Direct Candidates Generation: A Novel Algorithm for Discovering Complete Share-Frequent Itemsets

Yu-Chiang Li[1], Jieh-Shan Yeh[2], and Chin-Chen Chang[1, 3]

[1] Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi 621 Taiwan
{lyc, ccc}@cs.ccu.edu.tw
[2] Department of Computer Science and Information Management,
Providence University, Taichung 433, Taiwan
jsyeh@pu.edu.tw
[3] Department of Information Engineering and Computer Science,
Feng Chia University, Taichung 407, Taiwan

Abstract. The value of the itemset share is one way of evaluating the magnitude of an itemset. From business perspective, itemset share values reflect more the significance of itemsets for mining association rules in a database. The Share-counted FSM (ShFSM) algorithm is one of the best algorithms which can discover all share-frequent itemsets efficiently. However, ShFSM wastes the computation time on the join and the prune steps of candidate generation in each pass, and generates too many useless candidates. Therefore, this study proposes the Direct Candidates Generation (DCG) algorithm to directly generate candidates without the prune and the join steps in each pass. Moreover, the number of candidates generated by DCG is less than that by ShFSM. Experimental results reveal that the proposed method performs significantly better than ShFSM.

1 Introduction

Data mining techniques have been developed to find new and potentially useful knowledge from data. [11]. Traditional methods for mining association rules are based on the support-confidence framework to discover all relationships among items (each market product is called an *item*) from historical transaction databases.

The support value is applied to measure the importance of itemsets (a group of products bought together in a transaction) in a transaction database. It only reflects the percentage of transactions in which the itemset sold, but neither reveals the profit, the cost nor the real quantity sold of each itemset. For example, in Table 1, the column "Count" indicates the sale volume of each item in each transaction. According to the support value, $\{A\}$ appears in four transactions, but its real sale volume is 12.

Users are usually more interested in knowing which itemsets are bought in sufficient numbers to gain a certain net profit or attain a given cost. To reveal such knowledge, several issues have been proposed, such as share-confidence framework, profit mining and utility itemsets [7, 8, 16]. In 1997, Carter *et al*. first introduced a share-confidence framework [7]. Instead of discovering frequent itemsets, the method with

L. Wang and Y. Jin (Eds.): FSKD 2005, LNAI 3614, pp. 551 – 560, 2005.

share-confidence framework finds *share-frequent* (SH-frequent) itemsets. The share measure can provide valuable information of itemsets' net profit or cost, which the support measure cannot [6].

Table 1. Example of a transaction database with counting

TID	Transaction	Count	Total count
T01	{A, B, C, D, G, H}	{1, 1, 1, 1, 1, 1}	6
T02	{A, C, E, F}	{4, 3, 1, 2}	10
T03	{A, C, E}	{4, 3, 3}	10
T04	{B, C, D, F}	{4, 1, 2, 2}	9
T05	{A, B, D}	{3, 1, 2}	6
T06	{B, C, D}	{3, 2, 1}	6

An SH-frequent itemset usually includes some infrequent subsets even no frequent subset. Obviously, an exhaustive search method can be used to generate all SH-frequents, such as the ZP and the ZSP algorithms [4, 6]. However, the exhaustive search method is time-consuming and does not work efficiently in a large dataset. Some algorithms have been proposed to facilitate the extraction of SH-frequents with infrequent subsets, such as SIP, CAC and IAB [4, 5, 6], but they may not discover all SH-frequent itemsets. Recently, Li *et al*. first developed algorithms to swiftly discover all SH-frequent itemsets [12, 13]. Among these algorithms ShFSM is the best. In contrast to the number of SH-frequent itemsets, the performance bottleneck of ShFSM is generating too many candidates in each pass. Accordingly, this work proposes the Direct Candidate Generation (DCG) method to further improve the performance of ShFSM. Our scheme directly generates candidates in each pass without the join and the prune steps. Furthermore, DCG can efficiently lower the number of useless candidates and further accelerate the mining process. For simplicity and without loss of generality, this study supposes that the measure value of each item in each transaction is a non-negative integer.

The rest of this paper is organized as follows. Section 2 reviews support-confidence and share-confidence frameworks. Section 3 explains the proposed Direct Candidate Generation (DCG) algorithm for discovering all SH-frequent itemsets. Section 4 provides experimental results and evaluates the performance of the proposed algorithm. Finally, we conclude in Section 5 with a summary of our work.

2 Review of Support and Share Measures

2.1 Support-Confidence Framework

Agrawal *et al*. first defined the problem of mining association rules [2, 3]. The formal definition is as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of literals with binary attributes, called items. Let X be an itemset, where $X \subseteq I$. Let the transaction database $DB = \{T_1, T_2, ..., T_z\}$ be the set of transactions, where each transaction $T_q \in DB$, $T_q \subseteq I$, $1 \le q \le z$. The notation $X \Rightarrow Y$ presents an association rule, where $X \subseteq I$, $Y \subseteq I$ and $X \bigcap Y = \phi$. The rule $X \Rightarrow Y$ has *support s%*, denoted as $Sup(X \bigcup Y)$, in the transaction database *DB*

if the itemset $X \bigcup Y$ appears in *s%* of transactions in *DB*. The *confidence* of the rule $X \Rightarrow Y$, denoted as *Conf*($X \Rightarrow Y$), is *c%* if the set of transactions containing *X* in *DB* has *c%* transactions also containing *Y*. Thus, *Conf*($X \Rightarrow Y$) = *Sup*($X \bigcup Y$)/*Sup*(*X*). Given the minimum support (*minSup*) and minimum confidence (*minConf*) threholds, the process of mining association rules is to generate all rules that satisfy the two certain constraints, respectively.

Apriori is a level-wise (including multiple passes) algorithm. In each pass, Apriori employs the downward closure property to efficiently identify all frequent *k*-itemsets (itemsets with length *k* and their supports are greater than or equal to *minSup*). In fact, an arbitrary *k*-subset of a frequent (*k*+1)-itemset is also frequent; otherwise, the (*k*+1)-itemset is infrequent. This characteristic is called the downward closure property. That is, the itemsets violating the downward closure property will be deleted from the set of candidate (*k*+1)-itemsets. Up to now, there are many algorithms have been proposed to rapidly discover the frequent itemsets, including Apriori and subsequent Apriori-like algorithms [3, 15], and pattern-growth methods [1, 9, 14].

2.2 Share-Confidence Framework

The support measure does not concern the quantity purchased in a transaction. In real circumstances, products may be bought in plural in a transaction. Therefore, the support value of an item usually underestimates the actual frequency of product purchasing. Information derived from the support value may also be misleading [6]. To address this issue, Carter *et al.* first introduced the share-confidence framework [7]. Instead of a binary attribute, each item i_p involves a numerical attribute in each transaction T_q. The notations and definitions of share measure are described as follows [6, 12, 13].

The *measure value* $mv(i_p, T_q)$ represents the attribute value of i_p in transaction T_q. For example, in Table 1, *mv*(*C*, *T02*) = 3 and *mv*(*D*, *T04*) = 2. The *transaction measure value* is the total measure value of a transaction T_p, denoted as $tmv(T_p)$, where $tmv(T_p) = \sum_{i_p \in T_q} mv(i_p, T_q)$. For example, in Table 1, *tmv*(*T02*) = 10 and *mv*(*T04*) = 9.

The *total measure value Tmv*(*DB*) represents the total measure value in *DB*, where $Tmv(DB) = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q)$. For example, in Table 1, *Tmv*(*DB*) = 47.

Let db_X be a set of transactions that contain itemset *X* in *DB*. That is each *k*-itemset $X \subseteq I$ has an associated set of transactions $db_X \subseteq DB$, where $X \subseteq T_q$ and $T_q \in db_X$. For example, in Table 1, $db_{\{AC\}} = \{T01, T02, T03\}$.

Let $X \subseteq T_q$, the *itemset measure value* of an itemset *X* in a transaction T_q, denoted as $imv(X, T_q)$, is the total measure value of all items of *X* in T_q. That is, $imv(X, T_q) = \sum_{X \subseteq T_q, i_p \in X} mv(i_p, T_q)$. For example, in Table 1, *imv*({*AC*}, *T02*) = 7.

The *local measure value* of an itemset *X* in *DB*, denoted as *lmv*(*X*), is the sum of the itemset measure values of *X* in db_x. That is, $lmv(X) = \sum_{T_q \in db_x} imv(X, T_q)$. For example, in Table 1, $lmv(\{AC\}) = imv\{\{AC\}, T01\} + imv\{\{AC\}, T02\} + imv\{\{AC\}, T03\} = 2 + 7 + 7 = 16$.

The *itemset share value* of an itemset X, denoted as $SH(X)$, is the ratio of the local measure value of X to the total measure value in DB. That is, $SH(X) = \frac{lmv(X)}{Tmv(DB)}$. Given a minimum share (*minShare*) threshold *s%*, A *k*-itemset X is *share-frequent* (SH-frequent) if $SH(X) \geq s\%$; otherwise, X is infrequent.

Example 2.1 Consider the sample transaction database as shown in Table 1 and *minShare* = 30%. Table 2 lists the local measure value and the share value of each 1-itemset, where $Tmv(DB) = 47$. Let $X = \{B, D\}$, the local measure value of $\{B, D\}$ is $lmv(X) = imv(X, T01) + imv(X, T04) + imv(X, T05) + imv(X, T06) = 2 + 6 + 3 + 4 = 15$. $SH(X) = lmv(X)/Tmv(DB) = 15/47 = 0.319 > 30\%$. Therefore, $\{B, D\}$ is SH-frequent. Table 3 illustrates all SH-frequent itemsets.

Table 2. Local measure value and itemset share value of each 1-itemset

Item	{A}	{B}	{C}	{D}	{E}	{F}	{G}	{H}	Total
$lmv(i_p)$	12	9	10	6	4	4	1	1	47
$SH(i_p)$	25.5%	19.1%	21.3%	12.8%	8.5%	8.5%	2.1%	2.1%	100%

Table 3. All SH-frequent itemsets of the sample database

SH-frequent itemset	{A, C}	{B, D}	{A, C, E}	{B, C, D}
$lmv(X)$	16	15	18	16
$SH(X)$	34.0%	31.9%	38.3%	34.0%

2.3 Previous Methods of Discovering Share-Frequent Itemsets

The characteristic of downward closure cannot be applied for discovering SH-frequent itemsets, because the subsets of an SH-frequent itemset may be infrequent. For example, in Table 3, $\{A, C, E\}$ is SH-frequent, but its subsets $\{C, E\}$ and $\{A, E\}$ are infrequent. The ZP and ZSP algorithms can be known as the variants of the exhaustive search method. They generate all itemsets to be candidate set except the local measure values of itemsets are exactly zero [6]. Some algorithms have been proposed to extract SH-frequent itemsets with infrequent subsets, such as SIP, CAC, and IAB [4, 5, 6]. However, they do not generate complete SH-frequent itemsets. Recently, Li *et al*. first proposed the non-exhaustive search method, Fast Share Measure (FSM), to discover all SH-frequent itemsets efficiently [12]. FSM employs the level closure property to decrease the number of candidate itemsets.

Given a transaction database *DB* and *minShare*, the characteristic of level closure is described as follows. For a candidate *k*-itemsets X, and an integer *Level*, if $lmv(X) + (lmv(X)/k) \times MV \times Level < min\_lmv$, no superset of X with length $\leq k + Level$ is SH-frequent, where *MV* is the maximum measure value among all measure values and $min\_lmv = minShare \times Tmv(DB)$. In the case, $Level = ML - k$, where *ML* is the maximum length among all transactions, the level closure property guarantees that no superset of SH-infrequent if the inequality holds.

Li *et al*. also developed some more efficient algorithms than FSM to discover all SH-frequent itemset, including EFSM (Enhanced FSM), SuFSM (Support-counted FSM) and ShFSM (Share-counted FSM) [13]. EFSM reduces the time complexity of

generating candidate k-itemsets from $O(m^{2k-2})$ to $O(m^k)$, where m is the number of distinct items. SuFSM and ShFSM are based on EFSM. They add the support constraint and the share constraint to the critical function, respectively. The performance of ShFSM is the best among ZSP, FSM, EFSM, SuFSM and ShFSM on some synthetic datasets [13].

3 Direct Candidate Generation (DCG) Method

The key point of previous methods, including FSM, EFSM, SuFSM and ShFSM for discovering all SH-frequent itemsets is to eliminate itemset X from candidate set if X satisfies the inequality $CF(X) < min\_lmv$ [12, 13]. Those methods waste computation time in the join and the prune steps of candidate generation, and generate too many candidates of SH-frequent itemsets. In comparison with previous methods, this study proposes a novel algorithm called Direct Candidates Generation (DCG) method to directly generate a smaller candidate set without the join and the prune steps in each pass. Without lose of generality, we let the literal set I be a totally order set. That is, for any $i, j \in I$, either $i \leq j$ or $j \leq i$. We also denote $i < j$ if $i \leq j$ and $i \neq j$.

Definition 3.1 Let the candidate k-itemset X be $\{i_1, i_2, \ldots, i_k\}$ in the order of literals. Let $i_q \in I$ be an item. If $i_k < i_q$ then the $(k+1)$-superset of X $\{i_1, i_2, \ldots, i_k, i_q\}$ is defined as the *monotone (k+1)-superset* of X and is denoted as $X_{k+1}^{i_q}$. For example, Let $X = \{A, C, D\}$. $X_{k+1}^{E} = \{A, C, D, E\}$.

Definition 3.2 Let $X \subseteq I$, the associated set of transactions of X, $db_X = \{ T_q \in DB \mid X \subseteq T_q \}$, is the set of transactions that contain X. The *total measure value* of db_X is defined as $Tmv(db_X) = \sum_{T_q \in db_X} \sum_{i_p \in T_q} mv(i_p, T_q)$.

Let $X_{k+1}^{i_q} \subseteq I$ is an arbitrary monotone $(k+1)$-superset of X. $X_{k+1}^{i_q}$ has an associated set of transactions $db_{X_{k+1}^{i_q}} = \{T_q \in DB \mid X_{k+1}^{i_q} \subseteq T_q\}$. Clearly, $db_{X_{k+1}^{i_q}} \subseteq db_X$. For example, in Table 1, let $X = \{A, C\}$, then $db_X = \{T01, T02, T03\}$ and $db_{X_{k+1}^{E}} = \{T02, T03\}$.

Theorem 3.1 Given a DB and the $minShare$, let $min\_lmv = minShare \times Tmv(DB)$. For k-itemset X, if $Tmv(db_X) < min\_lmv$, all supersets of X (including X) are infrequent.

Proof. Let X' be an arbitrary superset of X with length $(k+i)$, where $i \geq 0$. Clearly, $lmv(X') \leq Tmv(db_{X'}) \leq Tmv(db_X)$. So, if the inequality $Tmv(db_X) < min\_lmv$ holds, $lmv(X') < min\_lmv = minShare \times Tmv(DB)$. That is, $SH(X') = lmv(X') / Tmv(DB) < minShare$. X' is infrequent. Q.E.D

By Theorem 3.1, if $Tmv(db_{X_{k+1}^{i_q}}) < min\_lmv$, $X_{k+1}^{i_q}$ and all supersets of $X_{k+1}^{i_q}$ are infrequent; otherwise, $X_{k+1}^{i_q}$ is a candidate $(k+1)$-itemset.

DCG is also a multiple-pass method for finding all SH-frequent itemsets. In the k-th pass, DCG scans the whole database to count the local measure value of each candidate k-itemset X and counts the potential maximum share value of each monotone $(k+1)$-superset of X. Next, DCG determinates the SH-frequent k-itemset, where their

local measure values are greater than *min_lmv*. Then, DCG selects $X_{k+1}^{i_q}$ to be a candidate (k+1)-itemset if the total measure value of $db_{X_{k+1}^{i_q}}$ satisfies the inequality $Tmv(db_{X_{k+1}^{i_q}}) \geq min\_lmv$. The pseudo-code of DCG algorithm is as follows.

Algorithm DCG()
Input: (1) *DB*: a transaction database with counting, (2) *minShare*
Output: All SH-frequent itemsets
Procedure:

```
1. C1:=I; // Ck: the set of candidate k-itemsets
2. for k:=1 to h
3.   Fk:=φ; Ck+1:=φ; Tmv(db_{X_{k+1}^{iq}}):=0 for all X in Ck;
4.   foreach T∈DB // scan DB
5.        if k==1 { count and store tmv(T); }
6.        foreach X∈Ck
7.             count lmv(X);
8.             foreach iq > ik && iq ∈ T
9.                  Tmv(db_{X_{k+1}^{iq}}) += tmv(T);
10.       foreach X∈Ck
11.            if lmv(X) ≥ min_lmv { Fk:= Fk+X; }
12.            foreach iq > ik && iq ∈ I
13.                 if Tmv(db_{X_{k+1}^{iq}}) ≥ min_lmv { Ck+1:=Ck+1+ X_{k+1}^{iq}; }
14.       if Ck+1==φ exitfor;
15. return F1∪F2∪…∪Fk;
```

In line 5 , DCG calculates the transaction measure value of each transaction and store it in an array when scanning *DB* first time. The transaction measure value of each transaction will be employed in each pass. In lines 8 to 9, DCG accumulates the total measure value of $db_{X_{k+1}^{i_q}}$. From lines 10 to 13, DCG determines which candidate k-itemsets are SH-frequent and directly generates (k+1)-candidates.

Example 3.1 Consider the sample transaction database as listed in Table 1 and *minShare* = 30%. Both *Tmv*(*DB*) = 47 and *min_lmv* = 15 can be calculated easily. In Fig. 1, each circle represents candidate itemset X and each number inside the circle is the local measure value of X, *lmv*(X), such as *lmv*({A})=12 and *lmv*({*BD*}) = 15. To speed-up counting the total measure value of each monotone (k+1)-superset of each candidate X, $Tmv(db_{X_{k+1}^{i_q}})$, we require an array table to store these values for each X. The transaction measure value table is listed in the column "Total count" of Table 1. In the first pass, all items are candidate 1-itemsets. After first scanning *DB*, we can obtain all local measure values of candidates and all $Tmv(db_{X_2^{i_q}})$ values, such as $Tmv(db_{\{A\}_2^C})$ = 26 and $Tmv(db_{\{C\}_2^G})$ = 6. No 1-itemset is SH-frequent and there are seven monotone 2-supersets, with values greater than *min_lmv*, in arrays. These 2-itemsets could be SH-frequent as shown in the dark cells of Fig. 1. Therefore, DCG directly generates the seven 2-itemset candidates {{A, C}, {A, E}, {B, C}, {B, D},

$\{C, D\}$, $\{C, E\}$, $\{C, F\}\}$. In the second pass, DCG discovers two SH-frequent 2-itemsets, $\{\{A, C\}, \{B, D\}\}$, and generates two candidate 3-itemsets, $\{\{A, C, E\}, \{B, C, D\}\}$, because $lmv(\{A, C\}) = 16$, $lmv(\{B, D\}) = 15$, $Tmv(db_{\{AC\}_3^E}) = 20$ and $Tmv(db_{\{BC\}_3^D}) = 21$ are all greater than *min_lmv*. No candidate of C_4 can be generated. Therefore, the process terminates after third scanning *DB*.

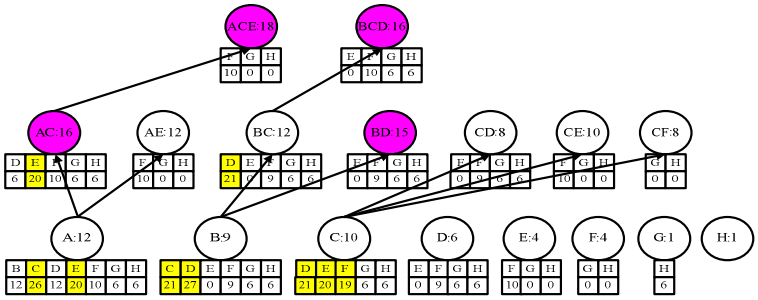

Fig. 1. An example of DCG algorithm

4 Experimental Results

To access the performance of DCG, experiments are conducted to compare its performance with that of FSM, EFSM, SuFSM and ShFSM on artificial and real datasets. All experiments were performed on a 1.5 GHz Pentium IV PC with 1.5 GB of main memory, running Windows XP Professional operating system. All algorithms were coded in Visual C++ 6.0. Each algorithm employed the hash tree structure to count the local measure value of each candidate. All SH-frequent itemsets were output to main memory to eliminate the effect of disk writing.

The artificial datasets were generated by IBM synthetic data generator [18]. To discover SH-frequent itemsets, each item must include an integer attribute. This study modifies these datasets with additional parameter *m*. The notation T*x*.I*y*.D*z*.N*n*.S*m* denotes a dataset with five given parameters *x*, *y*, *z*, *n* and *m*. The definition of the first four parameters is the same as those in [3]. The parameter *m* denotes the measure value which was randomly generated between 1 and *m*, and 50% of measure values in the dataset are set to be 1.

Figures 2 and 3 plot the performance curves of running time associated with these algorithms over various *minShare*, applied to T6.I4.D100k.N200.S10 and T10.I6.D100k.N1000.S10, respectively. Figure 2 uses a logarithmic scale for the *y*-axis and the range of *minShare* is from 0.1% to 1.2%. Figure 2 demonstrates that DCG performed better than FSM by more than one order of magnitude. DCG had the best performance, followed by ShFSM. For example, the running time of DCG was only 62%, 21%, 9.8 and 0.4% of those of ShFSM, SuFSM, EFSM and FSM, respectively with *minShare* = 0.4%.

In Fig. 3, the range of *minShare* is from 0.01% to 0.12%. FSM and EFSM were not illustrated in Fig. 3 because they generated too many candidates to store in main memory. In *minShare* ≤ 0.4% scenarios, SuFSM also generated too many candidates to

run. The running time of DCG was only 59.5% and 16.6% of those of ShFSM and SuFSM with *minShare* = 0.06%, respectively.

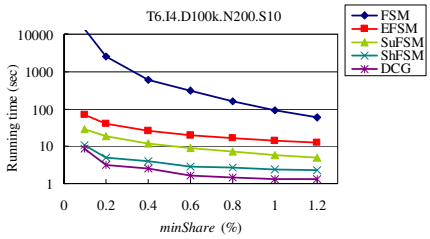

Fig. 2. Comparison of running times

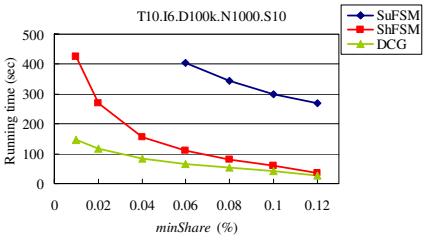

Fig. 3. Comparison of running times

To compare the difference of candidate numbers among these five algorithms in each pass, Table 4 presents the numbers of C_k and F_k in each pass using the dataset T6.I4.D100k.N200.S10 with *minShare* = 0.1%. DCG generated the fewest candidates among the five algorithms. DCG terminated the process at pass 10 and performed best. ShFSM also terminated the process at pass 10. The others executed the processes to pass 13. All five algorithms can discover all SH-frequent itemsets, even to those SH-frequent *k*-itemsets had no SH-frequent (*k*-1)-subset. For example, in the pass 5, all five SH-frequent 5-itemsets have no SH-frequent 4-subset.

Table 4. Comparison of the numbers of candidate sets in each pass

Method / Pass (*k*)	C_k					F_k
	FSM	EFSM	SuFSM	ShFSM	DCG	
k=1	200	200	200	200	200	159
k=2	19900	19900	19701	19306	7200	1844
k=3	829547	829547	564324	190607	9805	101
k=4	3290296	3290296	793042	20913	1425	0
k=5	393833	393833	25003	1050	967	5
k=6	26137	26137	11582	518	510	8
k=7	11141	11141	5940	204	203	7
k=8	4426	4426	2797	58	58	1
$k \geq 9$	2036	2036	1567	12	12	0
Time(sec)	13610.4	71.55	29.67	10.95	8.83	

Figures 4 and 5 compare the scalability of SuFSM, ShFSM and DCG. Figure 4 illustrates the scalability of three algorithms on the transaction numbers of *DB* using T6.I4.D*z*.N200.S10 with *minShare* = 0.3%. The range of *DB* size is between 100k and 1000k. The running times of SuFSM, ShFSM and DCG linearly increase with the growth of the *DB* size. Figure 5 presents the scalability of three algorithms on the maximum measure values of *DB* using T6.I4.D100k.N200.S*m* with *minShare* = 0.3%. The *x*-axis represents the maximum measure values between 10 and 60. The running time curves of SuFSM, ShFSM and DCG were found to be flat. The impact of the distinct maximum measure value on these three approaches was insignificant.

BMS-WebView-2 is a real dataset of several months' click stream data from an e-commerce web site [17]. This study modifies these datasets with an additional parameter *m*. The parameter *m* denotes the measure value of each item was randomly generated between 1 and *m*, and 50% of measure values in the dataset are set to be 1. Figure 6 plots the running-time curves associated with ShFSM and DCG. The range of *minShare* is between 0.2% and 1%. In these distinct *minShare* scenarios, FSM, EFSM and SuFSM generated too many candidates to store in main memory. In Fig. 6, when *minShare* ≤ 0.6%, the running times of DCG are only between 59.5% (*minShare* = 0.6%) and 16.6% (*minShare* = 0.2%) of that of ShFSM.

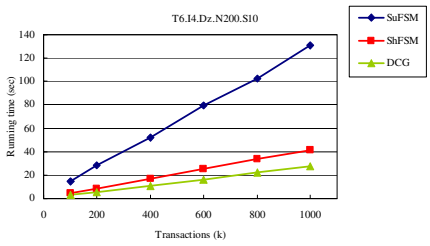

Fig. 4. Scalability of algorithms

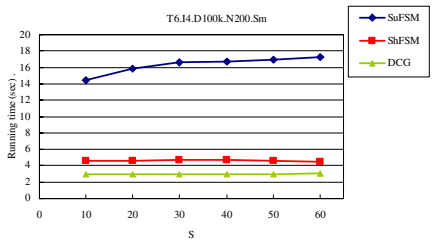

Fig. 5. Scalability of algorithms

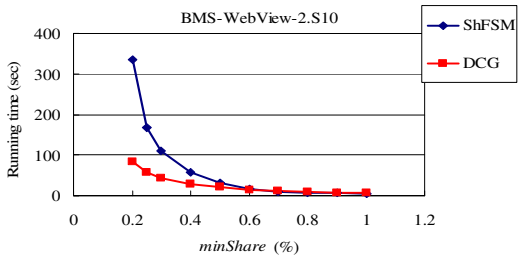

Fig. 6. Comparison of running times on BMS-WebView-2.S10

5 Conclusions

The share measure has been proposed to overcome the drawbacks of the support measure. Therefore, developing an efficient approach for discovering complete SH-frequent itemsets is very valuable. This study proposes the Direct Candidates Generation (DCG) algorithm to avoid the join and the prune steps in each pass. Furthermore, DCG significantly reduces the number of candidates and improves the performance. Experimental results indicate that DCG outperforms all other algorithms in several artificial datasets. Now, we are investigating the development of superior algorithms to efficiently accelerate the process of identifying long SH-frequent itemsets.

Acknowledgements

We would like to thank Blue Martini Software, Inc. for providing the BMS datasets.

References

1. R. C. Agarwal, C. C. Aggarwal, V. V. V. Prasad: A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing, **61** (2001) 350-361
2. R. Agrawal, T. Imielinski, A. Swami: Mining association rules between sets of items in large databases. In: Proc. 1993 ACM SIGMOD Intl. Conf. on Management of Data, Washington, D.C. (1993) 207-216
3. R. Agrawal, R. Srikant: Fast algorithms for mining association rules. In: Proc. 20th Intl. Conf. on Very Large Data Bases, Santiago, Chile (1994) 487-499
4. B. Barber, H. J. Hamilton: Algorithms for mining share frequent itemsets containing infrequent subsets. In: D. A. Zighed, H. J. Komorowski, J. M. Zytkow (eds.): 4th European Conf. on Principles of Data Mining and Knowledge Discovery. Lecture Notes in Computer Sciences, Vol. 1910. Springer-Verlag, Berlin Heidelberg New York (2000) 316-324
5. B. Barber, H. J. Hamilton: Parametric algorithm for mining share frequent itemsets. Journal of Intelligent Information Systems, 16 (2001) 277-293
6. B. Barber, H. J. Hamilton: Extracting share frequent itemsets with infrequent subsets. Data Mining and Knowledge Discovery, **7** (2003) 153-185.
7. C. L. Carter, H. J. Hamilton, N. Cercone: Share based measures for itemsets. In: H. J. Komorowski, J. M. Zytkow (eds.): 1st European Conf. on the Principles of Data Mining and Knowledge Discovery. Lecture Notes in Computer Science, Vol. 1263, Springer-Verlag, Berlin Heidelberg New York (1997) 14-24
8. R. Chan, Q. Yang, Y. D. Shen: Mining high utility itemsets. In: Proc. 3rd IEEE Intl. Conf. on Data Mining, Melbourne, FL (2003) 19-26
9. J. Han, J. Pei, Y. Yin, R. Mao: Mining frequent pattern without candidate generation: A frequent pattern tree approach. Data Mining and Knowledge Discovery, **8** (2004) 53-87
10. R. J. Hilderman, C. L. Carter, H. J. Hamilton, N. Cercone: Mining association rules from market basket data using share measures and characterized itemsets. Intl. Journal of Artificial Intelligence Tools, **7** (1998) 189-220
11. M. Kantardzic: Data mining: Concepts, models, methods, and algorithms. John Wiley & Sons, Inc., New York (2002)
12. Y. C. Li, J. S. Yeh, C. C. Chang: A fast algorithm for mining share-frequent itemsets. In: Y. Zhang, K. Tanaka, J. X. Yu, etc. (eds.): 7th Asia Pacific Web Conf. Lecture Notes in Computer Science, Vol. 3399, Springer-Verlag, Berlin Heidelberg New York (2005) 417-428
13. Y. C. Li, J. S. Yeh, C. C. Chang: Efficient algorithms for mining share-frequent itemsets. To appear in Proc. 11th World Congress of Intl. Fuzzy Systems Association (2005)
14. J. Liu, Y. Pan, K. Wang, J. Han: Mining frequent item sets by opportunistic projection. In: Proc. 8th ACM-SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, Alberta, Canada (2002) 229-238
15. J. S. Park, M. S. Chen, P. S. Yu: An effective hash-based algorithm for mining association rules. In: Proc. 1995 ACM-SIGMOD Intl. Conf. on Management of Data, San Jose, CA (1995) 175-186
16. K. Wang, S. Zhou, J. Han: Profit mining: From pattern to actions. In: C. S. Jensen, K. G. Jeffery, J. Pokorný, etc. (eds.): 8th Int. Conf. on Extending Database Technology. Lecture Notes in Computer Science, Vol. 2287**,** Springer-Verlag, Berlin Heidelberg New York (2002) 70-88
17. Z. Zheng, R. Kohavi, L. Mason: Real world performance of association rule algorithm. In: Proc.7th ACM-SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, San Francisco, CA (2001) 401-406
18. http://alme1.almaden.ibm.com/software/quest/Resources/datasets/syndata.html