DisClose: *Dis*covering Colossal *Close*d Itemsets via a Memory Efficient Compact Row-Tree

Nurul F. Zulkurnain[1, 3], David J. Haglin[2], and John A. Keane[3]

[1] Department of Electrical and Computer Engineering, Kuliyyah of Engineering, International Islamic University Malaysia, P.O. Box 10, 50728 Kuala Lumpur, Malaysia
nurulfariza@iium.edu.my
[2] High Performance Computing, Pacific Northwest National Laboratory, P.O. Box 999, MSIN J4-30, Richland, WA 99352, USA
david.haglin@pnl.gov
[3] School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK
{zulkurnn,jak}@cs.man.ac.uk

Abstract. A recent focus in itemset mining has been the discovery of frequent itemsets from high-dimensional datasets. With exponentially increasing running time as average row length increases, mining such datasets renders most conventional algorithms impractical. Unfortunately, large cardinality itemsets are likely to be more informative than small cardinality itemsets in this type of dataset. This paper proposes an approach, termed *DisClose*, to extract large cardinality (colossal) closed itemsets from high-dimensional datasets. The approach relies on a Compact Row-Tree data structure to represent itemsets during the search process. Large cardinality itemsets are enumerated first followed by smaller ones. In addition, we utilize a minimum cardinality threshold to further reduce the search space. Experimental results show that *DisClose* can achieve extraction of colossal closed itemsets in the discovered datasets, even for low support thresholds. The algorithm immediately discovers closed itemsets without needing to check if each new closed itemset has previously been found.

Keywords: Colossal closed itemset, high-dimensional dataset, minimum cardinality threshold.

1 Introduction

Rapid development in information technology has provided organizations with the ability to store, process and retrieve huge amounts of data. Nevertheless, there is a need to extract useful information and knowledge, efficiently and effectively, from these massive data stores. This serve to assist businesses, scientific and government related organizations to better plan, predict, and make decisions. This has led to the importance of data mining and the need to provide effective and efficient associated algorithm implementation.

Itemset mining has recently focused on the discovery of frequent itemsets from high-dimensional datasets with relatively few rows and a larger number of items [1],

T. Washio and J. Luo (Eds.): PAKDD 2012 Workshops, LNAI 7769, pp. 141–156, 2013.

[2], [3], [8]. With exponentially increasing running time as average row length increases, mining such datasets renders most conventional algorithms impractical. Several papers have proposed the row-enumeration method to discover frequent itemsets based on the set of rows space instead of the itemset space [1], [2], [3], [8].

Nevertheless, due to the large number of frequent itemsets, discovering all frequent itemsets remains difficult. Strategies to provide more compact sets of frequent itemsets have been proposed such as finding only maximal frequent itemsets [4] or only closed frequent itemsets [5]. Closed itemsets provide a smaller set of results without information loss. Nonetheless, due to the density of high-dimensional data, it is difficult to enumerate all closed itemsets especially at the lower of the support spectrum [1], [2], [3], [8].

The most frequent itemsets tend to be both relatively smaller in size and larger in number. This quickly leads to insufficient memory when attempting to reach less frequent itemsets. Also, the most frequent itemsets can easily be extracted. In addition, applying the support constraint results in the pruning of many large cardinality itemsets that exist at this lower end of the support spectrum. Hence, discovery that starts from the largest cardinality itemsets in high-dimensional datasets may provide interesting insight into how these itemsets correlate.

Determining whether a mined pattern is closed is regarded as the main challenge in closed itemset mining [6]. Repeated checking to verify whether the itemsets are closed is costly in term of processing. Hence, discovering closed itemsets during the search process, and thus reducing the need for checking, should reduce computation.

This paper proposes discovery of colossal closed itemsets using a compact row-tree which reduces the memory required to store itemsets during the search process. The search for itemsets proceeds from the largest to the smallest by applying a search strategy that begins with the largest cardinality itemset and builds smaller itemsets. This strategy is combined with a bottom-up row enumeration search. We further utilize a minimum cardinality threshold to reduce the search space and focus on only colossal closed itemsets. We show that the algorithm immediately discovers colossal closed itemsets without the need to check each previously discovered closed itemsets.

The paper is structured as follows: Section 2 formulates the problem; search strategies are discussed in Section 3; in Section 4 the closedness-checking method is described; Section 5 presents the Compact Row-Tree; the algorithm and supporting theory are given in Section 6; experimental result are presented in Section 7; and Section 8 concludes.

2 Problem Formulation

Let T be a dataset table that consists of a collection of rows (transactions), $R = \{r_1, r_2, ..., r_m\}$ and a list of discrete items, $I = \{o_i, o_2, \ldots, o_n\}$. This set of transactions represents the number of rows (m) and the set of items signifies the number of columns (n) in T.

Table 1. Example of a discretized high-dimensional dataset

tid	Item													
	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***
1	1	1	1	2	2	1	2	1	2	2	2	2	2	1
2	2	1	2	2	2	2	2	2	1	2	2	2	1	2
3	1	1	2	1	2	2	2	2	2	1	2	2	2	2
4	2	1	2	1	2	2	1	2	2	2	2	2	2	2
5	1	1	2	1	1	2	2	2	2	1	2	2	2	2

A nonempty subset $\alpha \subseteq I$ is called an *itemset*. An itemset, α_k, which consists of k items, is described as k-itemset. Each row r_i is represented by a unique row identifier. Let $t(r_i)$ denote the itemset at row i of the table. Within a dataset, all of the row identifiers must be unique, but there may be duplicate row itemsets. That is, for $r_1 \neq r_2$, it may be that $t(r_1) = t(r_2)$. A set of rows is termed a *rowset*.

Example 1. (Table *T*) Table 1 illustrate as example of a discretized high-dimnesional dataset, *T*, that contains five rows and 14 items, so $R = \{1, 2, 3, 4, 5\}$ and $I = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n\}$.

Definition 1. (*Support Set*) Given an itemset α, the *support set* is represented as the set of rows in the dataset, *T*, that contain α. This is represented as:

$$r(\alpha) = \{r_i \mid \alpha \subseteq t(r_i)\} \tag{1}$$

Example 2. (*Support Set*) In Table 1, for an itemset $\alpha = \{a_1, b_1, d_1, e_2\}$, the support set $r_\alpha = \{3, 4\}$.

Definition 2. (*Support*) The support of an itemset α is the number of rows in which α occurs in *T* – denoted as $| r_\alpha |$.

Example 3. (*Support*) From Example 2, the support for itemset $\alpha = \{a_1, b_1, d_1, e_2\}$, $| r_\alpha | = | \{3,4\} | = 2$.

The relative support of α is $|r(\alpha)|/|r|$. It is well known that the support measure has an anti-monotonic property where $|r(\alpha_1)| \geq |r(\alpha_2)|$ for $\alpha_1 \subset \alpha_2$.

Definition 3. (*Frequent Itemset*) Given a dataset *T* and a minimum support threshold *minsup*, an itemset α is frequent if $|r(\alpha)| \geq minsup$.

Definition 4. (*Closed Itemset*) An itemset α is a closed itemset in dataset *T* if there is no proper superset α' exists $(\alpha \subset \alpha')$ such that the support of α is the same as the support of α'.

Definition 5. (*Closed Rowset*) A rowset β is a closed rowset in table T if not a proper superset β' exists $(\beta \subset \beta')$ such that the support of β is the same as the support of β'.

Definition 6. (*Closure*) Given a rowset β, we define $I(\beta) = \{o_j \in I \mid \forall_{r_k} \in \beta : o_j \in t(r_k)\}$. Following this, we can define $C(\alpha)$ as the closure of itemset α and $C(\beta)$ as the closure of rowset β as follows:

$$C(\alpha) = I(r(\alpha)) \tag{2}$$

$$C(\beta) = I(r(\beta)) \tag{3}$$

3 Search Strategies

3.1 Existing Itemset Search Strategies

The traditional search strategy explores the itemset space bottom-up: beginning from the smallest itemset that appears frequently and uses intermediate results to progressively build larger and larger itemsets. Conventional algorithms that use this strategy, such as *FP-Close* [5], are efficient for datasets containing relatively many rows and fewer columns (items) e.g. transactional data.

In contrast, high-dimensional datasets have a relatively large number of columns (items) and relatively few rows. If k is the maximum itemset size, there could be 2^k potential frequent itemsets. Exploring the dataset based on the number of items makes searching for closed frequent itemsets over the itemset space impractical.

CARPENTER [1] is an example of algorithms that search for closed frequent itemsets based on the rowset space. The algorithm conducts a bottom-up traversal of the row enumeration tree. Each node is checked to see if it is frequent and closed. As this criterion is based on the minimum support threshold, the nodes that do not satisfy the support constraint still need to be checked. As a result, the algorithm consumes both more memory and time in order to obtain the desired threshold.

By using the pruning power of the support threshold to reduce the search space, a top-down approach using a row enumeration tree has been proposed to discover closed frequent itemsets [2]. The search begins from the largest rowset and successively builds smaller and smaller rowsets. However, difficulties are still encountered in reaching the lower end of the support spectrum as much memory is consumed by the large numbers of closed frequent itemsets at the higher end.

A related problem is that of finding a formal concept (FC) [3]. Given a 0/1 matrix, a formal concept is a subset of k rows and l columns such that all of the matrix entries in one of the k rows and l columns contain a 1. Such a row and column subset is called a 1-rectangle. If the rows were rearranged so that all of the k subset rows appeared first (i.e. in rows 1 through k) and all columns were rearranged so that the l columns of the subset appeared in columns 1 through l, the upper-left k by l rectangle of the matrix would contain all 1 entries.

3.2 Proposed Search Strategy

A strategy for high-dimensional datasets is to search for closed itemsets based on the row number. Previous algorithms that use row-enumeration strategies to discover closed itemsets rely on the support constraint to reduce search space [1], [2], [3], [8]. As the frequency threshold reduces, the time and memory required for these algorithms to find closed itemsets dramatically increases. Yet the most valuable closed itemsets in high-dimensional data may have relative support values much closed to 1 than 100.

Therefore, we propose that rather than generating closed itemsets from the smallest set of items with higher supports, we search from the largest set of items that exists in a row possibly with very small support threshold. From this collection of closed itemsets, we can build increasingly smaller itemsets with increasingly higher support.

Bottom-Up Row-Enumeration Search

To extract the largest itemset from a dataset involves extracting the largest column that exists in the dataset. This implies that the search strategy can be based on a top-down column enumeration.

However, it can be observed that for a dataset with m number of columns (items), there will also be m number of levels for a top-down column enumeration tree. In addition, the maximum number of nodes (itemsets) that will exist in the top-down column enumeration will equal 2^m-1. For a high-dimensional dataset, the value of m is very large (i.e. hundreds of thousands); hence, enumerating the itemsets based on the number of columns is infeasible.

It makes sense to search for closed itemsets based on the number of rows because it is relatively smaller than compared to the number of columns in high-dimensional datasets [1], [2], [8]. The largest cardinality itemset initially exists in every single row of the high-dimensional dataset (unless duplicate rows occur). Therefore, most large closed itemsets begin from the infrequent end of the support spectrum. As a result, using the bottom-up row enumeration tree as the basis of the search strategy would appear to be more appropriate.

Transposed Table

Since its proposal, the transposition method [7] has been widely used by algorithms that discover closed itemsets from high-dimensional datasets [1], [2], [8]. Mining the closed itemsets directly from the original dataset can be complicated. Therefore, applying the method of transposition to the original dataset helps to simplify the extraction of closed itemsets in high-dimensional data. This is because when the original dataset is transposed, each column (item) value of the original dataset will become a row value in the transposed table, and will be represented by a set of rows (rowset) where that particular item occurs.

Transposed dataset provides a sparser representation of the original input dataset. As a result of this simplification, the method of transposition is utilized in the algorithm proposed here.

Minimum Cardinality Threshold, *mincard*

Definition 7. (*Cardinality*) The cardinality of an itemset α refers to the number of items in α. This is denoted as $|\alpha|$.

In contrast to the support threshold that helps to reduce the search space based on occurrence frequency, we stop the mining process for closed itemsets upon reaching the threshold parameter value for the minimum itemset cardinality, *mincard*.

Let $CI(T) = \{\alpha \mid \alpha \in T$ is a closed itemset$\}$. Large cardinality closed itemset mining involves enumerating all $\alpha \in CI(T)$ with $|\alpha| \geq$ *mincard*. We refer to these large cardinality closed itemsets as colossal closed itemsets (*CCI*).

Using the bottom-up row enumeration tree [1], branch exploration stops once the cardinality of the associated itemset falls below *mincard*. We can safely prune the search because of the anti-monotone property.

Property 1. (*anti-monotone*) If a rowset β has its associated $\alpha = I(\beta)$ such that $|\alpha| <$ *mincard*, then for any $\beta' \supseteq \beta$ it must be that $|I(\beta')| <$ *mincard*.

Combining both the anti-monotone property and the definition of closure gives the following property.

Property 2. (*at-threshold*) If a rowset β has its associated $\alpha = I(\beta)$ such that $|\alpha| ==$ *mincard*, then for any $\beta' \supseteq \beta$ it must be that $|I(\beta')| <$ *mincard*.

Note: Using a depth-first order in a serial implementation would result both in the most aggressive pruning of the search space and require the least memory.

4 Closedness-Checking Method

Mining for colossal closed itemsets has two restrictions: firstly, the need to check if an itemset is a colossal itemset and secondly, the need to check if it is closed. Using the minimum cardinality threshold in a bottom-up row enumeration search takes advantage of the first constraint. However, discovering only the colossal itemsets may lead to the production of several identical colossal itemsets,

Therefore, when a colossal itemset is found, the next step is to develop a method to efficiently identify whether it is a closed itemset. The method of identifying whether the itemsets discovered are closed is related closely to the search strategy proposed.

To take advantage of the second restriction in making the mining of colossal itemsets more efficient, a method which is based on a unique generator is developed. To define the unique generator, we begin by providing the definition for *itemset generator* and *tidset generator*.

Definition 8. (*Itemset Generator*) Given a dataset *T*, an itemset α is an itemset generator if no proper subset $\alpha' \subset \alpha$ exists such that the support of α is the same as the support of α'.

The equivalence class of itemsets with the same support set consists of exactly one closed itemset, potentially many itemset generators and potentially many itemsets that are neither closed nor generators.

Definition 9. (*Rowset Generator*) Given a dataset *T*, a rowset β is a rowset generator if no proper subset $\beta' \subset \beta$ exists such that the itemset of β is the same as the itemset of β'.

Similarly, the equivalence class of rowsets β_i with same itemset α such that $I(\beta_i) = \alpha$ consists of exactly one closed rowset, there are potentially many rowset generators and potentially many rowsets that are neither closed nor generators.

It can be observed that unlike the definition of frequent itemsets, the definitions of generators and closed sets do not depend upon any threshold parameter.

To construct smaller closed itemsets from larger ones, we use the following property:

Theorem 1. Suppose α_1 and α_2 are closed itemsets, with $\alpha_1 \neq \alpha_2$. Let $\alpha = \alpha_1 \cap \alpha_2$. If $\alpha \neq \emptyset$ then α is a closed itemset.

Proof: We have three cases to consider:

1. **Case 1:** $[\alpha_1 \subset \alpha_2]$. ***Observe that in this case $\alpha = \alpha_1$, so α is a closed itemset.***
2. **Case 2:** $[\alpha_2 \subset \alpha_1]$. ***Observe that in this case $\alpha = \alpha_2$, so α is a closed itemset.***

   For Case 1 and Case 2, in order for α_1 and α_2 to be closed itemsets with one a proper subset of the other, it must be the case (by definition of closed itemset) that they have different support. But we do know that such a situation exists.

   Consider any closed itemset α_1 with support larger than one, and pick any row r_i containing α_1 (i.e. $\alpha_1 \subset t(r_i)$). Now consider $\alpha_2 = t(r_i)$. Note that by definition all full-rowsets are closed. Clearly, this satisfies the conditions of Case 1. The rest of the case is just fundamental set theory, so the result holds.

3. **Case 3:** [α_1 and α_2 are incomparable]. ***Observe that $\alpha \subset \alpha_1$ and $\alpha \subset \alpha_2$.***
   In this particular case, it is demonstrated that α is a closed itemset by contradiction. Assume that α is not a closed itemset, then there exists some item *i* such that $\alpha_i = \alpha \cup \{i\}$ has the same support as α. If $i \notin \alpha_1$, then all rows in $T_{\alpha_1} - T_\alpha$ are

not in T_{α_1}, but they are in T_α. Thus i must be in α_1. However, if $i \in \alpha_1$ (and not in α) then $i \notin \alpha_2$ and the same contradiction argument applies. Thus the assumption that α is not a closed itemset must be invalid.

Lemma 1. Every closed itemset that is not one of the entire transactions can be produced by intersecting some collection of closed itemsets.

Proof. Consider a closed itemset α and its corresponding rowset $\beta = T_\alpha$, as α is a closed itemset, $\alpha = \bigcap_{r_i \in \beta} \alpha_i$, where $(r_i, \alpha_i) \in T$. However, there may be many subsets of β for which $I(\beta) = \alpha$.

Using rowset enumeration as the control strategy for the search process, the same closed itemset would probably be found many times. The following observation allows a closed itemset to be found using only one of the rowsets. As stated above, for every closed itemset α, there is a unique rowset β that is a closed rowset.

Definition 10. (*Unique Generator*) Given the closed rowset $\beta = \{r_1, r_2, \ldots, r_k\}$, $r_i < r_j$ for all $i < j$, the smallest index for which $\beta_j = \{r_1, r_2, \ldots, r_j\}$ is a generator of β is a unique rowset generator for our itemset α.

It is simple to determine if a rowset β' is the unique generator. Let $\beta = T(I(\beta'))$. If $\beta' = \beta$, then the answer is that β' is the unique generator. If $\beta' \subset \beta$, β' is determined whether it is a prefix of β when the rowsets are written as lists in ascending order. If β' is not a prefix of β, then β' can be ignored and this branch of the search space is pruned.

The search for the unique generator will require relatively little computation when the number of rows is small; and this is the typical situation for high-dimensional datasets.

5 Compact Row-Tree

To assists the efficiency of the search, a compact tree data structure is built to store the itemsets from the transposed table, T^t. The *CR-Tree* is initially generated by building a set of nodes at the first level ($l = 0$) of the tree which represents each column value of the transposed table. These set of nodes are connected to each column of the transposed table through a set of pointers that link the node to the transposed table. The construction of the *CR-Tree* continues by adding the child nodes at each level of the tree. As the level of the tree increases, the number of child nodes decreases as the lowest node value from the previous level of the tree is discarded. A child pointer is then built to link between the nodes. In addition to the child pointer, an additional node link is made from the parent node to the child node that contains the same node value. The purpose of this node link is to assist in checking effectively for closed itemsets.

The structure of the *CR-Tree* is similar to the *FR-Tree* [2]. The *CR-Tree* is different in that instead of representing each branch of the tree to a rowset value, each node of the *CR-Tree* represents a group of rowset values. In this way, the *CR-Tree* becomes more compact as one node is shared by many rowset values. Each rowset value represents an itemset.

However, only one rowset value will be stored in each node of the *CR-Tree* during the search process. This is to ensure that a relatively small amount of memory is utilized during the process of mining the colossal closed itemset.

The characteristics of the *CR-Tree* are as follows:

1. The *CR-Tree* represents all values of the complete rowset.

   Let $N = \{n_i, n_{i+1}, \ldots, n_k\}$ be the set of nodes where $i = 1$ and k is the largest row value from the dataset. Let $M = \{m_j, m_{j+1}, \ldots, m_k\}$ be the set of child nodes where $j = i + 1$ and k is the largest row value of the dataset. Each $m_j = n_i \cup n_{i+l}$ where $l = \{1, 2, \ldots, k\text{-}1\}$. Therefore all β values are traversed until $i = k$.

   Subsequent itemsets of the child nodes are obtained by intersecting the itemsets of the parent nodes. To reduce memory during the search for colossal closed itemsets, each node in the *CR-Tree* only stores one itemset value from the intersection of its parent nodes. Each rowset of the parent nodes is a subset of a rowset value of the child node.

2. Each node of the *CR-Tree* only stores one β value at a time for rowset β, with $|\beta|$ = node level.

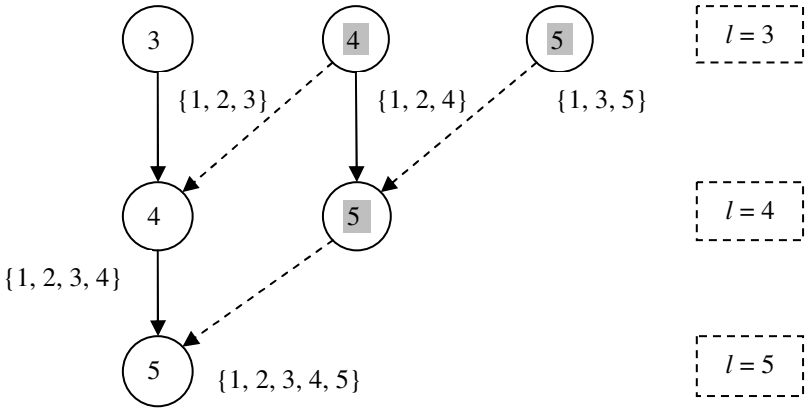


Fig. 1. Example of the second characteristics of the *CR-Tree*

Suppose at $l = 3$, $n_4 = \{1, 2, 4\}$ and $n_5 = \{1, 3, 5\}$. To obtain β for child node m_5, the union of the parent β values will produce, $n_4 \cup n_5 = \{1,2,4\} \cup \{1,3,5\} = \{1, 2, 3, 4, 5\}$. However, $\{1, 2, 3, 4, 5\}$ is not stored in m_5. This is because, based on

the depth-first strategy, the itemset for β = {1, 2, 3, 4, 5} will already have been discovered at l = 5.

The structure of the *CR-Tree* also assists in optimizing identification of the closed itemsets. For example, consider node 3 at the second level of the tree. Assume that the node contains an itemset with row values {1, 3}. Using the proposed closedness-checking method, the node will intersect with the nodes at the third level (Nodes 3, 4, 5) and then check with row values – {1, 2, 3}, {1, 3, 4} and {1, 3, 5} whether the itemset of {1, 3} exists in row 2, 4, or 5.

3. If discovered itemset, $\alpha_2 \subseteq \alpha_1$ where α_1 is the existing itemset in the node, the itemset α_2 will not replace α_1 although $\beta_1 \neq \beta_2$.

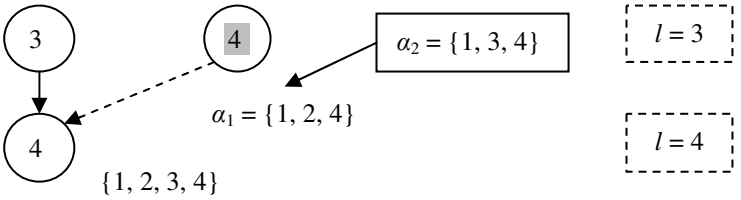


Fig. 2. Example of the third characteristics of the *CR-Tree*

In Fig. 2, suppose $\alpha_1 = \{\beta_1\} = \{1, 2, 4\}$ and $\alpha_2 = \{\beta_2\} = \{1, 3, 4\}$ at level l = 3, where $|\beta_1| = |\beta_2|$. If $\alpha_2 \subseteq \alpha_1$, this means that α_2 also exists in $\{\beta_1\}$. Therefore, $\beta_1 \cup \beta_2 = \beta$, where $|\beta| > |\beta_1|, |\beta_2|$. Thus α_2 will exists in β = {1, 2, 3, 4} where β already exists at level, l = 3 of the *CR-Tree*.

6 Algorithm *DisClose*

To show the effectiveness of the search strategy, the closedness-checking method and the data structure proposed, a colossal closed itemset mining algorithm called *DisClose* has been designed to mine all colossal closed itemsets from the transposed table T^t of table T. *DisClose*, shown in Algorithm 1, will search the row enumeration space and, for each rowset, β, check whether it is the unique generator in the equivalence class of rowsets for I (β). It is noted that using a depth-first order in a serial implementation would result in the most aggressive pruning of the search space and requires the least the amount of memory [1], [10]. For this reason, the general processing order for the rowsets is equivalent to the depth-first search of the row enumeration tree.

6.1 Major Steps of *DisClose*

Algorithm 1 shows the main steps of the algorithm *DisClose*. Assuming the *mincard* threshold value has been assigned, the algorithm begins by transforming table T into

Algorithm 1. *DisClose* algorithm

Input: Table T, and minimum cardinality threshold, *mincard*
Output: A complete set of colossal closed itemsets, *CCI*
Method:

1. Transform T into transposed table T^t
2. Build *CR-Tree*
3. Initialize $CCI = \emptyset$
4. Call Subroutine Colossal (T^t, *mincard*)

Subroutine Colossal (T^t, *mincard*)
Method:

5. **for** each node in the row enumeration space **do**
6. If | node [1][j] |.T^t | ≥ *mincard*
7. Store itemset at node [1][j]
8. Let β be the set of rows under consideration
9. node [l][j] → node [l+1][p] // pointing to child node
10. $\alpha = \alpha_1 \cap \alpha_2 = I(\beta)$, $\beta = \beta_1 \cup \beta_2$
11. ***Optimization S1***: If | α | < *mincard*, discard α
12. ***Optimization S2***: If | β | > current node level, discard β
13. ***Optimization S3***: If $\alpha \subseteq \alpha'$, discard α
14. Store α in node [l+1][p]
15. Call Subroutine **Closed** (*mincard*)

Subroutine Closed (*mincard*)
Method:

16. If node [l][j] == node [l+1][p] // checking for unique generator
17. Call Subroutine **Colossal** (*mincard*)
18. Store itemset in *CCI*
19. Call Subroutine **Colossal** (*mincard*)

transposed table T^t using the transposition operation. After the transposed table T^t is generated, the *CR-Tree* is built in Step 2 in order to access the colossal itemsets from T^t. The *CR-Tree* connects nodes at the first level to T^t through side-links. The side-link pointers enable direct access to the colossal itemsets from T^t. These pointers connect the node with the column T^t of equivalent value.

DisClose is composed of two main subroutines: *Colossal* and *Closed*. After initialization of the set of colossal closed itemsets *CCI* to be empty at Step 4, the subroutine *Colossal* is called to deal with the transposed table T^t using the *CR-Tree* and find all colossal itemsets. Following the bottom-up row enumeration as the search order in

step 5, the subroutine *Colossal* takes the transposed table, T^t and the *mincard* threshold as the parameter to ensure that itemsets with cardinality less than the specified is not stored as it is impossible for subsequent child nodes of the *CR-Tree* to contain itemsets of larger size.

There are seven sections in the subroutine *Colossal*, which will be explained one by one.

The first section is step 6 – step 7. The subroutine begins by accessing T^t through a side-link from the *CR-Tree*. The size of the itemset is checked for each column at Step 6. Only the itemsets that satisfy *mincard* are stored at the first level node of the *CR-Tree*; otherwise, it is not stored in the node as it will not contribute to obtaining larger itemsets. The advantage of this is that the algorithm does not require further access to the dataset, and hence, reduces the time required for repeated checking of the dataset. Note that this is the only role the transposed table T^t plays in the search process.

The second section is steps 8 – step 10. For each node in the *CR-Tree*, the intersection of the itemsets between the parent nodes continues using the depth-first search of the bottom-up row-enumeration tree in Step 10 is performed. By using depth-first search, *DisClose* produces the sequence $\beta \Rightarrow I(\beta)$. However, three optimization strategies are applied before the result of the intersection is stored in each child nodes.

At step 11, an optimization strategy *S1* is applied to stop further processing of the itemset if the size of the itemset does not satisfy the *mincard* constraint defined.

Step 12 performs the optimization strategy *S2* to prevent storage of itemsets with rowset values larger than the node level of the *CR-Tree*.

At step 13, optimization strategy *S3* is applied in order to ensure that the itemset obtained is not a subset of an already existing itemset in the child node.

Step 14 then stores the itemset that does not satisfy any of the three optimization strategies at the particular child node. The new itemset will replace the itemset that already exists in the node.

At step 15, the subroutine *Closed* is called when all the colossal itemsets of the child nodes have been discovered, in order to check whether the parent node is a closed itemset.

The subroutine *Closed* performs the closedness-checking method on the itemset. There are four main steps to this subroutine.

Step 16 sequentially compares the itemset α that exists in the parent node with the itemsets of its child nodes in order to identify the unique generator, based on a depth-first search of the rowset value in the row enumeration tree. Here, the node-link, which connects the parent and child node that contain the same node value, is used to perform the closedness-checking method. This is to ensure that it does not overlook existing child nodes with rowset β that contains a rid value that does not exist in rowset β' of the parent node.

7 Experimental Evaluation

Due to the space limitation, the experimental evaluation shows the comparison of *DisClose* with selected algorithms on one synthetic dataset.

The experiments were performed on a PC with a 2.66 GHz Intel Core2 Quad CPU Q9400 with 4.00 GB RAM and 150 GB hard disk. *DisClose* is implemented in C++.

The performance of *DisClose* was studied by comparing it with other state-of-the art algorithms. Each algorithm was selected to represent the different search strategies. These algorithms are: (i) *FP-Close* [5] - a representative of the column enumeration-based algorithms, (ii) *CARPENTER* [1] – a representative of bottom-up row enumeration-based algorithms, (iii) *D-Miner* [9] – a representative of constraint-based mining algorithms, and (iv) *TTD-Close* [2] – a representative of the top-down row enumeration search based set of algorithms.

All of the selected algorithms have been implemented in C++. Note: all of runtimes plotted in the figures include both computation time and I/O time.

Existing itemset mining algorithms – particularly those that find closed itemsets, - routinely present run-times for varying support thresholds. As *DisClose* uses a thresh old for cardinality, direct comparison is difficult. Given a support threahols greater than 1, existing algorithms would not find many large-cardinality closed itemsets; given a cardinality threshold greater than 1, *DisClose* would not find many frequent closed itemsets. The only fair way to compare the algorithms is to give both a threshold of 1, asking each to find all closed itemsets. The strength of *DisClose* is that it bypasses the huge number of small cardinality, high-frequency closed itemsets and focuses almost immediately on potentially valuable closed itemsets. However, this type of complete closed itemsets search does not really address the true purpose of either *DisClose* or the closed itemset mining algorithms.

Amongst the selected algorithms listed above, only *D-Miner* has been found to apply the minimum cardinality threshold, *mincard*. *D-Miner* is a constraint-based algorithm which uses the cardinality threshold in addition to the support constraint to discover concepts (closed itemsets). Hence, *D-Miner* is the closest comparison to *DisClose* with the exception that the support threshold in *D-Miner* is set to 1.

For other algorithms, an approach was to present the experimental results of *DisClose* with a secondary *x*-axis which represents the maximum support of the colossal closed itemsets discovered. Likewise, a secondary *x*-axis is also added to the results of *FP-Close*, *CARPENTER*, and *TTD-Close* which represents the maximum cardinality of the closed frequent itemsets discovered. Thus, by using this approach, it provides an observation on the ability and limitation of closed itemset mining algorithms that uses a support threshold in relation to *DisClose*, and vice-versa.

Another challenge in comparing performance of the algorithms is based on their implementation in identifying items in the datasets. For *FP-Close*, *CARPENTER*, and *D-Miner*, the algorithms were designed to identify each item based on the value present for each attribute of the dataset. However, for *TTD-Close*, each item in the dataset is read as a value that corresponds to the attribute of the data.

7.1 Synthetic Dataset

The synthetic dataset, generated using the IBM Quest Data Generator, consists of 100 rows, 4000 columns and an average itemset size of 2000. This dataset is represented as T100L2000N4000.

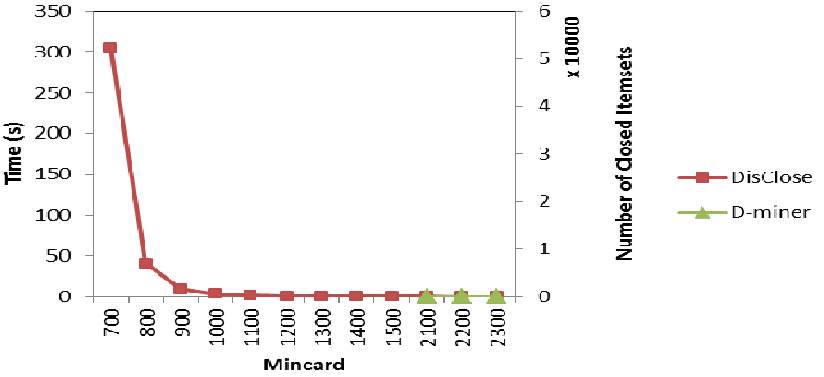
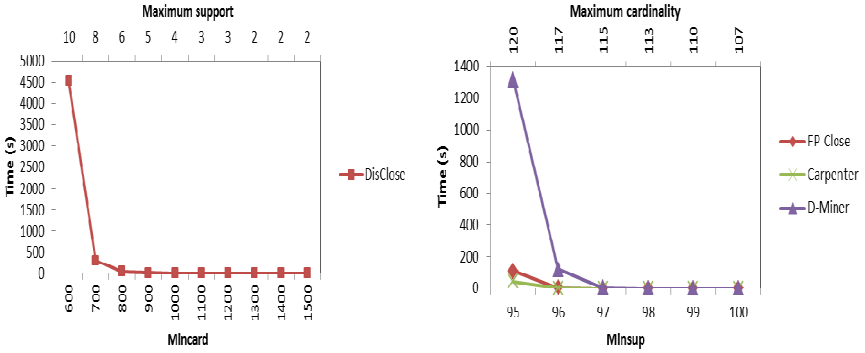


Fig. 3. Comparison with *D-Miner* using *mincard* threshold on T100L2000N4000

Fig. 3 shows the result of the performance between *DisClose* and *D-Miner*. It is observed that at a higher cardinality threshold, the difference in the time taken between the two algorithms is very small. However, as the *mincard* value decreases, *DisClose* largely outperforms *D-Miner*. Taking the maximum processing time of around 300 seconds, *DisClose* is able to discover colossal closed itemset with *mincard* = 700. For *D-Miner*, after *mincard* = 2100, the algorithm took more than 12 hours to discover the colossal closed itemsets.

As shown in Fig. 4(a), beginning with the largest closed itemsets, *DisClose* is able to discover the colossal closed itemsets with a maximum support of 10. The performance of *DisClose* sharply increases between *mincard* = 700 and *mincard* = 600. This is due to the large number of closed itemsets that exists between these thresholds. However, Fig. 4(b) shows that the algorithms are only able to reach *minsup* = 95 with the largest cardinality itemset of 120.



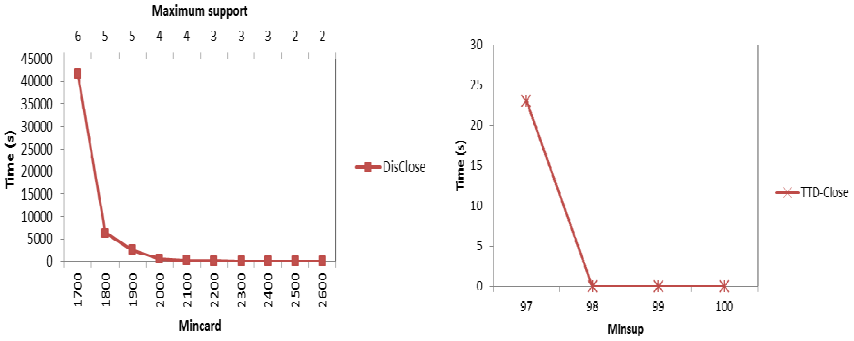
(a) (b)

Fig. 4. Comparison with *FP-Close*, *CARPENTER* and *D-Miner* on T100L2000N4000

As the nature of *TTD-Close* is to read each item in the dataset as a value, the largest itemset that exists in the dataset is equivalent to its column value. Therefore, in this particular case, more colossal closed itemsets are discovered.

Fig. 5(a) shows that as the *mincard* value increases, the time required to discover the colossal closed itemsets also increases. *DisClose* is able to reach closed itemsets with *mincard* = 1700. There are a total of 78,717,638 closed itemsets that exists when *mincard* = 1700 having the maximum support of 6. This shows that for dense dataset, even at a high *minsup* threshold, the size of itemsets can become very large.

Fig. 5(b) shows that *TTD-Close* could only reach *minsup* = 97 with a total of closed itemsets of 58,505. *TTD-Close* runs out of memory probably due to the existence of larger cardinality itemsets at smaller *minsup* thresholds. This shows that for dense dataset, even at a high *minsup* threshold, the size of itemsets can become very large.



(a) (b)

Fig. 5. Comparison with *TTD-Close* on T100L2000N4000

8 Conclusions and Future Work

This paper has introduced an algorithm *DisClose* that searches for colossal closed itemsets from the largest cardinality itemsets that exist in the dataset. This search is integrated with the bottom-up row enumeration strategy. We propose *mincard* to further reduce the search space. The closedness-checking method used reduces the need to check whether a newly discovered closed itemset already exists. The new approach bypasses the huge number of small-cardinality, high-frequency closed itemsets and focuses on the potentially most valuable large closed itemsets.

The results show that the algorithm exhibit scalable performance with run-time being almost correlated with closed itemset count. *DisClose*'s performance appears to be linear with respect to the number of closed itemsets. This suggests there may be a relationship between the colossal closed itemsets identified and their corresponding support sets. In particular, do any of the closed itemsets actually identify known classes of examples in the datasets?

Further evaluation of the algorithm on real datasets is also required in order to calibrate behavior and efficiency.

Acknowledgments. We would like to thank to the authors of *D-miner* and *TTD-Close* for providing the executable. We would also like to thank to Christian Borgelt for providing the implementation codes for *FP-Close* and *CARPENTER* through his website.

References

1. Pan, F., Cong, G., Tung, A.K.H., Yang, J., Zaki, M.J.: CARPENTER: Finding closed patterns in long biological datasets. In: Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003), pp. 637–642. ACM (2003)
2. Liu, H., Wang, X., He, J., Han, J., Xin, D., Shao, Z.: Top-down mining of frequent closed patterns from very high dimensional data. Information Science 179(7), 899–924 (2009)
3. Zhu, F., Yan, X., Han, J., Yu, P.S., Cheng, H.: Mining colossal frequent closed patterns by core pattern fusion. In: Proc. International Conference on Data Engineering (ICDE 2007), pp. 706–715. IEEE (2007)
4. Bayardo, R.J.: Efficiently mining long patterns from databases. In: Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD 1998), pp. 85–93. ACM, New York (1998)
5. Grahne, G., Zhu, J.: Efficiently using prefix-trees in mining frequent itemsets. In: Proc. 1st IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2003), pp. 123–132 (2003)
6. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15(1), 55–86 (2007)
7. Rioult, F., Boulicaout, J., Cremilleux, B., Besson, J.: Using transposition for pattern discovery from microarray data. In: Proc. 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2003), pp. 73–79. ACM (2003)
8. Cong, G., Tan, K.-L., Tung, A., Pan, F.: Mining Frequent Closed Patterns in Microarray Data. In: Proc. Fourth IEEE Int'l Conf. Data Mining (ICDM), vol. 4, pp. 363–366 (2004)
9. Besson, J., Robardet, C., Boulicaout, J.-F., Rome, S.: Constraint-based mining and its application to microarray data analysis. Intelligent Data Analysis Journal 9(1), 59–82 (2005)
10. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 1–12 (2000)