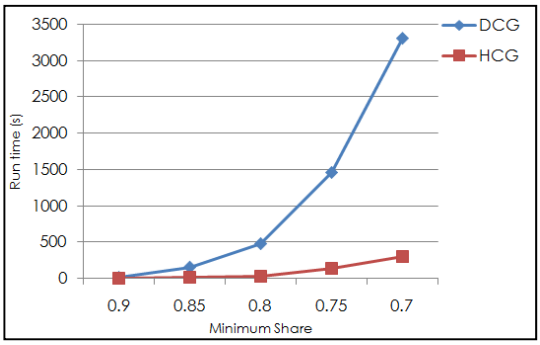
Dataset	|D|	|I|	T_{avg}	Type
Mushroom	8,124	119	23	Dense
Chess	3,196	75	37	Dense
T10I4D100K	100,000	870	10.1	Sparse
T40I10D100K	100,000	942	40.5	Sparse

3.2 Experimental Results for Dense Datasets

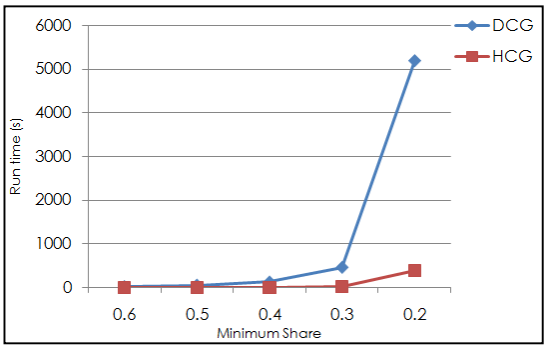
In the first experiment, we compare the running time of the HCG algorithm over the DCG method using the Mushroom and Chess datasets. As there is no share-frequent patternset, two ranges of minShare are chosen. For Mushroom dataset, the minimum share threshold is set from 0.6 to 0.2, while for Chess

dataset, it is set from 0.9 to 0.7. The results are shown in Figure 8 (a) and (b). It is obvious to see that two algorithms require more runtime when the minimum share threshold decreases. This is reasonable because when the threshold value became smaller, more candidates in the mining process are generated.

Moreover, the HCG algorithm performs better than the existing DCG algorithm with respect to all of the minimum share thresholds. The reason is that DCG requires more execution time to generate and test a huge number of useless candidates in the mining process since it treated all *1*-patterns as candidates for generating candidates in the later rounds. That is, *1*-patterns contain both high *tmv* patterns and low *tmv* patterns. For the HCG algorithm, it generates the candidate from only high transaction measure value atomic patterns in the second round. For the third round onwards, it created candidates from the combination of only the patterns which have their transaction measure value beyond a certain threshold. The number of candidate in the mining process and the number of generated candidate of both methods are shown in Table 3 and 4 respectively.



(a) Total running time on Chess



(b) Total running time on Mushroom

Figure 8 Running times of DCG and HCG with different thresholds on dense datasets

Table 3 The number of generated candidates of DCG and HCG on the chess dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.6	8,623	179	162	51
0.5	12,003	331	259	153
0.4	25,296	967	663	565
0.3	90,924	3,744	2,826	2,735
0.2	950,253	58,760	53,697	53,621

Table 4 The number of generated candidates of DCG and HCG on the mushroom dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.90	12,542	955	691	629
0.85	41,797	3,487	2,733	2,674
0.80	116,920	10,256	8,304	8,248
0.75	282,912	25,958	21,018	20,966
0.70	629,379	60,377	48,972	48,921

Table 5 compares the number of candidates in the mining process and the number of generated candidates of two algorithms in each round. A Mushroom dataset was employed and the minimum share threshold was set at 0.3%. In addition, we further found that, in every round of mining process, the HCG method produces a smaller number of candidates than that of the DCG one. The reason to this situation is the same as the content mentioned earlier. The HCG and the DCG methods terminate in the ninth round and tenth round respectively.

Table 5 The number of candidates in the mining process and the number of generated candidates on Mushroom dataset

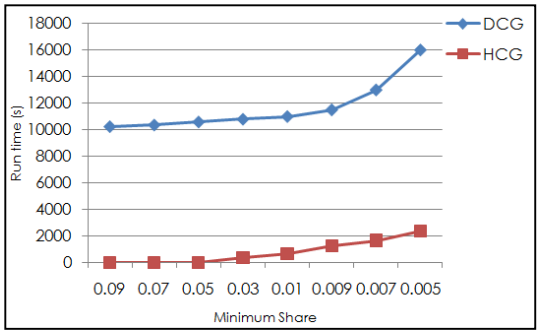
k-patternset	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
k=1	119	119	119	28
k=2	7,021	378	163	163
k=3	7,164	660	455	455
k=4	15,836	956	725	725
k=5	22,179	856	712	712
k=6	20,482	525	441	441
k=7	12,315	203	169	169
k=8	4,653	43	38	38
k=9	1,044	4	4	4
k=10	111	-	0	-
Total	90,924	3,744	2,826	2,735

3.3 Experimental Results for Sparse Datasets

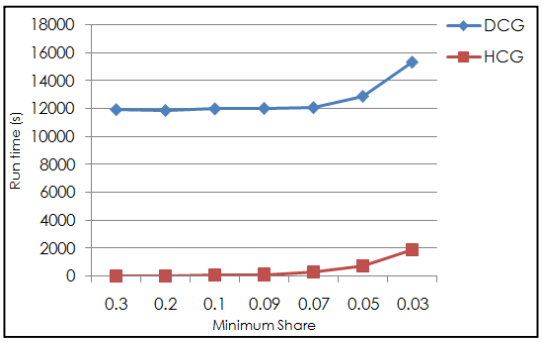
In this experiment, we present the comparison of HCG and DCG algorithms on sparse datasets,

T10I4D100K and T40I10D100K, the results in terms of consumed run time are shown in Figure 9 (a) and (b). Two ranges of min_share are chosen. For T10I4D100K dataset, the minimum share threshold was set from 0.09 to 0.005. While for T40I4D100K dataset, it was set from 0.3 to 0.05, as there is no share-frequent patternset.

It shows that our proposed HCG runs faster than the DCG one since the new proposed HCG can reduce the large number of unpromising candidates in the mining process because the HCG generated the candidate patterns from only the patterns with transaction measure value above a certain threshold.



(a) Total running time on T10I10D100K



(b) Total running time on T40I4D100K

Figure 9 Running times of DCG and HCG with different thresholds on sparse datasets

Table 6 and 7 compares the number of candidates in the mining process and the number of generated candidates of two algorithms. Obviously, from Table 6, the minimum share is set to 0.09, it could be seen that no candidate is larger than a certain threshold. The process of HCG starts to read all *1*-patterns, which is 870 items and then checks whether their values is high and no pattern is above a certain threshold (high), therefore the HCG terminates in the first round. For DCG algorithm, it reads all *1*-patterns and treats them as *1*-candidates (870 items) for generating-and-testing 2-patterns in second round. Then, no pattern for 2-patterns is above a certain threshold, so the DCG terminates in the second round.

Table 6 The number of generated candidates of DCG and HCG on the T10I4D100K dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.30	444,153	942	942	0
0.20	444,153	957	942	6
0.10	444,153	5,037	942	91
0.09	444,153	7,845	942	118
0.07	444,760	20,249	944	199
0.05	452,284	50,491	966	339
0.03	574,350	128,842	1,369	926

Table 7 The number of generated candidates of DCG and HCG on the T40I10D100K dataset

Minimum Share	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
0.090	378,885	870	870	0
0.070	378,885	876	870	4
0.050	378,885	1,023	870	18
0.030	378,885	5,241	870	94
0.010	384,432	85,545	889	431
0.009	397,609	99,694	929	504
0.007	435,750	131,367	1,105	746
0.005	605,839	185,077	1,797	1,532

Similar to Table 5, Table 8 compares the difference of the candidate numbers in the mining process and the generated candidate numbers between both algorithms in each pass. The T10I4D100K dataset was used and *min_share* was set to 0.009. We can observe that the HCG generates lower numbers candidate than that of the DCG method. In addition, the overall performance of HCG is better because it produces the candidate from only high the transaction measure value patterns. Through all of the experimental results, we can observe that the new proposed effectively much better than the existing DCG algorithm in terms of the total running time and the number of generated candidates on both dense and sparse datasets.

Table 8 The number of candidates in the mining process and the number of generated candidates on T40I10D100K dataset

k-patternset	The number of candidates in the mining process		The number of generated candidates	
	DCG	HCG	DCG	HCG
k=1	870	870	870	445
k=2	378,015	98,790	50	50
k=3	14,916	33	9	9
k=4	3,808	1	0	0
Total	397,609	99,694	929	504

4.0 CONCLUSION

This paper presented a new efficient method for mining share-frequent patterns, named HCG, aimed to develop for decreasing the number of unpromising candidate patterns. The HCG generated the candidate pattern from only the patterns which have their transaction measure value beyond a certain threshold in a level-wise way. Moreover, we also presented a new data structure that captures all atomic patterns with their count information, named PSTable, which is constructed once by a single scan database. When new transactions come in, they can suddenly add to the existing database without reconstructing. Extensive performance analysis shows that our approach outperforms the existing DCG algorithm in terms of the total running time and the number of generated candidates on both dense and sparse datasets.

References

[1] Agarwal, R., Imielinski, T. and Swami, A. 1993. Mining Association Rules between Sets of Items in Large Database. *Proceedings of the ACM SIGMOD on Management of Data*. 207-216.

[2] Agarwal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of 20th International Conference on Very Large Data Bases*. 487-499.

[3] Agarwal, R. and Srikant, R. 1995. Mining Sequential Patterns. *Proceedings of 11th International Conference on Data Engineering*. 3-14.

[4] Agarwal, R. and Srikant, R. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. *Proceedings of 5th International Conference on Extending Database Technology*. 3-17.

[5] Carter, C. L., Hamilton, H. J. and Cercone, N. 1997. Share Based Measures for Itemsets. *Lecture Notes in Computer Science*. 1263: 14-24.

[6] Park, J. S., Chen, M. S. and Yu, P. S. 1997. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering*. 9: 813-825.

[7] Brin, S., Motwani, R., Ullman, J. D. and Tsur, S. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *Proceedings of the ACM SIGMOD on Management of Data*. 255-264.

[8] Barber, B. and Hamilton, H. J. 2000. Algorithms for Mining Share Frequent Itemsets Containing Infrequent Subsets. *Lecture Notes in Computer Science*. 1910: 316-324.

[9] Han, J., Pei, J. and Yin, Y. 2000. Mining Frequent Patterns without Candidate Generation. *Proceedings of the ACM SIGMOD on Management of Data*. 1-12.

[10] Barber, B. and Hamilton, H. J. 2001. Parametric Algorithm for Mining Share Frequent Itemsets. *Journal of Intelligent Information Systems*. 16: 277-293.

[11] Pei, J., Han, J. and Lu, H. 2001. Hmine: Hyper-structure Mining of Frequent Patterns in Large Database. *Proceedings of International Conference on Data Mining*. 441-448.

[12] Agarwal, R., Aggarwal, C. and Prasad, V. V. V. 2001. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed*. 61: 350-371.

[13] Barber, B. and Hamilton, H. J. 2003. Extracting Share Frequent Itemsets with Infrequent Subsets. *Data Mining and Knowledge Discovery*. 7: 153-185.

[14] Han, J., Pei, J., Yin, Y. and Shi, C. 2004. Integrating Classification and Association Rule Mining: A Concept Lattice Framework. *Lecture Notes in Computer Science*. 1711: 443-447.

[15] El-Hajj, M. and Zaiane, O. R. 2004. COFI Approach for Mining Frequent Itemsets Revisited. *Proceeding of the ACM SIGMOD on Data Mining and Knowledge Discovery*. 70-75.

[16] Li, Y. C., Yeh, J. S. and Chang, C. C. 2005. A Fast Algorithm for Mining Share-frequent Itemsets. *Lecture Notes in Computer Science*. 3399: 417-428.

[17] Li, Y. C., Yeh, J. S. and Chang, C. C. 2005. Direct Candidates Generation: A Novel Algorithm for Discovering Complete Share-frequent Itemsets. *Lecture Notes in Computer Science*. 3614: 551-560.

[18] Nawapornanan, C and Boonjing, V. 2011. A New Share Frequent Itemsets Mining Using Incremental Bittable Knowledge. *Proceedings of 5th International Conference on Computer Sciences and Convergence Information Technology*. 358-362.

[19] Mohammad, N. Q., Hassan, F. H. A., Yahya, K. T. 2015. An Improved Documents Classification Technique Using Association Rules Mining. *Proceedings of IEEE International Conference on Research in Computational Intelligence and Communication Networks*. 460-465.

[20] Houda, E., Mohamed, E. F. and Mohammed, E. M. 2016. A Novel Approach for Mining Frequent Itemsets: AprioriMin. *Proceedings of 4th IEEE International Colloquium on Information Science and Technology*. 286-289.

[21] Peng, H. 2016. Improved Algorithm Based on Sequential Pattern Mining of Big Data Set. *Proceedings of 7th IEEE International Conference on Software Engineering and Service Science*. 115-118.

[22] Shubhangi, D. P., Ratnadeep, R. D. and D., K. K. 2016. Adaptive Apriori Algorithm for Frequent Itemset Mining. *Proceedings of International Conference System Modeling & Advancement in Research Trends*. 7-13.

[23] Frequent Itemset Mining Dataset Repository, "Chess", http://fimi.ua.ac.be/data/, 1987 (Accessed: June 9, 2017).

[24] Frequent Itemset Mining Dataset Repository, "Mushroom", http://fimi.ua.ac.be/data/, 1989 (Accessed: June 9, 2017

[25] Frequent Itemset Mining Dataset Repository, "T10I4D100K", http://fimi.ua.ac.be/data/, 2003 ((Accessed: June 9, 2017).

[26] Frequent Itemset Mining Dataset Repository, "T40I10D100K", http://fimi.ua.ac.be/data/, 2003 (Accessed: June 9, 2017).