DisClose: Discovering Colossal Closed Itemsets from High Dimensional Datasets via a Compact Row-Tree

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Engineering and Physical Sciences.

2012

Nurul Fariza Zulkurnain
School of Computer Science
# Table of Contents

List of Tables ............................................. 6  
List of Figures ........................................... 7  
Abbreviations & Acronyms ................................. 10  
Declaration .................................................. 12  
Copyright .................................................... 13  
Chapter: 1 ..................................................... 16  
**Introduction** .............................................. 16  
1.1 Research Motivation ................................... 18  
1.1.1 Challenges in Itemset Mining ....................... 19  
1.2 Aims and Objectives ................................... 22  
1.3 Contributions ........................................... 23  
1.4 Organization of the Thesis ............................ 24  
Chapter: 2 ..................................................... 26  
**Itemset Mining Preliminaries** .......................... 26  
2.1 Frequent Itemset in Association Rule Mining .......... 27  
2.2 Alternatives Approaches to Frequent Itemset Mining .... 31  
2.2.1 Maximal Frequent Itemset Mining .................. 31  
2.2.2 Closed Frequent Itemset Mining .................... 33  
2.3 \textit{CFI} in High-Dimensional Dataset ................. 35  
2.3.1 Transposition method .............................. 36  
2.3.2 \textit{CFI} Mining on Transposed Data ............... 37
Chapter: 3

Strategies for Closed Frequent Itemset Mining

3.1 Column Enumeration-based Strategy ............................................. 41
   3.1.1 Apriori-based Bottom-up Search ............................................. 41
   3.1.2 Pattern Growth without Candidate Generation ......................... 46
   3.1.3 Exploring the Vertical Data Format ......................................... 52
3.2 Row Enumeration-based Strategy .................................................. 54
   3.2.1 Bottom-up Search .................................................................... 54
   3.2.2 Top-down Search .................................................................... 59
3.3 Pattern-Fusion: Mining the Colossal Itemsets .................................... 63
3.4 D-Miner: Mining the Constraint-based Concept .................................. 65
3.5 Summary .......................................................................................... 67

Chapter: 4

DisClose: Mining Colossal Closed Itemsets

4.1 Overview .......................................................................................... 71
4.2 Proposed Search Strategies ............................................................ 72
   4.2.1 Bottom-up Row-enumeration Search ........................................ 72
   4.2.2 Transposed Table ..................................................................... 74
   4.2.3 Minimum Cardinality Threshold, mincard .................................. 75
4.3 Proposed Closedness-checking Method ............................................ 78
4.4 CR-Tree (Compact Row-Tree) .......................................................... 82
4.5 Algorithm DisClose .......................................................................... 85
   4.5.1 Major steps of DisClose ........................................................... 85
   4.5.2 Space and time analysis of DisClose .......................................... 95
Chapter: 5  97

Experimental Evaluation  97

5.1 Experimental Setting ................................................................. 98
   5.1.1 Challenges in Comparisons .................................................. 99

5.2 Synthetic Datasets ....................................................................... 100
   5.2.1 Dimensionality (columns) ...................................................... 101
   5.2.2 Number of rows .................................................................... 110
   5.2.3 Cardinality ............................................................................ 117

5.3 Discussions of Results from Synthetic Datasets ............................. 124

5.4 Real Datasets ................................................................................ 126
   5.4.1 Discretization ......................................................................... 126
   5.4.2 DLBCL Data ........................................................................... 127
   5.4.3 Leukemia Data ....................................................................... 131
   5.4.4 Lung Cancer Data ................................................................. 134
   5.4.5 MLL Data .............................................................................. 137
   5.4.6 Memory consumption ........................................................... 139

5.5 Discussion on Results on Real Datasets ......................................... 143

5.6 Summary ..................................................................................... 144

Chapter: 6  145

Conclusions and Future Work  145

6.1 Summary ....................................................................................... 145

6.2 Research Contributions ............................................................... 148
   6.2.1 Search Strategy for Colossal Closed Itemsets ......................... 149
   6.2.2 Implementation of Constraint Measure ..................................... 149
   6.2.3 Closedness-checking Method ............................................... 150
   6.2.4 Data Structure – CR-Tree ...................................................... 150
6.2.5 Algorithm to Discover Colossal Closed Itemsets ............................... 151

6.3 Future Work ........................................................................................................ 151

Bibliography .............................................................................................................. 155

Word Count = 32, 957
List of Tables

2.1: Example of transactional dataset, \( T \) ................................................................. 28
2.2: Example of discretized microarray dataset, \( T_m \) ................................................. 36
2.3: Transposed table \( T^t \) of microarray dataset, \( T_m \) ............................................. 37

3.1: Conditional pattern base and conditional FP-trees for each suffix item........ 48
3.2: Example of the vertical data format for transactional dataset, \( T \) .............. 52
3.3: Conditional transposed table, \( T^t_{[2, 3, 4]} \) ................................................................. 56
3.4: Example of the binary representation of dataset................................................. 66

4.1: Example of a discretized high-dimensional dataset................................. 74
4.2: Example of Transposed Dataset........................................................................ 75

5.1: Datasets used in experiments showing the number of instances (rows) and
    number of columns (attributes)........................................................................ 126
## List of Figures

1.1: Steps of the KDD Process

2.1: Frequent itemsets for $minsup = 3$

2.2: Maximal frequent itemsets for $minsup = 3$

2.3: Closed frequent itemsets for $minsup = 3$

3.1: Bottom-up column enumeration tree

3.2: Discovering closed itemsets with $A$-Close for $minsup = 3$

3.3: The $FP$-tree from Table 2.1 for $minsup = 3$

3.4: Bottom-up row enumeration tree

3.5: Top-down row enumeration tree

3.6: $FR$-Tree and $IP$-List

3.7: Discovering the concepts using $D$-Miner

4.1: Colossal Closed Itemset Mining Procedure

4.2: Top-down column enumeration tree

4.3: Example of colossal itemsets (highlighted in grey) for $mincard = 7$

4.4: Relationship between $CR$-Tree and $T'$

4.5: Nodes representing the rowsets in the $CR$-Tree

4.6: Itemset stored at level, $l = 0$, of the $CR$-Tree

4.7: Example of Optimization strategy $S1$

4.8: Example on Lemma 4.2

4.9: Example on Optimization strategy $S2$

4.10: Example on Lemma 4.3

4.11: Example of Pruning strategy $S3$

4.12: Closedness-checking in $CR$-Tree

4.13: Colossal closed itemsets with $mincard = 7$
5.1: The effect of changing dimensionality with mincard .......................... 102
5.2: Comparison on T100L2000N4000 with FP-Close, CARPENTER and D-Miner .......................................................................................................................... 104
5.3: Comparison on T100L2000N6000 with FP-Close, CARPENTER and D-Miner .......................................................................................................................... 105
5.4: Comparison on T100L2000N10000 with FP-Close, CARPENTER and D-Miner ......................................................................................................................... 107
5.5: Comparison on T100L2000N4000 with TTD-Close .............................. 108
5.6: Comparison on T100L2000N6000 with TTD-Close .............................. 109
5.7: Comparison on T100L2000N10000 with TTD-Close ............................ 111
5.8: The effect of changing the number of tuples on the runtime ............... 112
5.9: Comparison on T150L2000N4000 with FP-Close, CARPENTER and D-Miner ......................................................................................................................... 114
5.10: Comparison on T200L2000N4000 with FP-Close, CARPENTER and D-Miner ......................................................................................................................... 115
5.11: Result on T150L2000N4000 with TTD-Close ..................................... 116
5.12: Result on T200L2000N4000 with TTD-Close ..................................... 116
5.13: The effect of changing cardinality on the runtime ............................. 118
5.14: Comparison on T100L1500N4000 with FP-Close, CARPENTER and D-Miner ......................................................................................................................... 119
5.15: Comparison on T100L2500N4000 with FP-Close, CARPENTER and D-Miner ......................................................................................................................... 121
5.16: Comparison on T150L1500N4000 with TTD-Close ............................ 122
5.17: Result on T150L2500N4000 with TTD-Close ..................................... 123
5.18: Comparison of mincard thresholds on DLBCL with D-Miner .......... 128
5.19: Comparison on DLBCL with FP-Close, CARPENTER and D-Miner ...... 129
5.20: Comparison on DLBCL with TTD-Close ........................................... 130
5.21: Comparison of mincard thresholds on Leukemia with D-Miner ........ 131
5.22: Comparison on Leukemia with FP-Close, CARPENTER and D-Miner ... 132
5.23: Comparison on Leukemia with TTD-Close ........................................ 133
5.24: Comparison of mincard thresholds on Lung with D-Miner ............... 134
5.25: Comparison on Lung with FP-Close, CARPENTER and D-Miner ....... 135
5.26: Comparison on Lung with TTD-Close ............................................. 136
5.27: Comparison of mincard thresholds on MLL with D-Miner ............... 137
5.28: Comparison on MLL with FP-Close, CARPENTER and D-Miner ........ 138
5.29: Comparison on MLL with TTD-Close ............................................... 140
5.30: Memory usage comparison using the maximum mincard and minsup
       threshold.......................................................................................... 141
# Abbreviations & Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Acute Lymphoblastic Leukemias</td>
</tr>
<tr>
<td>AML</td>
<td>Acute Myeloid</td>
</tr>
<tr>
<td>CCI</td>
<td>Colossal Closed Itemset</td>
</tr>
<tr>
<td>CFI</td>
<td>Closed Frequent Itemset</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DLBCL</td>
<td>Diffuse Large B-Cell Lymphomas</td>
</tr>
<tr>
<td>DNA</td>
<td>Deoxyribonucleic Acid</td>
</tr>
<tr>
<td>FIM</td>
<td>Frequent Itemset Mining</td>
</tr>
<tr>
<td>FIMI</td>
<td>Frequent Itemset Mining Implementations</td>
</tr>
<tr>
<td>FL</td>
<td>Follicular Lymphomas</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>GHz</td>
<td>Gigahertz</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>ID</td>
<td>identity document</td>
</tr>
<tr>
<td>K</td>
<td>Kilo (10³)</td>
</tr>
<tr>
<td>KDD</td>
<td>Knowledge Discovery in Databases</td>
</tr>
<tr>
<td>MFI</td>
<td>Maximal Frequent Itemset</td>
</tr>
<tr>
<td>MLL</td>
<td>Mixed-Lineage Leukemias</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
</tbody>
</table>
Abstract

Data mining is an essential part of knowledge discovery, and performs the extraction of useful information from a collection of data, so as to assist human beings in making necessary decisions. This thesis describes research in the field of itemset mining, which performs the extraction of a set of items that occur together in a dataset, based on a user specified threshold. Recent focus of itemset mining has been on the discovery of closed itemsets from high-dimensional datasets, characterised by relatively few rows and a relatively larger number of columns. A closed itemset is the maximal set of items common to a set of rows. By exponentially increasing running time as the average row length increases, mining closed itemsets from such datasets renders most column enumeration-based algorithm impractical. Existing row enumeration-based algorithms also show that they struggle to reach large cardinality closed itemsets. This is due to the implementation of the support constraint, which is based on the frequency of occurrence of the itemset. Frequent closed itemsets are usually smaller in size and larger in numbers, hence taking much of the memory space. Unfortunately, large cardinality closed itemsets are likely to be more informative than small cardinality closed itemsets in this type of dataset.

The research investigates the area of large cardinality closed itemset discovery by examining and analysing the literature and identifying both strengths and weaknesses of existing approaches. Based on this synthesis, a new algorithm, termed DisClose, has been designed and developed to discover large cardinality (colossal) closed itemsets from high-dimensional datasets. The algorithm strategy begins by enumerating large cardinality itemsets and from these, builds smaller itemsets. This is done by applying a bottom-up search of the row-enumeration tree. A minimum cardinality threshold has been proposed to identify colossal closed itemsets and to further reduce the search space. A novel closedness-checking method has been proposed which uses a unique generator to immediately discover closed itemsets without the need to check if each new closed itemset has previously been found. These approaches have been combined using a Compact Row-Tree (CR-Tree) data structure designed to assist in the efficient discovery of the colossal closed itemsets. For evaluation purposes four state-of-the-art algorithms have been selected for comparison. Experimental results show that algorithm DisClose is scalable and can efficiently extract colossal closed itemsets in the considered dataset, even for low support thresholds that existing algorithms cannot find.
Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1998 (as amended) and regulations issued under it or , where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://www.campus.manchester.ac.uk/medialibrary/policies/intellectual_property.pdf), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www/manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
To my husband, Khairul Fadli,
for the love and sacrifices

To my children, Nabil Iman and Nadwa Iman,
for the constant hugs and kisses
Acknowledgements

Thank you God Almighty for giving me all the guidance and strength throughout this initial journey of my research. To my parents, Zulkurnain Awang and Wan Norasiah Che Jaafar, for the never ending love and support.

I would like to express my greatest appreciation and gratitude to my supervisor, Professor John A. Keane whom guidance, advice and support have enabled this thesis to be completed successfully. A special thanks to Professor David J. Haglin from Pacific Northwest National Laboratory, United States, for his idea that initiated this research.

I am truly grateful to the data mining research community for their knowledge and suggestions especially those who have provided the executable codes in order to substantially improve the research work. A special thanks to Mohammed Saree from the University of Salford for providing the IBM Data Generator executable.

It was a pleasure to work with my colleagues and the many people in the University of Manchester, especially from the School of Computer Science. Their supports and ideas in many occasions have helped me progress. I am also indebted to my Malaysian friends in Manchester, for their friendship and support in making mine and my family’s lives here vibrant and memorable. I would like to acknowledge my employer, International Islamic University Malaysia and the Government of Malaysia for the financial funding in which without it I would not be able to make it here in the first place. Not willing to miss anyone, I would like to express my gratitude to every single individual who has contributed directly and indirectly, towards completing my studies.

Finally, I honour the sacrifice of my beloved family who has continuously without doubt provided me with ample moral support and consistently encouraged me to complete this thesis.
Chapter: 1

Introduction

_Data mining is nothing else than torturing the data until it confesses... and if you torture it enough, you can get it to confess to anything_ (Fred Menger)

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 serves 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.

Data mining is the analysis step of _knowledge discovery in databases_ (KDD) process (Fayyad et al., 1996). The steps in the KDD process are shown in Figure 1.1. Data mining is defined as ‘the analysis of (often large) observational datasets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner’ (Hand et al., 2001).
CHAPTER 1. INTRODUCTION

There are numerous data mining tasks, which include the discovery of association rules (Agrawal et al., 1993), sequential rules (Agrawal and Srikant, 1995), correlations (Brin et al., 1997), episodes (Mannila et al., 1997), multi-dimensional patterns (Lent et al., 1997), maximal patterns (Bayardo, 1998) and various other discovery tasks (Han & Kamber, 2001).

This thesis will focus on the task of association rule discovery. Association rules (Agrawal et al., 1993) aim to describe noteworthy relationships between variables. Agrawal et al. (1993) introduced the itemset mining problem as part of association of rule discovery. An itemset is a collection of related items that occur together in a given dataset. This initial research was motivated by analysis of market basket (transactional) data. Given a transactional dataset, the aim is to identify all items which have been bought together most often. The set of items is represented by the customer’s transaction IDs. The results obtained help to generate association rules. This should then assists companies in better understanding of the purchasing behavior of customers, which should in turn help to improve decision making about marketing activities.

In addition to market basket analysis, discovery of association rules has been employed in many other areas. These include: telecommunications (detecting intrusion in networks or system activities (Zhong and Qin, 2004; Patcha and Park,
CHAPTER 1. INTRODUCTION

2007; Vaarandi and Podins, 2010)), bioinformatics (generating new knowledge in biology and medicine (Creighton and Hanash, 2003; Pan et al., 2003; Guns et al., 2010)) and web usage analysis (discovering patterns from the web (Eirinaki and Vazirgiannis, 2003; Youssefi et al., 2004; Baraglia and Silvestri, 2007)).

Generating association rules is a rather straightforward, computationally inexpensive part of the discovery task. Since the area was initially proposed, the focus of researchers and scientist has mostly been on optimizing the itemset mining process. In this thesis, a new method has been developed to efficiently mine large cardinality itemsets that exist in very large datasets. It is anticipated that the method will assist the discovery of association rules from large cardinality itemsets in this type of dataset.

1.1 Research Motivation

A typical business transaction dataset for market basket analysis contains a relatively large number of rows (transactions) compared to a relatively small number of columns (dimensions). However, other application areas such as gene expression matrices analysis in bioinformatics (Creighton and Hanash, 2003; Cong et al., 2004b; Borgelt et al., 2011) and text processing (Nahm and Mooney, 2001; He et al., 2011; Nassem, 2012) involve another kind of dataset, one which is characterized by a relatively small number of rows compared with a relatively large number of columns (or dimensions). Due to these features, this is known as a high-dimensional dataset.

The opportunities created by high-dimensional datasets are significant. Such datasets have also attracted interest from researchers to devise new methods to effectively extract significant information. The amount of information that can be revealed is potentially huge, but extracting information, and ultimately knowledge, from these datasets is a non-trivial task.

Applications that deal with high-dimensional datasets include:
CHAPTER 1. INTRODUCTION

- Discovering relationships between the data values within gene expression matrices or microarray datasets, in order to assist in understanding the cause and effect of biological processes. For instance, such relationships can help in predicting gene function for uncharacterized genes based on the similarity of their expression profiles to those of known genes (Brown et al., 2000); identifying genes that are important in specific cellular processes, diseases, or in cell differentiations (Segal et al., 2003); learning about gene regulation by finding and studying groups of regulated genes (Carmona-Saez et al., 2006); and finding how cells respond to various compounds, and then classifying predictions of responses by new compounds (Huang et al., 2009).

- Sorting a set of documents automatically into categories from a pre-defined set in order to increase connectivity and availability of the documents at all levels of the information chain. This is also known as text categorization (Sebastiani, 2002). Text categorization can be used to identify document genre (Bhattacharya et al., 2008), automated population of hierarchical catalogues of web resources (Golub and Lykke, 2009), indexing scientific articles according to predefined thesauri of technical terms (Renear and Palmer, 2009), and authorship attribution (Stamatatos, 2009).

1.1.1 Challenges in Itemset Mining

The primary issue in itemset mining is to efficiently and effectively discover the complete set of itemsets in a dataset with respect to a given user-defined threshold. However, the presence of an itemset of length \( k \) also implies the presence of \( 2^k - 2 \) additional itemsets. It is to be noted that given the number of itemsets with a large \( k \), enumerating the entire collection of itemsets has been demonstrated to be unfeasible for most algorithms. This is true especially when dealing with real-life datasets,
where the dataset size may be very large (Pan et al., 2003; Zhu et al., 2007; Liu et al., 2009). The physical limitation of real memory space results in an inability to store all the itemsets discovered. Further, enumerating the entire collection of itemsets also has an effect on processing costs.

Besides scalability (or lack thereof), due to sheer size, discovered itemsets are difficult to interpret. This is called the information overload problem which has several side effects. For example, large itemsets increase the time and space complexity of the mining task. The complexity of the mining task is exponential with respect to the number of dimension (column) because of the notorious curse of dimensionality effect (Wang and Yang, 2010). Moreover, it is likely that there is a considerable amount of overlap between itemsets.

Most strategies proposed to overcome these challenges have involved reducing the amount of output. These include finding only maximal itemsets (Bayardo, 1998) or finding only closed itemsets (Pasquier et al., 1999). An itemset is a maximal itemset if there is no immediate superset of the itemset. On the other hand, an itemset is a closed itemset if there is no proper superset with the same row (transaction) values. More formal definitions and examples of these two terms will be given in the next chapter. Depending on the dataset, maximal or closed itemsets can offer significant compression. Nevertheless, most algorithms opt to discover closed itemsets, due to their ability to provide a compact version of itemsets without information loss (Pasquier et al., 1999). However, for a high-dimensional dataset, the size of the solutions (i.e. collection of extracted closed itemsets) is still too large to deal with the threshold value (Rioult et al., 2003).

Numerous algorithms have been proposed to mine closed itemsets on transactional data (Bayardo, 1998; Wang et al., 2003; Zaki and Gouda, 2003). These algorithms usually search the itemset space of the dataset; therefore, they are termed column enumeration-based algorithms. The method works well for datasets with small average row length, as if $i$ is the maximum row size, there could be $2^i - 1$
CHAPTER 1. INTRODUCTION

potential itemsets\(^1\); usually \(i < 100\). On the contrary, a high-dimensional dataset contains a large number of items (columns). The run time for a columnenumeration based search strategy increases exponentially with an increasing average row length, which results in poor performance. As a consequence, a search over the entire itemset space is impractical.

There are a group of related algorithms that attempt to overcome the limitation of column-based search by enumerating the dataset in a row-wise manner (Pan et al., 2003; Cong et al., 2004b; Liu et al., 2009). These kinds of algorithms are termed *row enumeration-based* algorithms. However, the majority of these algorithms begin their search for closed itemsets that occur from the largest row (transaction) values. The number of closed itemsets that exists at the larger end of the row values tends to be small in size and bigger in number. As a result, it takes much memory space to store these many small closed frequent itemsets, thus making the proposed algorithms computationally infeasible to reach the large closed frequent itemsets. This is true especially for large and dense datasets such as high-dimensional ones. It could face the risks of overseeing significant patterns.

In association mining tasks, itemsets that are bigger in size are usually of greater importance, especially in domains such as bioinformatics, as bigger itemsets tend to be more informative compared to small ones (Zhu et al., 2007; Han et al., 2007). Closed itemsets that are bigger in size can also be referred to as large cardinality closed itemsets. The term ‘*cardinality*’ refers to the measurement of the number of elements (items) that contain in a set. These large cardinality closed itemsets are called *colossal itemsets*, in order to distinguish them from closed itemsets with a large number of rows (Zhu et al., 2007).

Section 2.3 of the survey paper by Han et al. (2007) also contends that the main challenge in mining closed itemsets is to ensure whether a pattern mined is closed. Several existing closed itemset mining algorithms require the dataset to be

\(^{1}\) In mathematics, given a set \(S\), the **powerset** of \(S\) (\(2^S\)), is the set of all subsets of \(S\).
checked repeatedly to see if an itemset is closed. Repeated checking for closed itemsets within the dataset or the result set lead to an increase in processing time.

1.2 Aims and Objectives

Existing algorithms do not address the challenges stated to find useful large cardinality itemsets; yet relatively large (colossal) closed itemsets in high-dimensional datasets can provide valuable insights into the meaning of the datasets (Zhu et al., 2007; Han et al., 2007). The main hypothesis of our research is that such itemsets can be derived efficiently by using a strategy that begins the search from the largest itemset and progressively builds smaller itemset. This approach is supported by applying an effective closedness-checking method as well as a compact data structure.

The aim of this thesis is to efficiently discover all large cardinality closed itemsets that exist in high-dimensional datasets, based on a user-defined cardinality threshold.

To achieve this aim, the following objectives are identified by means of several research questions:

- The row-enumeration search strategy has addressed the problem of discovering closed itemsets from high-dimensional dataset. However, the majority of approaches identify the closed itemsets as beginning from the most frequent. Frequent itemsets generally tend to be smaller in size.
  
  To bypass these small frequent itemsets, can the search begin with the largest cardinality itemsets?

- By using the support threshold, the existing closed itemset mining algorithms limit the search space, based on the occurrence of closed itemsets in the dataset.
  
  To identify large cardinality itemsets, is it possible to identify a method that can acquire and utilise an alternative threshold other than the support threshold?
CHAPTER 1. INTRODUCTION

- To reduce the memory space and processing time, can the closed itemsets be identified during the search thus avoiding the necessity of having to check whether the closed itemsets have already been discovered?
- Generating candidates in order to discover large cardinality closed itemsets can necessitate usage of more memory space and increased computation time. This adds up if these candidates are not closed itemsets in the original dataset.
  *Are there ways to avoid generating unnecessary candidate itemsets thus reducing memory space usage and computation time?*
- Existing algorithms begin their search from the most frequent itemsets. Large cardinality itemsets usually exist at the infrequent end of the support spectrum.
  *Is there an efficient way to represent the results in order to compare and demonstrate the strengths and weaknesses of any proposed new method?*

### 1.3 Contributions

This thesis studies itemset mining, particularly the mining of closed itemsets from high dimensional data. In particular, the following contributions are made.

- Direct extraction of large cardinality closed itemsets by avoiding the search for small cardinality closed itemsets in high-dimensional data.
- An alternative threshold is introduced to reduce the search space and identify the large closed itemsets based on their cardinality.
- A closedness-checking method to check whether an itemset is closed. The method identifies large closed itemsets without the need for repeated checking from the result set.
- A compact row-tree based *(CR-Tree)* data structure, which integrates the proposed techniques to provide a compact representation of the dataset and search space.
CHAPTER 1. INTRODUCTION

- Two effective optimization strategies to reduce the generation of candidate itemsets, in order to speed up the search process in the CR-Tree.
- An algorithm to discover large cardinality closed itemsets which also represents sufficient support values of the closed itemsets that other algorithms were not able to reach from high-dimensional datasets.
- An experimental study on both synthetic and real world datasets to compare the performance of the proposed algorithm with selected state-of-the-art algorithms.

Part of the work presented in this thesis has been published as:


1.4 Organization of the Thesis

The remainder of this thesis is structured as follows:

**Chapter 2: Itemset Mining Preliminaries** presents a systematic overview of itemset mining in relation to association rule discovery. The chapter begins by defining frequent itemsets in association rule mining. The types of frequent itemsets and the context of itemset mining in high-dimensional datasets are presented.

**Chapter 3: Strategies for Closed Itemset Mining** begins by identifying limitations of general itemset mining when applied to high-dimensional datasets. Following this, work specifically developed for mining closed itemsets in high-dimensional...
datasets is considered. Advantages and disadvantages in existing closed itemset mining approaches for high-dimensional datasets are discussed.

Chapter 4: DisClose: Mining Colossal Closed Itemsets describes the development of closed itemset mining in high-dimensional datasets through the proposed algorithm, DisClose. It also addresses many of the research questions articulated in Section 1.2. A detailed description of the mining process is given which includes the proposed search strategy, the search threshold, closedness-checking method and data structure.

Chapter 5: Experimental Evaluation presents a performance study of the DisClose algorithm. The chapter begins by providing the experimentation environment and the datasets chosen. It also presents the test results obtained by comparing DisClose with selected state-of-the-art algorithms. This is followed by an analysis discussion of the results to assess the capability of DisClose and provide the basis for future work.

Chapter 6: Conclusions and Future Work discusses and summarizes the research contributions and their limitations. Suggestions and proposals for future work are also provided.
Chapter: 2

Itemset Mining Preliminaries

In this chapter, a systematic overview of itemset mining in relation to association rule discovery will be provided; with the aim of understanding the concept. This includes the terms and definitions that will provide the foundation for the remaining part of the research presented in this thesis.

The chapter begins with Section 2.1, which introduces association rule mining and its role. This is followed by the definition of a frequent itemset by providing examples from simple transactional data. An illustration of frequent itemsets discovered from transactional data is also presented.

Section 2.2 presents and defines two alternative approaches that have been proposed in order to overcome the problem of mining all frequent itemsets - maximal frequent itemset and closed frequent itemset.

Section 2.3 provides the notion of dimensionality in the context of datasets, and in particular, high-dimensional data. The transposition method is presented, which was proposed in order to reduce the complexity of the search for closed frequent itemsets in high-dimensional data.

Section 2.4 summarizes the chapter.
CHAPTER 2. ITEMSET MINING PRELIMINARIES

2.1 Frequent Itemset in Association Rule Mining

The application of association rule mining has been widely used in order to discover interesting relationships between variables in large datasets. Association rule mining, as first proposed by Agrawal et al. (1993), examines the behaviour of customers in terms of the products (items) they often purchase together in a shop visit (transaction). The collection of data stored is known as a transactional dataset.

Let $T$ be a dataset table that consists of a collection of rows (transactions), $R = \{r_1, r_2, \ldots, r_m\}$ and a list of items, $I = \{o_1, o_2, \ldots, o_n\}$. This set of transactions represents the number of rows ($m$) and the set of items signifies the number of columns ($n$) in $T$.

A nonempty subset $\alpha \subseteq I$ is called an itemset. An itemset, $a_k$, which consists of $k$ items, is described as a $k$-itemset. Each transaction $r_i$ is represented by a unique 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 2.1 (Table $T$) Table 2.1 illustrates an example of a transactional dataset, $T$, that contains six rows and five items, so $R = \{1, 2, 3, 4, 5, 6\}$ and $I = \{a, b, c, d, e\}$.

Definition 2.1 (Support Set) For any itemset $\alpha$, the support set is represented as the set of rows in the dataset, $T$, that contains $\alpha$. This is represented as:

$$r(\alpha) = \{r_i \mid \alpha \subseteq t(r_i)\} \tag{2.1}$$

Example 2.2 (Support Set) In Table 2.1, for an itemset $\alpha = \{a, b, d, e\}$, the support set $r_\alpha = \{1, 3, 5\}$.

Definition 2.2 (Support) The support of an itemset $\alpha$ is the number of rows in which $\alpha$ occurs in $T$ – denoted as $|r_\alpha|$. 

27
Example 2.3 (Support) From Example 2.2, the support for itemset $\alpha = \{a, b, d, e\}$, $|r_\alpha| = |\{1,3,5\}| = 3$.

Definition 2.3 (Frequent Itemset) Given a dataset $T$ and a minimum support threshold $\text{minsup}$, an itemset $\alpha$ is frequent if $|r_\alpha| \geq \text{minsup}$.

Example 2.4 (Frequent Itemset) Suppose the user would like to identify items that occur in at least three of the transactions. A total of 19 frequent itemsets were found from Table 2.1 for $\text{minsup} = 3$, this is presented in Figure 2.1 where each itemset is shown along with its rowset.

An association rule is an implication of the form $\alpha_1 \Rightarrow \alpha_2$, where $\alpha_1$ and $\alpha_2$ are itemsets and $\alpha_1 \cap \alpha_2 = \emptyset$. The strength of an association rule is mainly measured by support and confidence.


CHAPTER 2. ITEMSET MINING PRELIMINARIES

<table>
<thead>
<tr>
<th>Support</th>
<th>Itemset {rowset}</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>b {1, 2, 3, 4, 5, 6}</td>
</tr>
<tr>
<td>5</td>
<td>e {1, 2, 3, 4, 5}, be {1, 2, 3, 4, 5}</td>
</tr>
<tr>
<td>4</td>
<td>a{1, 3, 4, 5}, c {2, 4, 5, 6}, d {1, 3, 5, 6}, ab {1, 3, 4, 5}, ae {1, 3, 4, 5}, bc {2, 4, 5, 6}, bd {1, 3, 5, 6}, abe {1, 3, 4, 5}</td>
</tr>
<tr>
<td>3</td>
<td>ad {1, 3, 5}, ce {2, 4, 5}, de {1, 3, 5}, abd {1, 3, 5}, ade {1, 3, 5}, bce {2, 4, 5}, bde {1, 3, 5}, abde {1, 3, 5}</td>
</tr>
</tbody>
</table>

Figure 2.1: Frequent itemsets for \textit{minsups} = 3

**Definition 2.4** (Support of a rule) The rule \( \alpha_1 \Rightarrow \alpha_2 \) holds in a dataset \( T \) with \textit{support}, \textit{sup}, where:

\[
sup(\alpha_1 \Rightarrow \alpha_2) = \frac{|R(\alpha_1 \cup \alpha_2)|}{|R|}
\]

\textbf{Example 2.5} (Support of a rule) In Table 2.1, the support for the rule \( ab \Rightarrow de \):

\[
sup(ab \Rightarrow de) = \frac{|R(ab \cup de)|}{|R|} = \frac{3}{6} = 0.5
\]

**Definition 2.5** (Confidence of a rule) The rule \( \alpha_1 \Rightarrow \alpha_2 \) holds in the dataset \( T \) with \textit{confidence}, \textit{conf}, where:

\[
conf(\alpha_1 \Rightarrow \alpha_2) = \frac{|R(\alpha_1 \cup \alpha_2)|}{|R_{\alpha_1}|}
\]
CHAPTER 2. ITEMSET MINING PRELIMINARIES

Example 2.6 (Confidence of a rule) The confidence for a rule \( ab \Rightarrow de \) in Example 2.5:

\[
\text{conf} (ab \Rightarrow de) = \frac{|r(ab \cup de)|}{|r_{ab}|} = \frac{3}{4} = 0.75
\]

The aim of association rule mining is to find a complete set of rules that has support and confidence of no less than the user-specified thresholds. Frequent Itemset Mining (FIM) consists of the first part of association rule discovery, by identifying a set of items equal to or above a user specified minimum support threshold, \( \text{minsup} \). Following this, the confidence of all rules that can be formed from the frequent itemsets can be calculated.

Therefore, association rule discovery may be divided into two parts (Agrawal et al., 1993):

1. Mining of all frequent itemsets in dataset \( T \) that have support that is greater than or equal to the user specified minimum support threshold, \( \text{minsup} \).
2. Generating association rules from each of the frequent itemsets discovered, with a confidence greater than or equal to the user specified minimum confidence threshold, \( \text{minconf} \).

Various methods have been introduced which focus on the efficient discovery of frequent itemsets. The problem of mining frequent itemsets is to find the complete set of frequent itemsets in a dataset, \( T \), with respect to a given support threshold \( \text{minsup} \). Extracting frequent itemsets is the most costly task of association rule mining; this is due to the fact that it requires enumerating all possible combinations of itemset. Once all frequent itemsets and their support are known, the association rule generation is straightforward.

However, the difficulty of mining the entire set of frequent itemsets is that the amount of frequent itemsets occurring in a dataset may be very large. Algorithms
developed in order to discover frequent itemsets have been shown to be inadequate when discovering frequent itemsets at the lower minimum support thresholds, or on datasets that contains long frequent itemsets (Agrawal et al., 1993; Han et al., 2000; Zaki, 2000a). This is because the presence of a frequent itemset of length $k$ implies the presence of $2^k - 2$ additional frequent itemsets as well. Therefore, generating and counting the supports of all frequent itemsets in the dataset cannot be achieved within a reasonable time. In addition, storing the complete set of frequent itemsets requires higher memory cost. Studies have shown that frequent itemsets contain much redundant information (Bayardo, 1998; Pasquier et al., 1999).

The next section discusses the two alternative approaches to frequent itemset mining that have been proposed to address these problems – maximal frequent itemset (Bayardo, 1998) and closed frequent itemset (Pasquier et al., 1999).

2.2 Alternatives Approaches to Frequent Itemset Mining

2.2.1 Maximal Frequent Itemset Mining

Maximal frequent itemset (MFI) mining was first proposed through an algorithm called MaxMiner (Bayardo, 1998). The advantage of mining maximal frequent itemset is that it has the ability to discover long frequent itemsets by providing a compact set of items from the dataset.

**Definition 2.6 (Maximal Itemset)** An itemset $\alpha$ is a maximal itemset in $T$ if there exists no immediate supersets $\alpha'$ where $\alpha' \in T$, such that $\alpha \subset \alpha'$.

**Definition 2.7 (Maximal Frequent Itemset)** An itemset is a maximal frequent itemset if none of its immediate supersets has support value equal or greater than minsup.

**Example 2.7 (Maximal Frequent Itemset)** Figure 2.2 shows the maximal frequent itemsets (highlighted in bold). **bce** and **abde** are the largest itemsets with no other
supersets discovered with \( \text{minsup} = 3 \), hence, they are the maximal frequent itemsets.

<table>
<thead>
<tr>
<th>Support</th>
<th>Itemset {rowset}</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>{1, 2, 3, 4, 5, 6}</td>
</tr>
<tr>
<td>5</td>
<td>{1, 2, 3, 4, 5}, {1, 2, 3, 4}</td>
</tr>
<tr>
<td>4</td>
<td>{1, 3, 4, 5}, {2, 4, 5, 6}, {1, 3, 5, 6}, {1, 3, 5}, {2, 4, 5, 6}, {1, 3, 5}, {1, 3, 5}, {1, 3, 5}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 3, 5}, {2, 4, 5}, {1, 3, 5}, {1, 3, 5}, {1, 3, 5}, {1, 3, 5}, {1, 3, 5}, {1, 3, 5}</td>
</tr>
</tbody>
</table>

Figure 2.2: Maximal frequent itemsets (highlighted in bold) for \( \text{minsup} = 3 \)

The number of maximal frequent itemsets is typically orders of magnitude fewer than the number of frequent itemsets. Hence, mining them is computationally less complex than mining all frequent itemsets (Bayardo, 1998; Lin and Kedem, 2002; Gouda and Zaki, 2001; Burdick et al., 2005). However, maximal frequent itemsets do not provide the complete subset frequency for generating association rules. As an example, taking the maximal itemset \( \text{abde} \{1, 3, 5\} \), based on the Definition 2.4, the support for the rule \( ab \Rightarrow de \) can be obtained since the frequency of occurrence for this itemset is known to be 3. Hence, the support of this rule is equal to \( (3/6) \) or 0.5. On the other hand, this is not true for rule \( ab \Rightarrow e \). Based on Figure 2.2 the support of this rule does not equal to \( (3/6) \) since the frequency of occurrence for this itemset is 4. Mining for maximal frequent itemsets does not produce \( \text{abe} \{1, 3, 4, 5\} \) given that \( \text{abe} \subseteq \text{abde} \), therefore it is unable to generate the rule.
CHAPTER 2. ITEMSET MINING PRELIMINARIES

Given that the consequences of mining maximal itemsets result in loss of information, further discussion on maximum frequent itemset mining will not be elaborated on further in the thesis (interested readers can refer to Bayardo (1998), Agarwal et al. (2000), Gouda and Zaki (2001), Lin and Kedem (2002), Burdick et al. (2005), Grahne and Zhu (2005) for further details).

2.2.2 Closed Frequent Itemset Mining

Closed Frequent Itemset (CFI) mining was proposed in order to overcome the problems of mining frequent itemsets and maximal frequent itemsets by removing itemsets that are not needed, and at the same time, being able to generate the complete set of association rules (Pasquier et al., 1999).

Definition 2.8 (Closed Itemset) An itemset $\alpha$ is a closed itemset in dataset $T$ if there is no proper superset $\alpha'$ ($\alpha \subsetneq \alpha'$) such that the support of $\alpha$ is the same as the support of $\alpha'$.

Closed itemsets are also the maximal set of items common to a rowset (Pasquier et al., 1999).

Definition 2.9 (Closed Frequent Itemset) An itemset $\alpha$ is a closed frequent itemset in dataset $T$ if $|r_{\alpha}| \geq \minsup$.

Example 2.8 (Closed Frequent Itemset) In Figure 2.3, there are 7 closed frequent itemsets discovered from Table 2.1 with $\minsup = 3$: $b \{1, 2, 3, 4, 5, 6\}$, $be \{1, 2, 3, 4, 5\}$, $bc \{2, 4, 5, 6\}$, $bd \{1, 3, 5, 6\}$, $abe \{1, 3, 4, 5\}$, $bce \{2, 4, 5\}$, and $abde \{1, 3, 5\}$. As can be observed, the closed itemsets discovered are the maximal set of itemsets amongst the itemsets of the same rowset value.

The closed itemset lattice is defined by employing a closure mechanism, based on the Galois connection, a theory of order and lattices (Davey and Priestley,
The closed itemset lattice is a sub-order of the itemset lattice; hence the search space is much smaller.

<table>
<thead>
<tr>
<th>Support</th>
<th>Itemset {rowset}</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>b {1, 2, 3, 4, 5, 6}</td>
</tr>
<tr>
<td>5</td>
<td>e {1, 2, 3, 4, 5}, be {1, 2, 3, 4, 5}</td>
</tr>
<tr>
<td>4</td>
<td>a{1, 3, 4, 5}, c {2, 4, 5, 6}, d {1, 3, 5, 6}, ab {1, 3, 4, 5}, ae {1, 3, 4, 5}, bc {2, 4, 5, 6}, bd {1, 3, 5, 6}, abe {1, 3, 4, 5}</td>
</tr>
<tr>
<td>3</td>
<td>ad {1, 3, 5}, ce {2, 4, 5}, de {1, 3, 5}, abd {1, 3, 5}, ade {1, 3, 5}, bce {2, 4, 5}, bde {1, 3, 5}, abde {1, 3, 5}</td>
</tr>
</tbody>
</table>

Figure 2.3: Closed frequent itemsets (highlighted in bold) for \textit{minsup} =3

The closed itemset lattice is used as a formal framework for discovering closed frequent itemsets, based on the following properties (Pasquier et al., 1999):

i. All subsets of a frequent itemset are frequent.

ii. All supersets of an infrequent itemset are infrequent.

iii. All subsets of a closed itemset of a frequent closed itemset are frequent.

iv. All supersets of a closed itemset of an infrequent closed itemset are infrequent.

v. The set of maximal frequent itemset is identical to the set of maximal frequent closed itemsets.
vi. The support of a frequent itemset \( \alpha \) which is not closed is equal to the support of the smallest frequent closed itemset containing \( \alpha \) (i.e. the closure of a frequent itemset is frequent).

Given that mining closed frequent itemsets limits the search space based on the closed itemset lattice, both the number of dataset passes and the CPU overhead incurred by frequent itemset searching decreases (Pasquier et al., 1999; Pei et al., 2000; Wang et al., 2003; Grahne and Zhu, 2005; Zaki and Hsiao, 2005). In addition, closed frequent itemsets are lossless, in the sense that they can produce a complete set of association rules from a much smaller set of frequent itemsets. Thus, the frequent itemset mining problem is reduced to the problem of determining closed frequent itemsets and their support.

2.3 \textit{CFI} in High-Dimensional Dataset

As was highlighted in Chapter 1, a typical business transaction dataset is represented with the characteristics of having a relatively large number of rows (transactions) and a relatively small number of columns (items). Recent interest has led to applying association rule mining to high-dimensional datasets. An example of such a dataset is the gene expression matrices or microarray data, where association rule mining is applied to discover significant relationships among different genes, based on expression levels (Tuzhilin and Adomavicius, 2002).

Contrary to a transactional dataset, high-dimensional datasets (microarray data) usually contains a relatively large number of columns (genes) and a relatively small number of rows (biological samples). Table 2.2 shows a simple example of a discretized microarray dataset, \( T_m \). The transaction IDs represents a set of patients and the items denote a set of genes. There are altogether 5 patients (rows) and nine groups of genes (items) in Table 2.2. Therefore, \( R = \{1, 2, 3, 4, 5\} \) and \( I = \{a_1, a_2, b_1, c_1, c_2, d_1, d_2, e_1, e_2\} \).
CHAPTER 2. ITEMSET MINING PRELIMINARIES

Table 2.2: Example of discretized microarray dataset, $T_m$

<table>
<thead>
<tr>
<th>Transaction id (Patients)</th>
<th>Items (Genes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a₁  a₂  b₁  c₁  c₂  d₁  d₂  e₁  e₂</td>
</tr>
<tr>
<td>1</td>
<td>1    0    1    1    0    0    1    0    1</td>
</tr>
<tr>
<td>2</td>
<td>0    1    1    0    1    0    1    0    1</td>
</tr>
<tr>
<td>3</td>
<td>1    0    1    0    1    1    0    0    1</td>
</tr>
<tr>
<td>4</td>
<td>0    1    1    0    1    0    1    0    1</td>
</tr>
<tr>
<td>5</td>
<td>1    0    1    0    1    1    0    1    0</td>
</tr>
</tbody>
</table>

2.3.1 Transposition method

The advantages of the closed-based technique, designed to handle pattern redundancy, have made it relatively common place in applying association rule discovery on high-dimensional datasets (Pan et al., 2003; Pan et al., 2004; Liu et al., 2006; Zhu et al., 2007; Liu et al., 2009).

Due to the complexity of searching for closed frequent itemsets, based on the number of columns in high-dimensional data, the transposition method was proposed (Rioul et al., 2003). The study states that by using the Galois connection (Davey and Priestley, 1994), the same results may be extracted from the transposed table by associating the sets of rows with the sets of columns, as per the original table, which associates sets of columns with sets of rows. Hence, the original dataset is transposed, so that each item (with different level of expressions) is now represented as a row value, and the rowset related to each row is represented as a column value.
CHAPTER 2. ITEMSET MINING PRELIMINARIES

Table 2.3: Transposed table $T^t$ of microarray dataset $T_m$

<table>
<thead>
<tr>
<th>Items</th>
<th>Tidset</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>{1, 3, 5}</td>
</tr>
<tr>
<td>a₂</td>
<td>{2, 4}</td>
</tr>
<tr>
<td>b₁</td>
<td>{1, 2, 3, 4, 5}</td>
</tr>
<tr>
<td>c₁</td>
<td>{1}</td>
</tr>
<tr>
<td>c₂</td>
<td>{2, 3, 4, 5}</td>
</tr>
<tr>
<td>d₁</td>
<td>{3, 5}</td>
</tr>
<tr>
<td>d₂</td>
<td>{1, 2, 4}</td>
</tr>
<tr>
<td>e₁</td>
<td>{5}</td>
</tr>
<tr>
<td>e₂</td>
<td>{1, 2, 3, 4}</td>
</tr>
</tbody>
</table>

Definition 2.10 (Transposed Table $T^t$) Given a table $T = (R, I)$, the transposed table $T^t$ of $T$ consists of a set of tuples. Each tuple corresponds to an item $o_k \in I$ and a rowset. If tuple $o_k$ contains $r_j$ in $T^t$, it means item $o_k$ is included in row $r_j$ in table $T$.

Example 2.9 (Transposed Table $T^t$) Table 2.3 represents the transposed version of Table 2.2, denoted $T^t$. As an example, $\{2, 4\}$ is the set of rows (rowset) that contain item $a_2$ in $T_m$ which is the second tuple in $T^t$.

2.3.2 CFI Mining on Transposed Data

The transposition method enables reduction of the complexity of the search on datasets that contain relatively few rows and relatively many columns. As the smaller dimension concerns the number of rows, the closed frequent itemsets can be discovered by searching for a large closed rowset from transposed table $T^t$. 
CHAPTER 2. ITEMSET MINING PRELIMINARIES

**Definition 2.11 (Closed Rowset)** Given the transposed table $T'$, a rowset $\beta$ is a closed rowset if not a proper superset $\beta'$ exists ($\beta \subsetneq \beta'$), such that the support of $\beta$ is the same as the support of $\beta'$.

**Example 2.10 (Closed Rowset)** From Table 2.3, for a rowset $\{1, 2\}$, the common itemset that occur in $\{1, 2\}$ is $b_1d_2e_2$. However, $b_1d_2e_2$ occurs also in rowset $\{4\}$. Therefore $b_1d_2e_2 = \{1, 2, 4\}$. Based on Definition 2.11, $\{1, 2\}$ is not a closed rowset as $\{1, 2\} \subset \{1, 2, 4\}$. Hence, $\{1, 2, 4\}$ is a closed rowset.

**Definition 2.12 (Closure)** Given a list of items, $I = \{o_1, o_2, \ldots, o_n\}$, an itemset $\alpha \subseteq I$ and a rowset $\beta \subseteq R$, it is defined that:

$$R(\alpha) = \{r_k \in R \mid \forall o_j \in I : r_k \in o_j\}, \quad 2.4$$

$$I(\beta) = \{o_j \in I \mid \forall r_k \in \beta : o_j \in \tau(r_k)\}. \quad 2.5$$

By this definition, $C(\alpha)$ can be defined as the closure of itemset $\alpha$ and $C(\beta)$ as the closure of rowset $\beta$, as follows:

$$C(\alpha) = I(R(\alpha)) \quad 2.6$$
$$C(\beta) = R(I(\beta)) \quad 2.7$$

Hence, an itemset $\alpha$ is a closed itemset if $\alpha = C(\alpha)$ and $\beta$ is a closed rowset if $\beta = C(\beta)$.

**2.4 Summary**

This chapter has provided an overview and definitions of association rule mining, beginning from frequent itemset mining in transactional datasets and then considering mining closed frequent itemsets in high-dimensional data. This will provide a basic understanding on the types of itemsets that can be mined, especially closed itemsets, in relation to the forthcoming chapters.
CHAPTER 2. ITEMSET MINING PRELIMINARIES

Mining closed frequent itemsets has been shown to be the best alternative when compared to mining only frequent itemsets or maximal frequent itemsets. The ability to provide a complete and reduced set of answers shows that closed itemset mining is computationally less costly in the association rule discovery.

The next chapter will consider the search strategies that have been proposed as a means of discovering closed frequent itemsets, in particular for high-dimensional datasets. This includes closedness-checking methods proposed by the algorithms to identify the closed itemsets. The advantages and disadvantages of these approaches are analyzed and discussed.
Chapter: 3

Strategies for Closed Frequent Itemset Mining

This chapter provides a review of the literatures on various search strategies that have been proposed in order to discover closed frequent itemsets. By examining these search strategies, an understanding of their advantages and disadvantages will be provided, and the gaps in current methods will be identified.

The chapter begins with Section 3.1, which describes the historical development of search strategies for mining closed frequent itemsets and their drawbacks.

Section 3.2 presents the current approach to discovering closed frequent itemsets from high-dimensional data.

Section 3.3 gives an example of an algorithm that proposes the search for large cardinality closed itemsets in high-dimensional datasets.

Section 3.4 considers an algorithm that searches closed itemsets using more than one constraint.

Finally, Section 3.5 summarizes the chapter, and concludes by pointing out the gaps in the literature that will be addressed in this thesis.
3.1 Column Enumeration-based Strategy

3.1.1 Apriori-based Bottom-up Search

The enumeration-based strategy delivers numerical information in the form of counts for individual items or events retrieved (Brown, 1995). A column set enumeration-based strategy explores a dataset according to its item value. The earliest frequent itemset mining algorithm that employed this strategy was the Apriori (Agrawal et al., 1993). The Apriori algorithm enumerates the frequent itemsets in ascending order of size. This enumeration approach is termed a bottom-up search. Figure 3.1 illustrates an example of the bottom-up column (item) enumeration tree, showing all the item combinations of the dataset in Table 2.1 from Chapter 2.

Figure 3.1: Bottom-up column enumeration tree
Apriori traverses the column enumeration tree using a bottom-up search in *breadth-first* order. This means that each level of the tree must be fully explored to discover frequent itemsets before moving onto the next level. The algorithm implies that frequent itemsets are mined through an iterative level-wise approach, based on candidate generation. *Candidate itemsets* refer to the itemsets generated whose supports are counted during the process of discovering frequent itemsets. Therefore, to identify entire frequent itemsets, all possible candidate itemsets must be tested.

However, as in reality the number of existing candidate itemsets can be huge, and therefore, identifying all candidates for these itemsets is both challenging and time consuming, and runs into the problem of achieving scalability. To reduce the search space for candidates, Apriori applies the *anti-monotonic* or *downward closure* property, which defines an itemset as frequent if and only if all of its sub-itemsets are frequent (Agrawal et al., 1993). This means that all of the supersets of an infrequent itemset found do not have to be considered.

**Definition 3.1 (Anti-monotonic)** Given a dataset, \( T \), with items \( I \), let \( \alpha_1 \) and \( \alpha_2 \) be two itemsets such that \( \alpha_1, \alpha_2 \subseteq I \), then:

\[
\alpha_1 \subseteq \alpha_2 \Rightarrow |T_{\alpha_1}| \geq |T_{\alpha_2}|
\]

The *Apriori* algorithm, as presented in Algorithm 3.1, begins by first scanning the dataset to find the frequent 1-itemsets. It then uses the frequent 1-itemsets to generate candidate frequent 2-itemsets, and checks these against the dataset to obtain the frequent 2-itemsets, and so on. The algorithm iterates until no more frequent \( k \)-itemsets can be generated for some \( k \).

*Apriori* is a level-by-level candidate-generation-and-test algorithm where, to discover frequent itemset of size \( n \), the algorithm has to scan the dataset \( n \) times and requires the checking of \( 2^n - 1 \) candidate itemsets. Several frequent itemset mining algorithms have been proposed that extend *Apriori* in various ways, which includes
implementing the hashing technique to reduce the number of candidate itemsets (Park et al., 1995). There are methods that attempt to reduce the number of dataset searches by dividing the dataset into non-overlapping partitions (Savasere et al., 1995) and dynamically counting candidate itemsets of varying length (Brin et al., 1997). Another method, proposed by Bastide et al. (2000) performs pattern counting inference based on the concept of key patterns. A key pattern is the smallest itemset that represents a group of itemsets with equivalent support. This leads to a reduction in the number of patterns counted, as well as a reduction in dataset scans. As the focus of this research is on discovering closed itemsets, further details of these algorithms will not be discussed.

An example of a well-known algorithm that discovers closed frequent itemsets using an Apriori-based search is A-CLOSE.

Algorithm 3.1: Apriori algorithm

**Input:** D = Dataset, minsup = minimum support  
**Output:** \( F_1, F_2, \ldots, F_k \), a set of frequent itemsets

\[
F_1 = \{ \text{Frequent 1-itemsets} \}; \\
\text{for } (k=2, F_{k-1} \neq 0, k++) \text{ do begin} \\
\quad C_k = \text{New candidate generation from } F_{k-1} \\
\quad \text{forall transactions } t \in D \text{ do begin} \\
\quad \quad C_t = \text{Candidate contained in } t \\
\quad \quad \text{forall candidates } c \in C_t \text{ do} \\
\quad \quad \quad c.\text{count}++; \\
\quad \text{end} \\
\quad F_k = \{ c \in C_k \mid c.\text{count} \geq \text{minsup} \} \\
\text{end}
\]
CHAPTER 3. STRATEGIES FOR CLOSED FREQUENT ITEMSET MINING

Algorithm 3.2: A-CLOSE algorithm

**Input:** Dataset, \( \text{minsup} \) = minimum support  
**Output:** \( CFI \), a set of closed frequent itemsets

\[
G_1 = \{1\text{-itemsets generators}\};  
\text{support} = \text{count}(G_1)  
\]  
\[
\text{for all generators } p \in G_1 \text{ do begin}  
\quad \text{if (support}(p) < \text{minsup}) \text{ then delete } p \text{ from } G_1;  
\text{end}  
\]  
\[
\text{level} = 0;  
\text{for } (i = 1; G_i\text{.generator} \neq \emptyset; i++) \text{ do begin}  
\quad G_{i+1} = \text{Generate } (i+1)\text{-generators of } G_i  
\quad \text{if (level} = 0) \text{ then level} = i;  
\quad \text{// Iteration number of the first prune}  
\text{end}  
\]  
\[
\text{if (level} > 2) \text{ then begin}  
\quad G = \bigcup\{G_j | j < \text{level} - 1\};  
\quad \text{// Those generators are all closed}  
\quad \text{for all generators } p \in G \text{ do begin}  
\quad \quad p\text{.closure} = p\text{.generator;}  
\quad \text{end}  
\text{end}  
\]  
\[
\text{if (level} \neq 0) \text{ then begin}  
\quad G' = \bigcup\{G_j | j \geq \text{level} - 1\};  
\quad \text{// Some generators that are not closed}  
\quad G' = \text{Closure of generator } G'  
\text{end}  
\]  
\[
\text{CFI} = \{c\text{.closure, c.support} | c \in G \cup G' \};  
\]

A-CLOSE (Pasquier et al., 1999) (see Algorithm 3.2), was the first algorithm to discover closed frequent itemsets using an Apriori-based framework. An example of the application of the algorithm is illustrated in Figure 3.2, based on the dataset from Table 2.1. The algorithm constructs a set of generators to identify closed
CHAPTER 3. STRATEGIES FOR CLOSED FREQUENT ITEMSET MINING

frequent itemsets. These generators are the smallest itemsets that can determine the closed itemsets, based on the properties of the closed itemset lattice\(^2\).

During the search process, generators that have the same support as one of their subsets and therefore have the same closure as the subset are pruned. At the end of the search, the closure of all the generators identified is obtained by

\(^2\) Properties of the closed itemset lattice have been outlined in Chapter 2.
intersecting all the transactions that contain the generator as a subset. Duplicate closures are then removed.

Apriori-based algorithms have shown good performance when applied to sparse datasets where the frequent itemsets or closed frequent itemsets are relatively short. However, with dense datasets, these algorithms have been shown to scale poorly and are impractical, because of high-computational costs (Brin et al., 1997; Pasquier et al., 1999; Bastide et al., 2000). This drawback is because of: (i) generation of a huge number of candidate itemsets (or generators in the case of the A-CLOSE algorithm) and (ii) repeated scanning of the dataset and checking the candidates by pattern matching.

3.1.2 Pattern Growth without Candidate Generation

To overcome the limitations of the Apriori-based approach, Han et al. (2000) proposed the discovery of frequent itemsets without candidate generation through an algorithm called FP-growth (Frequent Pattern-growth). The main idea of this algorithm relies on a compact tree data structure called the FP-tree, which stores only information related to the mining of frequent itemsets – i.e. the number of itemsets and its frequency of occurrence.

An example of the FP-tree that is constructed from Table 2.1 for minsup = 3 is given in Figure 3.3. The dataset is initially scanned to derive a list of frequent items, which are then ordered in frequency descending order, e.g. \( \langle b: 6, e: 5, a: 4, c: 4, d: 4 \rangle \). These items are stored in the header table. The dataset is then scanned for the second time to build the FP-tree. Only frequent 1-itemsets are stored as nodes in order to ensure the compactness of the tree. The nodes are arranged in frequency descending order, so that frequently occurring nodes will have better chances in prefix sharing than otherwise. If two transactions share a common prefix node, these nodes are merged as one prefix structure, and the count of each node associated with the prefix is incremented. This helps to prevent repeated scanning of the dataset.
The FP-growth algorithm is given in Algorithm 3.3. The method searches for the frequent itemsets by recursively partitioning the FP-tree into non-overlapping subsets based on the item list. Following the frequency ascending order

Algorithm 3.3: FP-growth algorithm

**Input:** Tree = FP-tree constructed from dataset D, minsup = minimum support  
**Output:** F, a set of frequent itemsets

if Tree contains a single path P;  
then for each combination (denoted as α) of the nodes in the path P do  
generate itemset α ∪ F with support = minsup of nodes in α;  
else for each item i_k in the header Tree do  
generate itemset α = i_k ∪ F with support = i_k.support;  
construct α’s conditional pattern base and then α’s conditional FP-tree Tree_α;  
if Tree_α ≠ Ø  
then call FP-growth (Tree_α, α)  
}
of the item list, the algorithm begins with each frequent length-1 itemset as the *suffix item*. It then traverses the *FP-tree* by following the link of each frequent item that co-occurs with the suffix item. The collection of all frequent itemsets co-occurring with the suffix item forms the *conditional pattern base*. The *FP-tree* constructed from the conditional pattern base is called the *conditional FP-tree*.

*FP-growth* is achieved by concatenating the suffix item with the frequent itemsets generated from the conditional *FP-tree*. The suffix item of length-1 itemset will then be used to continuously generate those with a length equal to 2, and so on, until the conditional *FP-tree* contains only one single path from which frequent itemsets can be directly generated. Table 3.1 shows the conditional pattern base and conditional *FP-tree* of Figure 3.3 for every suffix item.

As an example, in Figure 3.3, \((d: 4)\) is the suffix item with the smallest number of support after item ordering. There are 3 branches that co-occur with item \(d\). These branches are the conditional sub-database related to suffix item \(d\). The conditional *FP-tree* for item \(d\) consists of \{\((b: 4), (e: 4), (a: 3), (b, e: 3), (b, a: 3), (a, e: 3), (b, e, a: 3)\)\}, and all the combinations of frequent itemsets that consists of \(d\) as its component are \{(d : 4), (bd : 4), (ed : 3), (ad : 3), (bed : 3), (bad : 3), (aed : 3),

<table>
<thead>
<tr>
<th>item</th>
<th>conditional pattern base</th>
<th>conditional FP-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>{(b, e, a, c : 1), (b, e, a : 2), (b, c : 1)}</td>
<td>{(b : 4), (e : 3), (a : 3), (b, e : 3), (b, a : 3), (a, e : 3), (b, e, a : 3)}</td>
</tr>
<tr>
<td>c</td>
<td>{(b, e, a : 2), (b, e : 1), (b : 1)}</td>
<td>{(b : 4), (e : 3), (b, e : 3)}</td>
</tr>
<tr>
<td>a</td>
<td>{(b, e : 4)}</td>
<td>{(b, e : 4)}</td>
</tr>
<tr>
<td>e</td>
<td>{(b : 5)}</td>
<td>{(b : 5)}</td>
</tr>
<tr>
<td>b</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Several closed frequent itemset mining algorithms have been devised as extensions to the FP-growth method that maintain the discovered itemsets using the FP-tree structure or in a pattern-tree similar to an FP-tree. These include: (i) CLOSET, (ii) CLOSET+, (iii) FP-Close, (iv) AFOPT.

(a) CLOSET (Pei et al., 2000) identifies the closed frequent itemsets by performing a depth-first search on the FP-tree. The algorithm applies the single prefix path compression technique which searches for a single prefix path in the FP-tree in which closed itemsets can be directly extracted from the conditional pattern base. As an example, for a conditional database in Table 3.1, its corresponding FP-tree has only one branch: \((b:6),(e:5)\), hence the closed frequent itemsets, \((abe:4)\) can be directly enumerated. The discovery of closed frequent itemsets is then continued by recursively building the conditional FP-tree and identifying the superset of the remaining itemsets that appear in every transaction before checking the subset of the particular itemset. CLOSET has shown to be much faster than Apriori-based A-CLOSE algorithm on dense dataset when the minimum support threshold is low (Pei et al., 2000).

(b) CLOSET+ (Wang et al., 2003) introduces a hybrid tree-projection method which builds the conditional pattern base depending on whether the dataset is sparse or dense. The algorithm also proposed two subset-checking techniques to determine if a discovered itemset is a subset of an already found closed itemset candidate with the same support. For dense datasets, a two-level hash-indexed result tree is applied, where each level uses the ID of the last item and the support of the current itemset as the hash key. Each closed itemset discovered is inserted into the result tree and the length of its path is recorded. In contrast to the FP-tree structure, the support of a node is replaced by the maximum value among the support of closed itemsets sharing the common prefix. For sparse datasets, the subset-checking is applied on the global prefix-tree because
the result tree is not very space-efficient. All the nodes of the tree and their corresponding prefix path can be traced by following the side-link pointer recorded in its header table. Therefore, a closed itemset is obtained using the upward subset-checking to see whether it appears in each prefix path with respect to the prefix itemset. \textit{CLOSET+} shows that it is an order of magnitude faster and consumes less memory than \textit{CLOSET} at lower support thresholds. \textit{CLOSET+} is also more scalable than \textit{CLOSET} as the number of rows increases (Wang et al., 2003). \textit{CLOSET+} has the advantage of applying different methodologies prior to the characteristics of the dataset whether it is sparse or dense.

(c) Grahne and Zhu (2003) introduced the algorithm \textit{FP-Close} to discover closed itemsets by constructing a \textit{CFI-tree} (Closed Frequent Itemset-tree). Like \textit{CLOSET+}, during the insertion of a closed frequent itemset, the support count of the nodes in the \textit{CFI}-tree is replaced by the current maximal support count for the related itemset. \textit{FP-Close} also performs a similar subset-checking technique to \textit{CLOSET+} with the difference that the algorithm also considers the support count of the itemset. The support count of each item in the list must be equal to, or greater than, the support count of the itemset, before ensuring that it is not a subset of another itemset with the same support value. \textit{FP-Close} shows similar performance at lower support thresholds as compared to the algorithms selected in Grahne and Zhu (2003). The reason for this is that \textit{FP-Close} generates more non-closed frequent itemsets hence increases the amount of time needed to check for closed itemsets. However, \textit{FP-Close} requires less amount of memory due to the compactness of the constructed \textit{CFI-tree}.

(d) Liu et al. (2003) introduced the algorithm \textit{AFOPT}, which stores closed frequent itemsets in a tree structure called a \textit{Condensed Frequent Pattern tree} or \textit{CFP-tree}. The algorithm traverses the tree in both a top-down and
bottom-up manner. Each node of the CFP-tree is a variable-length array, in which items in the same node are sorted in frequency ascending order. The CFP-tree has two properties: (1) the left containment property that ensures the items of an itemset can only appear in the subtrees pointed to by the itemset or from previous itemsets in the tree; (2) the Apriori property ensures that the support of any child nodes of a CFP-tree for a particular itemset cannot be greater than the support of that itemset. AFOPT performs superset-checking of the CFP-tree based on these two properties. Superset-checking ensures that an itemset is a closed itemset if all of its supersets have a lower support threshold. The algorithm also performs subset-checking by applying a two-layer hash map similar to CLOSET+ to check whether the itemset is closed before searching the CFP-tree. The hash map contains the item and the maximal length of the itemset mapped to it. A closed itemset is discovered if any of the items it mapped to contain a lower value than its length. AFOPT shows that the algorithm scales well as the average transaction length increases as compared to the algorithms selected in Grahne and Zhu (2003). The algorithm demonstrates better performance in terms of running time on dense datasets due to its adaptive nature and the efficiency of the subset checking technique. AFOPT is also memory efficient due to the construction of the compact CFI-tree.

The high compression ratio of the FP-tree has contributed to the reduction of the search space for discovering closed frequent itemsets, especially in dense datasets. In addition to the breadth-first order applied in the Apriori-based approach, the search for frequent itemsets from the FP-tree can be made in a depth-first manner. This means that the supports of all descendant itemsets of a node are determined before determining the frequent extensions of other nodes in the column enumeration tree. Thus, the depth-first search strategy quickly tends to find the
longer itemsets first in the search process and the branches of a node is searched only if the itemset is frequent. Hence, the strategy is able to reduce the processing time by cutting down the search space that contains itemsets which do not satisfy the desired threshold. However, $FP$-tree based algorithms are unable to give good compression for long itemsets. Building the $FP$-tree will require a larger amount of time and memory space, especially for datasets with large number of columns (items).

### 3.1.3 Exploring the Vertical Data Format

Datasets for mining frequent itemsets are generally represented in a horizontal format, with each row corresponding to a list of items (i.e., $\{\text{rid: itemset}\}$), where $\text{rid}$ is the row-id and $\text{itemset}$ is the set of items in row $\text{rid}$. A study by Zaki (2000a) proposed an alternatively representation of the dataset which shows that the row information can also be recorded in a vertical data format. The vertical data format is an inverted representation of the original dataset. It is generated by scanning the dataset, and builds the rowset of each single item where the identities of the rows containing the item are listed (i.e., $\{\text{item: rowset}\}$). Table 3.2 shows an example of the vertical representation of the transactional dataset from Table 2.1. The advantage of applying the vertical data layout is that there is no need to scan the dataset to find

<table>
<thead>
<tr>
<th>Items</th>
<th>Tidset</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1 3 4 5</td>
</tr>
<tr>
<td>b</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>c</td>
<td>2 4 5 6</td>
</tr>
<tr>
<td>d</td>
<td>1 3 5 6</td>
</tr>
<tr>
<td>e</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>
the support of \((k + 1)\)-itemsets (for \(k \geq 1\)). This is because the rowset of each \(k\)-itemset carries the complete information required for counting such support. Hence, computing the supports is simpler and faster.

An example of the algorithm that utilizes the vertical data format to discover closed frequent itemsets is CHARM (Zaki and Hsiao, 2005). CHARM simultaneously explores both the itemset and rowset space in a depth-first manner on a dual itemset-tidset search tree (\(IT\)-tree). In addition to the itemset value, each node in the \(IT\)-tree also represents its rowset value. The rowset of the corresponding \((k + 1)\)-itemsets is obtained by intersecting the rowset of the frequent \(k\)-itemsets. The process repeats, until no frequent or candidate itemsets can be found.

The closed itemsets are identified using the \(IT\)-pair (Itemset-Tidset pair) properties proposed in the study made by Zaki and Hsiao (2005). In addition, a hash function is applied to the rowset value by performing the sums of rids in the rowset to quickly identify the closed itemsets.

The drawbacks of the vertical data format are that it consumes a lot memory to store large cardinality rowsets, and increasing number of rowset intersections. CHARM attempts to reduce the size of the intermediate rowset through diffset which keeps track of the differences in rids of the candidate itemsets from its parent frequent itemsets. This has led to an increase in the algorithm’s performance due to the less number of rowset intersection.

CHARM has shown that it performs better than pattern-growth based closed itemset mining algorithms, such as A-CLOSE and CLOSET. It is several orders of magnitude faster than A-CLOSE and CLOSET at low support thresholds. CHARM also scales well, having linear increase in running time with increasing number of transactions. One of the advantages of CHARM is that the diffset format is resilient to sparsity. However, if the dataset contains many short itemsets, the tidset/diffsets operation in CHARM can be expensive. Also, CHARM is a column-enumeration based algorithm that performs the search using the Apriori-based approach, which is known to generate large number of candidate itemsets.
3.2 Row Enumeration-based Strategy

In contrast to the transactional data, high-dimensional datasets may contain 10 K – 100 K columns or items but usually have only 100 to 1000 rows or transactions (typically a difference of a few orders of magnitude). Column enumeration-based mining algorithms described above typically begin the search for closed frequent itemsets with small itemsets that appear frequently and use these intermediate results to build larger and larger itemsets. This strategy is generally effective for datasets with the characteristic of having a relatively large number of rows and a relatively smaller number of columns, hence the term column-enumeration search.

Datasets with relatively many more columns than rows present efficiency challenges for algorithms that search based on the column values. This is because the number of possible column combinations is extremely high, and hence, correspondingly increases the search space size. For this reason, a high-dimensional dataset is considered to be dense. Therefore, enumerating the closed frequent itemsets by considering the row-space (e.g. experiments) rather than the column-space (e.g. items) should be more effective.

3.2.1 Bottom-up Search

CARPENTER was the first algorithm to adopt the approach of mining the closed frequent itemsets in high-dimensional datasets using the row enumeration space in a bottom-up manner (Pan et al., 2003). The bottom-up search strategy of the row enumeration space implies that the dataset is searched starting from the smallest rowset value, and builds larger rowset values during the process. Figure 3.4 illustrates an example of the row enumeration tree that lists all the rowset values in bottom-up order. Unlike the column enumeration-based search tree, the nodes of the row enumeration tree are now viewed as a set of row values (rowset), as opposed to a set of column (itemset) value.

The algorithm integrates the advantage of vertical data format (Zaki, 2000a) by transposing the dataset (Rioult et al., 2003) so that the rowset is viewed as a set
of rows. In contrast to the column enumeration-based method which performs intersection on the rowset values of the transposed table, row enumeration-based is driven by intersecting the itemsets in order to discover the closed frequent itemsets.

CARPENTER applies a depth-first search of the row enumeration tree. The algorithm recursively constructs conditional transposed tables where each computed conditional transposed table represents a node in the row enumeration tree. Each conditional transposed table contains items that exist in the conditional rowset, along with rids of the items that are larger than any of the conditional rowset.

The advantage of the conditional transposed table is that once a closed itemset is discovered for that particular rowset, further checking on the node for the rowset value is unnecessary. An example of a conditional transposed table, $T^r$,
where $x = \{2, 3, 4\}$, from Table 3.2 is as shown in Table 3.3. As rid 5 occurs in each
taxe of $T'_{[2,3,4]}$, $b_1c_2 = \{2, 3, 4, 5\}$ is a closed itemset.

Study shows that CARPENTER performs better with respect to run time than
CHARM and CLOSET, as the minimum support threshold varies (Pan et al., 2003).
This is due to the fact that the increase in the column enumeration space leads to the
decrease in the performance of the column-encryption based algorithm such as
CHARM and CLOSET. CARPENTER also is 100 times faster than CHARM and
1000 times faster than CLOSET as the number of length ratio of the dataset
increases.

There are several algorithms that have their basis in CARPENTER ideas, and
these include: (i) FARMER, (ii) TopKGRS, (iii) COBBLER.

(a) FARMER (Cong et al., 2004a) was particularly designed to find all
association-based classification rules by row enumeration. The algorithm
searches the row enumeration tree in depth-first order to build classifiers of
the form $X \rightarrow C$, where $C$ is a class label and $X$ is a set of attributes. Hence, it
requires a duplicate of the dataset in order to classify the classes’ information
prior to mining. The method is supported through the transposed tables,
taking into account class information. Thus, each itemset in the transposed
table is enumerated according to a positive and negative class. In this
particular algorithm, the association rules discovered are required to satisfy
more than one constraint such as support, confidence and chi-square (Cong
et al., 2004a). FARMER shows that it is 2 to 3 orders of magnitude faster
than ColumnE (Bayardo and Agrawal, 1999) and CHARM as the minimum
support threshold decreases (Cong et al., 2004a). This is because FARMER
depends on the number of row combination of the high-dimensional dataset as compared to the number of column combinations made by ColumnE and CHARM.

(b) \textit{RERII} (Cong et al., 2004b) extracts all closed frequent itemsets by searching the row-enumeration space depth first. The algorithm begins by removing all infrequent 1-itemsets from the dataset. Each row value of the sibling nodes in the enumeration tree is then intersected with one another, iteratively generating sub-itemsets of greater support. If the sub-itemset is equal to the parent itemset, the support of the parent itemset increases. This continues recursively until no smaller itemsets can be formed or the branch of the sub-itemset is equivalent to the parent itemset. All the closed frequent itemsets that do not satisfy the \textit{minsup} threshold are pruned. \textit{RERII} has been shown to be faster in terms of runtime than the column enumeration-based algorithms \textit{CLOSET+} and \textit{CHARM} at low support thresholds for similar reasons, as with \textit{FARMER}. The algorithm also performs 2-4 times faster than \textit{CARPENTER} on the test datasets since it does not require building the conditional transposed tables (Cong et al., 2004b).

(c) Similar to \textit{FARMER}, \textit{TopKGRS} was designed to discover a set of rule groups (Cong et al., 2005). The algorithm uses a preference selection to specify the number of top covering rule groups (top-k), in order to reduce the number of rule set. By implementing the top-k, \textit{TopKGRS} has shown to be 2 to 3 orders of magnitude faster than \textit{FARMER} especially at low \textit{minsup} threshold. This is because \textit{FARMER} discovers a large number of rule groups at low \textit{minsup} as compared to the restricted number of rule groups obtained by \textit{TopKGRS}. The runtime of \textit{TopKGRS} monotonously increase with increasing value of \textit{k}. The improvement in the runtime is also due to the implementation of the compact prefix-tree in the algorithm (Cong et al., 2005).

(d) \textit{COBBLER} (Pan et al., 2004) employs dynamic evaluation of closed frequent itemsets by combining both the bottom-up row-enumeration and bottom-up
column enumeration approach, depending on the dataset characteristics. Similar to CARPENTER, COBBLER performs a depth-first traversal search of both trees, by recursively constructing several conditional tables and conditional transposed tables. Each conditional table represents a column enumerated node, while each conditional transposed table represents a row enumerated node. The change from row enumeration to feature enumeration or vice versa is decided through evaluating a switching condition. The objective is to estimate the enumeration costs for the sub-trees and selecting the smallest one from both sub-trees, i.e. column or row based. The enumeration cost is estimated from two components of the tree, its size and the computation cost at each node of the tree. The size of the tree is based on the estimated number of nodes it contains, while the computation cost at a node is measured using the estimated number of rows (features) that will be processed at the node. The advantage of this strategy is that each portion of the dataset can be processed using the most suitable method, hence making the mining more efficient. Experiments show that at higher minsup thresholds, when the dataset needs to consider large numbers of rows as compared to the number of columns, column enumeration-based algorithms, CLOSET+ and CHARM performs better in terms of runtime in most of the cases. If the minsup threshold is reduced, COBBLER performs better in terms of run-time. The effectiveness of COBBLER has been demonstrated in experiments on a dataset, with both a relatively large number of rows and columns (Pan et al., 2004).

(e) MAXCONF (McIntosh and Chawla, 2007) applied the row enumeration-based bottom-up search to discover closed itemsets using the confidence measures. This algorithm arose from the observation that implementing the support threshold has lead to the pruning of many interesting unknown itemsets that could provide high confidence rules. The algorithm proposed two confidence pruning methods, which results in MAXCONF to scale well
with the changes in the confidence threshold. It also shows that it could
discover interesting rule groups with high confidence as compared to the
support-based algorithm, RERII (McIntosh and Chawla, 2007).

As stated in the previous chapters, mining closed frequent itemsets based on
the support constraint as the search threshold means discovering closed itemsets that
contain large rowsets. Therefore, the main restriction of the bottom-up approach is
the size of the rowset. As it is monotonic in terms of the bottom-up search order, it is
hard to prune the row enumeration search space early. For example, suppose the
\textit{minsup} is set to 3, although all the nodes in the first two levels from the root
obviously cannot satisfy this constraint, these nodes still need to be checked (Pan et
al., 2003; Cong et al., 2004a; Cong et al., 2004b). Hence, as \textit{minsup} increases, the
time needed to complete the mining process does not decrease correspondingly or
rapidly. This limits the application of this kind of algorithm in real situations. In
addition, the inability to prune the search space earlier adds to the increase in
memory cost. For example, the \textit{CARPENTER} algorithm needs to save many \textit{x-}
\textit{conditional} transposed tables in memory during the mining process. For a table with
\textit{n} rows, the maximum number of different levels of transposed tables in memory is
\textit{n}, although among these, the first (\textit{minsup}-1) levels will not contribute to the final
result.

\textbf{3.2.2 Top-down Search}

To take advantage of the support constraint, Liu et al. (2006) proposed that the row
enumeration-tree is traversed in a top-down manner. A top-down search implies that
along each path of the tree, the rowsets are check from large to small ones. Given a
\textit{minsup} threshold, for a dataset with \textit{n}-rows, the search for closed frequent itemsets
ends for levels greater than (\textit{n}-\textit{minsup}) in the row enumeration tree.

Figure 3.5 shows an example of the top-down row enumeration for 5 rows
from Table 2.3. Each node of the tree represents a rowset value. The level of root
node is defined as 0, and the highest level for a dataset with \textit{n} rows is (\textit{n}-1). Suppose
the \( \minsup = 2 \), a further search is stopped at level 3, because the rowsets represented by nodes at level 3 and 4 will not contribute to the set of closed frequent itemsets.

Two examples of the algorithms that apply the top-down approach are: (i) \textit{TD-Close} and (ii) \textit{TTD-Close}.

(a) \textit{TD-Close} (Liu et al., 2006) was the first algorithm to implement the top-down approach of the row enumeration tree. Similar to \textit{CARPENTER}, \textit{TD-Close} employs an array-based data structure by performing sub-division of the itemsets using conditional transposed tables. Each sub-table corresponds to a node in the row enumeration tree. The conditional transposed table is called an \textit{x-excluded transposed table}, where \( x \) is a rowset, excluded from the table. The algorithm finds all the items from the transposed table that contain the rowset id, \( rids \), greater than the specified
rowset value, $x$, but only items which contain rids less than $x$ are retained in the $x$-excluded transposed table. During this process, items that do not satisfy the minsup constraint are discarded. Based on Definition 2.11, a rowset is closed if a larger rowset value containing the same itemset does not exist. To satisfy this condition, TD-Close performs a trace-based closedness-checking method by keeping track of the rids excluded during the intersection of the itemset. Hence, to facilitate the search for closed rowsets, a column value is added to the $x$-excluded transposed table called skip-rowset. The itemsets that occur in the same rowset are merged and at the same time, the intersection of the skip-rowset values is performed. The itemset which produces an empty skip-rowset during the merge is a closed itemset. Experimental results demonstrate that TD-Close is faster than FP-Close and CARPENTER in terms of runtime as the minsup threshold decreases. The ability of the top-down search of the row enumeration tree to prune the search space that does not satisfy the minsup threshold earlier adds significantly to the running time of the algorithm. Because of this, TD-Close consumes less memory as compared to a bottom-up approach in CARPENTER. On the other hand, FP-Close is a column enumeration-based algorithm which requires an explosive number of frequent itemsets that need to be checked.

(b) The TTD-Close (Liu et al., 2009) algorithm, a development of TD-Close, represents the dataset using a tree data structure, as opposed to the flat table in TD-Close to perform the search for closed frequent itemsets. The main advantage of using this approach is that the tree structure provides a more compact representation of the dataset. The tree structure, termed FR-tree (Frequent Rowset-tree), is similar to the representation of the FP-tree (Han et al., 2000). The differences between the trees are that instead of representing the nodes with the item value (FP-tree), each node of the FR-tree is represented with the rid value and the nodes in the FR-tree are
linked through a parent pointer instead of a child pointer (FP-tree). An additional structure, termed *IP-List* (Itemset Pointer List), is added to the *FR-tree*, and contains information that assists in the discovery of closed frequent itemsets. An example of the *FR-Tree* and the *IP-List* for the transposed table $T'$ of Table 2.3 is shown in Figure 3.6. *IP-List* is similar to the $x$-excluded transposed table in *TD-Close*. The difference is that instead of using the excluded rowset $x$ during recursion, the algorithm uses the rowsets which do not contain $x$. The *IP-List* consists of four parts: (i) a set of pointers each of which represents an itemset and points to a node in the

![Figure 3.6: FR-Tree and IP-List](image-url)
CHAPTER 3. STRATEGIES FOR CLOSED FREQUENT ITEMSET MINING

FR-Tree; (ii) an explicit rowset which contains several rids that may represent the itemset; (iii) an implicit rowset which represents rids that exist for the particular itemset and (iv) the current minimum support threshold, \( c_{\text{Minsup}} \). The recursion path of \( TTD\)-Close follows the changes in the explicit rowset. Experiments conducted by Liu et al. (2009) demonstrate that \( TTD\)-Close provides the least runtime as compared to the algorithms, \( TD\)-Close, \( FP\)-Close and CARPENTER. \( TTD\)-Close also consumes the least amount memory during mining for closed itemsets. This is because \( TD\)-Close needs to build several \( x\)-excluded transposed tables, CARPENTER needs to deal with rowsets that are smaller than the \( \text{minsup} \) threshold and \( FP\)-Close needs to build more \( FP\)-trees.

However, due to the density of high-dimensional datasets, the row enumeration based strategy still encounters exponential space size with respect to the number of itemsets. As the frequency threshold gets smaller, the time required to find closed frequent itemsets dramatically increases (Liu et al., 2009). Even with the various search strategies proposed, these algorithms still encounter challenges mining relatively large itemsets. This is because the previous search processes require the generation of an explosive number of small frequent itemsets, hence taking much of the memory space to store large frequent ones.

3.3 Pattern-Fusion: Mining the Colossal Itemsets

Association mining tasks usually give greater importance to itemsets that are bigger in size, especially in areas such as bioinformatics. These long cardinality itemsets are termed colossal itemsets (Zhu et al., 2007).

The concept of colossal itemsets was first introduced by Zhu et al. (2007) in an algorithm based on pattern-fusion for finding a measurably good approximation to the enumeration of all colossal closed itemsets in high-dimensional datasets. The algorithm traverses the tree according to the column (item) enumeration. However,
CHAPTER 3. STRATEGIES FOR CLOSED FREQUENT ITEMSET MINING

instead of traversing each node of the tree, it randomly discovers large cardinality itemsets by merging the small cardinality candidate frequent itemsets selected. These small cardinality candidate itemsets are known as core-patterns.

Definition 3.2 (Core Pattern) For a pattern \(\alpha\), an itemset \(\beta \subseteq \alpha\) is said to be \(\tau\)-core pattern of \(\alpha\) if \(\frac{|\mathcal{T}_\alpha|}{|\mathcal{T}_\beta|} \geq \tau\), \(0 < \tau \leq 1\). \(\tau\) is called the core ratio and \(T\) is the transaction dataset.

Pattern-Fusion begins by generating a desired set of small frequent itemsets. Based on the user-specified maximum number of itemsets to be mined, random selections of core-patterns are made from the generated small frequent itemsets. All the itemsets that satisfy the core ratio for each core patterns are then combined to produce larger cardinality itemsets.

The concept of a core pattern was proposed in order to provide the ability for the algorithm to skip a large number of frequent itemsets whenever possible. This is because the growth of each itemset is not performed by adding one item each time, but by an agglomeration of selected multiple itemsets. Hence, Pattern-Fusion is able to traverse down the search tree much more rapidly toward the colossal itemsets. Zhu et al. (2007) also stated that colossal itemsets exhibit robustness, in the sense that if a small number of items are removed from the itemset, the resulting itemset will have a similar support set. This is based on the relationship between the support set of a colossal itemset and those of its sub-itemsets: the larger the itemset size, the more prominent the robustness observed.

Definition 3.3 \(((d, \tau)\text{-Robustness})\) A pattern \(\alpha\) is \((d, \tau)\)-robust if \(d\) is the maximum number of items that can be removed from \(\alpha\) for the resulting pattern to remain a \(\tau\)-core pattern of \(\alpha\).
As the number of colossal itemsets discovered is an approximation of a complete solution, Zhu et al. (2007) also proposed an evaluation model to assess the quality of mining results against the complete set. This model provides a way to measure the goodness of an approximate solution against a complete solution by measuring the distance between two arbitrary itemsets.

Several studies conducted on both synthetic and real datasets have demonstrated that Pattern-Fusion is able to provide a good approximation for discovering colossal itemsets in datasets. Unlike existing frequent itemset mining algorithms, Pattern-Fusion skips the need to examine a large number of mid-sized ones. Interestingly, their experimental results were presented with a minimum support threshold as the x-axis whereby the running time for Pattern-Fusion is 10 times faster at lower support threshold as compared to the selected algorithms.

### 3.4 D-Miner: Mining the Constraint-based Concept

Another approach in mining a high-dimensional dataset is to find a formal concept (FC) (Dong et al., 2005). Given a 0/1 matrix, a formal concept is a subset of $k$ rows and $l$ columns, such that all 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 $l$ 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. A closed itemset may be considered as column subsets and the collection of support as a row subset.

Besson et al. (2005) have applied the mining of (formal) concepts using constraints on high-dimensional datasets with the D-Miner algorithm. The high-dimensional dataset is initially transposed and is represented using a binary format (Table 3.4) where the values in the dataset are represented as 0 or 1. The algorithm begins with the largest cardinality itemset and the largest cardinality rowset that represents the dataset. This set of itemsets and rowsets are called a bi-set. D-Miner
performs a depth-first search of concepts by recursively splitting the initial bi-set into smaller bi-sets that do not contain “0” values. The division of the bi-sets is made using the elements that are found in the dataset which represents the “0” value. These elements are called cutters. The result of each divided bi-sets contains a concept of an itemset without a rowset value of “0” and a concept of a rowset without an itemset value of “0”.

Figure 3.7 shows an example of how D-Miner discovers the concepts using the binary representation of the dataset shown in Table 3.4. The search begins with the largest itemset (a b c d) and rowset (t1 t2 t3) value. The first cutter, (a, t2), which is represented in a box, is identified from the dataset as it is represented with a “0” value. The cutter is then used to divide the first bi-sets into further sub-bi-sets. The method repeats until there are no more cutters which leave the concepts representing a 1-rectangle. All the discovered concepts are shown in the last line of Figure 3.7. However, bi-set (cd, t1) (highlighted in bold) is not a concept as it is a subset of the bi-set (acd, t1). This is done through comparison with the existing concepts discovered.

As stated previously, entire concepts (or closed itemsets) that exist in the high-dimensional dataset are unlikely to be discovered. Therefore, to reduce the search space, D-Miner attempts to discover concepts that satisfy two constraints, based on the length of the rowsets, as well as the length of the itemsets discovered. The performance of D-Miner has been compared to CLOSET and CHARM, using the
support threshold. Unsurprisingly, *D-Miner* performs better than these two algorithms on high-dimensional datasets especially at lower minimum support threshold (Besson et al., 2005). The results of the study also show that *D-Miner* has gain significant decrease (as high as 97%) in the total closed itemsets discovered when applying more than one constraint. Hence, this shows that the algorithm is useful for the discovery of a particular group of closed itemsets satisfying specified constraints.

### 3.5 Summary

This chapter outlines the strategies to discover closed frequent itemsets, with examples of the algorithms as well as quantitative comparison and discussions of the drawbacks of the methods when applied to high-dimensional datasets.

From the foregoing, it can be concluded that:
A column enumeration-based search strategy is suitable for datasets that contain a relatively smaller number of columns or items. There are three basic methodologies described under this strategy: (i) Apriori-based generation that produces candidate itemsets in a level-wise manner. An example of the algorithm using this generation method is A-Close; (ii) the Pattern-growth method that mines the complete set of frequent itemsets without candidate generation. Examples of this type of algorithm include CLOSET, CLOSET+, FP-Close and AFOPT; and (iii) Vertical dataset representation in which each item in the dataset is represented with a set of row values. An example of the algorithm using the vertical data representation is CHARM.

All these methods adopt a bottom-up search of the column enumeration tree. For high-dimensional dataset, the characteristics of the dataset of having a relatively smaller number of rows and a relatively large number of columns means that the column-enumeration based methods require a considerable amount of resource to search the itemset space.

A row enumeration-based search strategy is suited to mining itemsets in high-dimensional datasets due to the fact that it searches the rowset space.

- The initial approach of the search was to traverse the row enumeration tree in a bottom-up manner. By using the support threshold, traversing the row enumeration tree bottom-up does not take advantage of the constraint. This is because the method has to go through all the nodes in the levels of the tree that do not satisfy the constraint and discard them. Examples of algorithms using the bottom-up row enumeration search are CARPENTER, FARMER, RERII, TopKGRS, COBBLER and MAXCONF.

- The top-down search strategy takes advantage of the support threshold by discovering itemsets beginning from the largest rowset value (the most frequent). However, by applying the support
constraint the strategy struggles to reach the large cardinality itemsets that exist at the lower end of the support threshold. Examples of algorithms within this category are TD-Close and TTD-Close.

- The Pattern-Fusion algorithm is an example of an algorithm that attempts to discover large cardinality (colossal) closed itemsets by approximating the number of colossal closed itemsets generated. However, approximating the number of colossal closed itemsets discovered might lead to missing some of the colossal closed itemsets that are of value. In addition, experimental results show that using the minimum support threshold, raises the question - “What is the maximum ‘colossal’ value of the itemsets that the algorithm can discover?”

- D-Miner applies more than one constraint to discover the closed itemsets in high-dimensional datasets. However, the objective of the algorithm was to discover only a group of closed itemsets that are of interest.

In the next chapter (Chapter 4), the limitations of the approaches as stated above, will be address through a new proposed algorithm.
Chapter: 4

*DisClose*: Mining Colossal Closed Itemsets

This chapter presents the proposed algorithm which has been developed to efficiently discover the colossal closed itemsets from high-dimensional datasets.

Section 4.1 provides an overview of the steps taken in order to efficiently discover the colossal closed itemsets from high-dimensional datasets.

Section 4.2 introduces the approach to discovering large cardinality closed itemsets from high-dimensional datasets. The section continues by describing the implementation of the transposition operation on the original dataset. An example of the input dataset, as well as its transposed version, is also given. In addition, the proposed user defined threshold is described, including the definitions and examples of colossal closed itemsets.

Section 4.3 introduces and defines the closedness-checking approach proposed. This section includes examples and proofs that demonstrate the correctness of the method.
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

Section 4.4 introduces the proposed data structure that enables efficient search for colossal closed itemsets. An illustration of the structure and its relationship with the dataset is also presented.

Section 4.5 introduces the algorithm DisClose, developed on the basis of the search strategies, the closeness-checking approach and the data structure proposed. Examples and proofs of the search process are presented. Space and time analysis of DisClose is also discussed.

Finally Section 4.6 summarizes the chapter.

4.1 Overview

Figure 4.1 shows the overall steps undertaken in mining the colossal closed itemset proposed in this thesis. Mining for colossal closed itemsets from high dimensional

```
Figure 4.1: Colossal Closed Itemset Mining Procedures
```
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

dataset requires 2 major steps: (i) discovering the colossal itemsets; and (ii) identifying the closed itemsets. Several strategies shown in Figure 4.1 were proposed in order to accomplish these two main steps. These proposed strategies are discussed in detail in the subsequent sections.

4.2 Proposed Search Strategies

Finding the most common itemsets in high-dimensional data leads to the likelihood of finding itemsets common to most situations (r ids - row ids) but which contain only a few of the items (columns). The computational complexity of having to obtain all closed frequent itemsets, usually results in algorithms that struggle to find larger itemsets. It may be that itemsets common to only a few situations (r ids), which contain a larger number of items (columns), may provide interesting insights into the nature of the dataset. Therefore, to discover these large closed itemsets, rather than generating candidate itemsets and checking for the closure property, this study proposes an approach which begins with closed itemsets (entire transactions) that exist in the dataset, which may have very small support (usually only one, unless duplicate transactions exist). From this collection of closed itemsets, smaller itemsets are built with higher support.

4.2.1 Bottom-up Row-enumeration Search

Extracting large itemsets involves determining the column (attribute) that has the highest cardinality of values associated with it in the dataset. This implies that the search strategy can be based on a top-down column enumeration. Figure 4.2 shows an example of a top-down column enumeration tree for a dataset which contains five items, \{a, b, c, d, e\}.

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 unfeasible.

It makes sense to search for closed itemsets based on the number of rows because, as previously stated, it is relatively small compared to the number of columns in high-dimensional datasets (Pan et al., 2003; Cong et al., 2004; Liu et al., 2009). 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.

Figure 4.2: Top-down column enumeration tree
4.2.2 Transposed Table

Since the proposal of the method by Rioult et al. (2003), transposition has been widely used by algorithms that discover closed itemsets from high-dimensional datasets (Pan et al., 2003; Cong et al., 2004a; Liu et al., 2009). High-dimensional datasets in domain such as biomedical engineering, telecommunications, geospatial data, and climate data are known to be dense (Han et al., 2002). A dataset tend to be dense in that they have any or all of the following properties: (i) many frequently occurring items; (ii) strong correlation between several items; (iii) many items in each record (Bayardo et al., 1999). Table 4.1 shows an example of a discretized high-dimensional dataset. 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.

Table 4.2 represents the transposed version of Table 4.1. It can be observed that the transposed dataset provides a sparser representation of the original input.
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

Table 4.2: Example of Transposed Dataset

<table>
<thead>
<tr>
<th>Item</th>
<th>Tidset</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>1</td>
</tr>
<tr>
<td>a₂</td>
<td>2</td>
</tr>
<tr>
<td>b₁</td>
<td>1</td>
</tr>
<tr>
<td>c₁</td>
<td>1</td>
</tr>
<tr>
<td>c₂</td>
<td>2</td>
</tr>
<tr>
<td>d₁</td>
<td></td>
</tr>
<tr>
<td>d₂</td>
<td>1</td>
</tr>
<tr>
<td>e₁</td>
<td></td>
</tr>
<tr>
<td>e₂</td>
<td>1</td>
</tr>
<tr>
<td>f₁</td>
<td>1</td>
</tr>
<tr>
<td>f₂</td>
<td>2</td>
</tr>
<tr>
<td>g₁</td>
<td></td>
</tr>
<tr>
<td>g₂</td>
<td>1</td>
</tr>
<tr>
<td>h₁</td>
<td>1</td>
</tr>
<tr>
<td>h₂</td>
<td>2</td>
</tr>
<tr>
<td>i₁</td>
<td>2</td>
</tr>
<tr>
<td>i₂</td>
<td>1</td>
</tr>
<tr>
<td>j₁</td>
<td></td>
</tr>
<tr>
<td>j₂</td>
<td>1</td>
</tr>
<tr>
<td>k₁</td>
<td>1</td>
</tr>
<tr>
<td>k₂</td>
<td></td>
</tr>
<tr>
<td>l₁</td>
<td></td>
</tr>
<tr>
<td>l₂</td>
<td>1</td>
</tr>
<tr>
<td>m₁</td>
<td>2</td>
</tr>
<tr>
<td>m₂</td>
<td>1</td>
</tr>
<tr>
<td>n₁</td>
<td>1</td>
</tr>
<tr>
<td>n₂</td>
<td>2</td>
</tr>
</tbody>
</table>

dataset. As a result of this simplification, the method of transposition is utilised in the algorithm proposed here.

4.2.3 Minimum Cardinality Threshold, \textit{mincard}

As highlighted in Chapter 3, it is impractical to mine all closed itemsets from high-dimensional datasets, due to their cardinality. Various types of threshold have thus been proposed in order to reduce the search space. The most common of these thresholds is the minimum support threshold, \textit{minsup}, which assists in reducing the search space based on the frequency of occurrence. Essentially, a low support
threshold may incur a combinatorial explosion in the number of closed frequent itemsets, thus limiting the search for large cardinality (or exceptional) closed itemsets.

In this particular study, as the objective is to focus on the discovery of large closed itemsets, the search process is stopped upon reaching a threshold parameter value for the minimum itemset cardinality, mincard.

**Definition 4.1 (Cardinality)** The cardinality of an itemset \( \alpha \) refers to the number of items in \( \alpha \). This is denoted as \(| \alpha |\).

**Example 4.1 (Cardinality)** The cardinality of the itemset \{b_1, e_2, i_2, j_2, k_2, l_2, m_2\} in Table 4.1 is \(| \{b_1, e_2, i_2, j_2, k_2, l_2, m_2\} | = 7\).

**Definition 4.2 (Colossal itemset)** Given a minimum cardinality threshold, mincard, an itemset \( \alpha \) is colossal if \(| \alpha | \geq\) mincard.

The search space of the dataset can be safely pruned by using the cardinality constraint, because of its anti-monotone property.

**Property 4.1 (anti-monotone)** If a rowset \( \beta \) has its associated itemset, \( \alpha = I(\beta) \), such that \(|\alpha| < \text{mincard}\), then for any \( \beta' \supseteq \beta \) it must be that \(|I(\beta')| < \text{mincard}\).

Combining the anti-monotone property with the definition of closure (Definition 2.12) ensures the following property.

**Property 4.2 (at-threshold)** If a rowset \( \beta \) has its associated itemset, \( \alpha = I(\beta) \), such that \(|\alpha| = \text{mincard}\), then for any \( \beta' \supseteq \beta \) it must be that \(|I(\beta')| < \text{mincard}\).
**CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS**

1 \implies (14): a_1, b_1, c_1, d_2, e_2, f_1, g_2, h_1, i_2, j_2, k_2, l_2, m_2, n_2

12 \implies (7): b_1, d_2, e_2, g_2, j_2, k_2, l_2

123 \implies (5): b_1, e_2, g_2, k_2, l_2

1234 \implies (4): b_1, e_2, k_2, l_2

12345 \implies (2): b_1, k_2

1235 \implies (3): b_1, g_2, k_2

124 \implies (5): b_1, e_2, j_2, k_2, l_2

1245 \implies (2): b_1, k_2

125 \implies (3): b_1, g_2, k_2

13 \implies (8): a_1, b_1, e_2, g_2, i_2, k_2, l_2, m_2

134 \implies (6): b_1, e_2, i_2, k_2, l_2, m_2

1345 \implies (4): b_1, i_2, k_2, m_2

135 \implies (6): a_1, b_1, g_2, i_2, k_2, m_2

14 \implies (7): b_1, e_2, i_2, j_2, k_2, l_2, m_2

145 \implies (4): b_1, i_2, k_2, m_2

15 \implies (6): a_1, b_1, g_2, i_2, k_2, m_2

2 \implies (14): a_2, b_1, c_2, d_2, e_2, f_2, g_2, h_2, i_1, j_2, k_2, l_2, m_1, n_2

23 \implies (9): b_1, c_2, e_2, f_2, g_2, h_2, k_2, l_2, n_2

234 \implies (8): b_1, c_2, e_2, f_2, h_2, k_2, l_2, n_2

2345 \implies (6): b_1, c_2, f_2, h_2, k_2, n_2

235 \implies (7): b_1, c_2, f_2, g_2, h_2, k_2, n_2

24 \implies (10): a_2, b_1, c_2, e_2, f_2, h_2, j_2, k_2, l_2, n_2

245 \implies (6): b_1, c_2, f_2, h_2, k_2, n_2

25 \implies (7): b_1, c_2, f_2, g_2, h_2, k_2, n_2

3 \implies (14): a_1, b_1, c_2, d_2, e_2, f_2, g_2, h_2, i_2, j_1, k_2, l_2, m_2, n_2

34 \implies (11): b_1, c_2, d_1, e_2, f_2, g_2, h_2, i_2, k_2, l_2, m_2, n_2

345 \implies (9): b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2

35 \implies (12): a_1, b_1, c_2, d_1, e_2, f_2, g_2, h_2, i_2, j_1, k_2, m_2, n_2

4 \implies (14): a_2, b_1, c_2, d_1, e_2, f_2, g_1, h_2, i_2, j_2, k_2, l_2, m_2, n_2

45 \implies (9): b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2

5 \implies (14): a_1, b_1, c_2, d_1, e_1, f_2, g_2, h_2, i_2, j_1, k_2, l_1, m_2, n_2

Figure 4.3: Example of colossal itemsets (highlighted in grey) for

\[ \text{mincard} = 7 \]

Figure 4.3 shows all frequent itemsets obtained from Table 4.1 and the colossal itemsets discovered (highlighted in grey) using the bottom-up search order of the row enumeration tree. The size of the itemset is indicated in parentheses for each row value and the itemset that it represents. By applying the \[ \text{mincard} \] threshold,
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

the branch exploration is stopped once the cardinality of the associated itemset falls below the threshold value. This is described in the following example.

**Example 4.2 (Colossal Itemset)** Suppose a user defines the minimum cardinality threshold value, \( \text{mincard} = 7 \). In Figure 4.3, the search space is explored beginning from the largest cardinality itemset, \(| \alpha | = 14 \) and it will stop when \(| \alpha | < 7 \). A total of 17 colossal itemsets are discovered.

### 4.3 Proposed 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. This can be observed in Figure 4.3, where the same colossal itemsets are discovered in rowsets \( \{2, 5\} \) with \( \{2, 3, 5\} \) and rowsets \( \{4, 5\} \) with \( \{3, 4, 5\} \). Producing duplicate colossal itemsets leads to redundancy. Although in this example, only a small number of colossal itemsets discovered are redundant, in real life datasets such redundant itemsets can occur in very large numbers, which leads to a commensurate decrease in performance.

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. Several closedness-checking methods have been discussed in Chapter 3, for example in the studies made by Grahne and Zhu (2003), Pan et al., (2003), Zaki and Hsiao (2005), and Liu et al., (2006).

To take advantage of the second restriction in making the mining of colossal itemsets more efficient, in this study, a method which is based on a unique
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

generator is developed. To define the unique generator, the study begins by providing the definition for itemset generator and tidset generator as follows:

**Definition 4.3 (Itemset Generator)** Given a dataset \( T \), an itemset \( \alpha \) is an itemset generator if no proper subset \( \alpha' \subset \alpha \) exists such that the support of \( \alpha \) is the same as the support of \( \alpha' \).

**Example 4.3 (Itemset Generator)** In Figure 4.3, itemset \( \{b_1, e_2, j_2, k_2, l_2\} \) for rowset \( \{1, 2, 4\} \), with support = 3, is an itemset generator as there are no itemsets that are a subset of \( \{b_1, e_2, j_2, k_2, l_2\} \) with the same support. Itemset \( \{b_1, e_2, g_2, k_2, l_2\} \) for rowset \( \{1, 2, 3\} \) is not an itemset generator as a subset of \( \{b_1, e_2, g_2, k_2, l_2\} \supset \{b_1, g_2, k_2\} \) exists at rowset \( \{1, 2, 5\} \) with the equivalent support.

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

**Example 4.4 (Rowset Generator)** In Figure 4.3, rowset \( \{1, 2, 3, 5\} \) is not a rowset generator as rowset \( \{1, 2, 5\} \subset \{1, 2, 3, 5\} \) also contains the itemset \( \{b_1, g_2, k_2\} \). However, \( \{1, 2, 4, 5\} \) is a rowset generator as there are no subsets of the rowset value that contains \( \{b_1, k_2\} \).

Similarly, the equivalence class of rowsets \( \beta_i \) with the same itemset \( \alpha \) 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.

As stated at the beginning of Section 4.3, the largest closed itemsets could exist in an entire transaction unless duplicate rows exist. To construct smaller closed itemsets from larger ones, the following property is used:

**Theorem 1** Suppose \( \alpha_1 \) and \( \alpha_2 \) are closed itemsets, with \( \alpha_1 \neq \alpha_2 \). Let \( \alpha = \alpha_1 \cap \alpha_2 \).

If \( \alpha \neq \emptyset \) then \( \alpha \) is a closed itemset.

**Proof:** There are three cases to consider:

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

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

For Case 1 and Case 2, in order for \( \alpha_1 \) and \( \alpha_2 \) to be closed itemsets with one a proper subset of the other, it must be the case (by the definition of closed itemset) that they have different support. However, it is known that such a situation exists.

Consider any closed itemset \( \alpha_1 \) with support larger than one and select any row \( r_i \) containing \( \alpha_1 \) (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 fundamental set theory, so the result holds.

3. **Case 3:** \([ \alpha_1 \) and \( \alpha_2 \) are incomparable]. *Observe that \( \alpha \subset \alpha_1 \) and \( \alpha \subset \alpha_2 \).*

In this particular case, it is demonstrated that \( \alpha \) is a closed itemset by contradiction. Assuming \( \alpha \) is not a closed itemset, there then exists some item \( i \) such that \( \alpha_i = \alpha \cup \{i\} \) has the same support as \( \alpha \). If \( i \notin \alpha_1 \), then all transactions
in $T_{a_i} - T_a$ are not in $T_a$, but they are in $T_{a_i}$. Thus $i$ must be in $a_1$. However, if $i \in a_1$ (and not in $a$) then $i \notin a_2$ and the same contradiction argument applies. Thus the assumption that $a$ is not a closed itemset must be incorrect.

An example can be seen in Figure 4.3, let $a_1 = \{b_1, c_2, e_2, f_2, h_2, k_2, l_2, n_2\}$ which occurs at rowset $\{2, 3, 4\}$ and $a_2 = \{b_1, c_2, f_2, g_2, h_2, k_2, n_2\}$ which occurs at rowset $\{2, 3, 5\}$. $a = \{b_1, c_2, f_2, h_2, k_2, n_2\}$ at rowset $\{2, 3, 4, 5\}$ is a closed itemset in whereby $a \subseteq a_1$ and $a \subseteq a_2$.

Every closed itemset that is not one of the transactions can be produced by the intersection of a collection of closed itemsets. Consider a closed itemset $a$ and its corresponding rowset $\beta = T_a$. As $a$ is a closed itemset, $a = \bigcap_{i \in \beta} \alpha_i$, where $(t_i, \alpha_i) \in T$. However, there may be many subsets of $\beta$ for which $I(\beta) = a$. If the rowset enumeration were to perform as the control strategy for the search process, it is likely that the same closed itemsets would be found many times.

The following observation enables the proposed closedness-checking method of this study to discover a closed itemset using only one of the rowsets. According to the definition of closure (Definition 2.12), for every closed itemset $a$, there is a unique rowset $\beta$ that is a closed rowset.

**Definition 4.5 (Unique Generator)** Given the closed rowset $\beta = \{t_1, t_2, \ldots, t_k\}$, with $t_i < t_j$ for all $i < j$, the smallest index $j$ for which $\beta_j = \{t_1, t_2, \ldots, t_j\}$ is a generator of $\beta$ is a unique rowset generator for the itemset $a$.

It is simple to determine if a rowset $\beta'$ is the unique generator. Let $\beta = T(I(\beta'))$. If $\beta' = \beta$, then the answer is that $\beta'$ is the unique generator. If $\beta' \subset \beta$, $\beta'$ is determined whether it is a prefix of $\beta$ when the rowsets are written as lists in ascending order. If $\beta'$ is not a prefix of $\beta$, then $\beta'$ can be ignored and this branch of the search space is pruned.
Example 4.5 (Unique Generator) From Figure 4.3, $\beta' = \{2, 3\}$ is a unique generator as the closed rowset for $\{b_1, c_2, e_2, f_2, g_2, h_2, k_2, l_2, n_2\} = \{2, 3\}$. However, $\beta'' = \{2, 5\}$ is not a unique generator. This is because the closed rowset for $\{b_1, c_2, f_2, g_2, h_2, k_2, n_2\} = \{2, 3, 5\}$ and $\{2, 5\}$ is not a prefix of $\{2, 3, 5\}$.

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.

4.4 CR-Tree (Compact Row-Tree)

To assist the efficiency of the search, a compact tree data structure is built to store the itemsets from $T'$. 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, $T'$. These sets 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, as will be further discussed in the following section.

Figure 4.4 shows the relationship between the CR-Tree and the transposed table $T'$. The structure of the CR-Tree is similar to the FR-Tree (Liu et al., 2009). 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. Figure 4.5 shows an example of the nodes in the CR-Tree, representing the rowset values for Table 4.2.
**Lemma 4.1** The CR-Tree nodes represent all the rowset, $\beta$, values of a complete row enumeration tree.

**Proof** Let $N = \{n_i, n_{i+1}, \ldots, n_k\}$ be the set of nodes where $i = 1$ and $k$ is the largest rid 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 rid value of the dataset. Each $m_j = n_i \cup n_{i+l}$ where $l = \{1,
Therefore all $\beta$ values are traversed until $j = k$ for the maximum tree level of $k-1$.

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. Several optimization strategies are also proposed in order to guarantee that the CR-Tree will not miss any itemsets during the process and at the same time that it will discard those that it deems redundant. These strategies will act as a proof of correctness on the data structure proposed and will be described in detail in the next section.
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

4.5 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 $T'$ of table $T$. DisClose, shown in Algorithm 4.1, will search the row enumeration space and, for each rowset, $\beta$, check whether it is the unique generator in the equivalence class of rowsets for $I(\beta)$. It is to be 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 amount of memory (Han et al., 2000; Pan et al., 2003; Zaki and Hsiao, 2005; Liu et al., 2006). For this reason, the general processing order for the rowsets is equivalent to the depth-first search of the row enumeration tree.

4.5.1 Major steps of DisClose

Algorithm 4.1 shows the main steps of the algorithm DisClose. The example input dataset in Table 4.1 is used to demonstrate DisClose in the following discussions.

The algorithm begins with the transposition operation that transforms table $T$ to the transposed table $T'$ as shown in Table 4.2. Then, the CR-Tree (Compact Rowset Tree) is built, as demonstrated in Figure 4.4.

After initialization of the set of colossal closed itemsets $CCI$ to be empty, the subroutine Colossal is called to deal with the transposed table $T'$ 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, $T'$ and the minimum cardinality threshold, $\text{mincard}$, as the parameter and performs the search for colossal closed itemsets.

There are seven sections in the subroutine Colossal, which will be explained one by one. Assume this example uses the $\text{mincard}$ threshold value of 7.

The first section is steps 6 - step 7. Each node at the first level of the CR-Tree attempts to store the itemset from the transposed table $T'$ into the node by
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

Algorithm 4.1: 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 $T'$
2. Build CR-Tree
3. Initialize $CCI = \emptyset$
4. Call Subroutine Colossal ($T'$, $mincard$)

**Subroutine Colossal ($T'$, $mincard$)**

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

**Subroutine Closed ($mincard$)**

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

ensuring that the itemset in each column value of the transposed table $T'$ satisfies the $mincard$ threshold. An itemset that satisfy the $mincard$ threshold is then stored in each node; otherwise, it is not stored in the node as it will not contribute to obtaining
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

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 $T'$ plays in the search process.

Figure 4.6 shows the CR-Tree which contains the itemsets stored for the nodes at level, $l = 0$, after applying steps 6 - step 7 from Figure 4.3.

The second section is steps 8 - step 10. For each node in the CR-Tree, an itemset intersection is performed. By using a depth-first search, DisClose produces the sequence of $\beta \Rightarrow I(\beta)$ shown in Figure 4.3. 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.

Optimization strategy $S1$: If the size of the itemset is less than the minimum cardinality threshold, $|a| < \text{mincard}$, then there is no need to perform any further operation on the itemset. If the itemset size is less than the specified threshold, then a further intersection should lead to a much smaller or equivalent

![Figure 4.6: Itemset stored at level $l = 0$, of the CR-Tree](image-url)
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

Example 4.5 (Optimization Strategy S1) Figure 4.7 shows an example of a node in the CR-Tree that contains an itemset with cardinality is less than the specified threshold. Node 3 at level \( l = 2 \) contains itemset \( \{b_1, e_2, g_2, k_2, l_2\} \) where \( |\{b_1, e_2, g_2, k_2, l_2\}| = 5 \). As the cardinality of the itemset is less than the \textit{mincard} threshold, further intersection between Node 3 and Node 4 at level \( l = 2 \), \( \{b_1, e_2, g_2, k_2, l_2\} \cap \{b_1, e_2, j_2, k_2, l_2\} = \{b_1, e_2, k_2, l_2\} \), leads to a smaller cardinality itemset, \( |\{b_1, e_2, k_2, l_2\}| = 4 \), which is stored in Node 4 at level \( l = 3 \). Therefore, optimization strategy \( S1 \) is required to prevent storage of itemsets that do not satisfy the desired threshold, which in turn should lead to a reduction in memory space and processing time.

Lemma 4.2 Each node of the CR-Tree only stores one value at a time for rowset, \( \beta \), with \( |\beta| \) = node level.

Proof In Figure 4.8, suppose at level \( l = 2 \), the rowsets stored at node \( n_4 = \{1, 2, 4\} \) and node \( n_5 = \{1, 3, 5\} \). To obtain rowset, \( \beta \), for child node \( m_5 \) at level \( l = 3 \), the union of the parent \( \beta \) values will produce, \( n_4 \cup n_5 = \{1, 2, 4\} \cup \{1, 3, 5\} = \{1, 2, 3, 4, 5\} \). However, the rowset \( \{1, 2, 3, 4, 5\} \) is not represented by node \( m_5 \). This is
because based on the depth-first strategy, the itemset for \( \beta = \{1, 2, 3, 4, 5\} \) have been discovered at \( l = 4 \).

Based on Lemma 4.2, step 12 performs the Optimization strategy \( S_2 \) to prevent storage of itemsets with rowset values larger than the node level of the CR-Tree.

Optimization strategy \( S_2 \): If the size of the rowset \( |\beta| \) is greater than the level where the node is present, then there is no need to store the itemset \( \alpha \) obtained. This is explained in the following example.

Example 4.6 (Optimization Strategy S2) As DisClose performs a depth-first search of the CR-Tree, rowsets of larger cardinality have been obtained earlier and further steps will only lead to a repetition of the itemset with the same rowset value. Figure 4.9 shows an example in which Optimization strategy \( S_2 \) is applied. Suppose at level \( l = 2 \), node 4 contains the itemset \( \{b_1, e_2, j_2, k_2, l_2\} \) with the rowset value of \( \{1, 2, 4\} \) and node 5 contains the itemset \( \{a_1, b_1, g_2, i_2, k_2, m_2\} \) with the rowset value of \( \{1, 3, 5\} \). The intersection between these nodes will produce the itemset \( \{b_1 k_2\} \) with a rowset value of \( \{1, 2, 3, 4, 5\} \). However, \( |\{1, 2, 3, 4, 5\}| = 5 \) is greater than the node level value, \( l = 3 \). Therefore the itemset will not be stored at node 5 where \( l = \)

---

Figure 4.8: Example on Lemma 4.2
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

3. As observed, the itemset with the same rowset value has already been stored at node 5 where \( l = 4 \).

**Lemma 4.3** If discovered itemset, \( \alpha_1 \subseteq \alpha_2 \) where \( \alpha_2 \) is the existing itemset in the node, the itemset \( \alpha_1 \) will not replace \( \alpha_2 \) although \( \beta_1 \neq \beta_2 \).

**Proof** In Figure 4.10, suppose \( \alpha_2 = \{\beta_1\} = \{1, 2, 4\} \) and \( \alpha_1 = \{\beta_2\} = \{1, 3, 4\} \) at level \( l = 2 \), where \(|\beta_1| = |\beta_2|\). If \( \alpha_1 \subseteq \alpha_2 \), this means that \( \alpha_1 \) also exists in \( \{\beta_2\} \). Therefore, \( \alpha_1 = \beta_1 \cup \beta_2 = \beta \), where \(|\beta| > |\beta_1|, |\beta_2|\). Thus \( \alpha_1 \) will exists in \( \beta = \{1, 2, 3, 4\} \) stored in node \( m_4 \) at level, \( l = 3 \) of the CR-Tree.

Figure 4.9: Example of Optimization strategy S2

Figure 4.10: Example on Lemma 4.3
At step 13, Optimization strategy S3 is applied based on Lemma 4.3 in order to ensure that the itemset obtained is not a subset of an already existing itemset in the child node.

**Optimization strategy S3**: If the current itemset $\alpha$ obtained is a subset of an already existing itemset $\alpha'$, $\alpha \subseteq \alpha'$, for the particular child node in the CR-Tree, then itemset $\alpha$ can be discarded.

**Example 4.7** (Optimization Strategy S3) Figure 4.11 shows an example in which Optimization strategy S3 is applied. Suppose Node 5 at level $l = 1$ already contains an itemset from the earlier iteration. The result of the intersection between Node 4 and Node 5 at level $l = 0$ produces an itemset which is a subset of the already stored itemset in the child node, $\{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\} \subseteq \{a_1, b_1, c_2, d_1, f_2, g_2, h_2, i_2, j_1, k_2, m_2, n_2\}$. Although itemset $\{a_1, b_1, c_2, d_1, f_2, g_2, h_2, i_2, j_1, k_2, m_2, n_2\}$ occurs in rowset $\{3, 5\}$ and $\{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\}$ in rowset $\{4, 5\}$, $\{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\}$ shows that $\{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\}$ also occurs in $\{3, 5\}$. The result of the intersection between the two

---

![Figure 4.11: Example of Pruning strategy S3](image-url)
rowsets has already been produced at the higher level node during the earlier iteration, due to the depth-first search strategy. This is shown in Figure 4.11 where Node 5 at level \( l = 2 \) contains the itemset \( \{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\} \) with rowset value \( \{3, 4, 5\} \). Therefore, Optimization strategy \( S3 \) is applied on itemset \( \{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\} \) with rowset value \( \{4, 5\} \).

**Step 14** then stores an 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 \( \alpha \) 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 \( \beta \) that contains a rid value that does not exist in rowset \( \beta' \) of the parent node.

**Example 4.8 (Closedness-checking in CR-Tree)** Figure 4.12 shows an example of checking whether the itemset \( \alpha \) stored at node 3 of level \( l = 1 \) is a closed itemset. As \( \alpha \) occurs in rowset \( \{2, 3\} \), it needs to be compared with all the itemsets in the child nodes to which it points, in order to check whether it also occurs in another rid value. This check whether \( \alpha \) occurs at rid 1, rid 4 or rid 5. Each child node at level \( l = 2 \) contains the itemset with the rowset values containing one of the rids. In this
By applying the proposed closedness-checking method proposed, an itemset $\alpha$ is not closed if it also occurs in another rid value that is not already in $\beta'$. As soon as a rowset holding a copy of $\alpha$ is found, further comparison with other child nodes is unnecessary.

As stated previously, the use of the unique generator requires only a small amount of computation based on the number of rows; hence this algorithm can run with very little memory. Another advantage is that the method allows the algorithm $DisClose$ to simply write each encountered closed itemset, and not keep a copy in memory for later comparison.

At step 17 the subroutine $Colossal$ is activated if the itemset is found to not be closed. A further search for colossal itemsets is then continued at the next node level.

Step 18 outputs the itemset that is found to be closed into the set of colossal closed itemsets, $CCI$.

At step 19 the subroutine $Colossal$ is called to further continue the search until all potential node values of the $CR$-$Tree$ have been traversed.
1 ⇒ (14): \{a_1, b_1, c_1, d_2, e_2, f_1, g_2, h_1, i_2, j_2, k_2, l_2, m_2, n_1\}
12 ⇒ (7): \{b_1, d_2, e_2, g_2, j_2, k_2, l_2\}
123 ⇒ (5): \{b_1, e_2, g_2, k_2, l_2\}
1234 ⇒ (4): \{b_1, e_2, k_2, l_2\}
12345 ⇒ (2): \{b_1, k_2\}
1235 ⇒ (3): \{b_1, g_2, k_2\}
124 ⇒ (5): \{b_1, e_2, j_2, k_2, l_2\}
1245 ⇒ (2): \{b_1, k_2\}
125 ⇒ (3): \{b_1, g_2, k_2\}
13 ⇒ (8): \{a_1, b_1, e_2, g_2, i_2, k_2, l_2, m_2\}
134 ⇒ (6): \{b_1, e_2, i_2, k_2, l_2, m_2\}
1345 ⇒ (4): \{b_1, i_2, k_2, m_2\}
135 ⇒ (6): \{a_1, b_1, g_2, i_2, k_2, m_2\}
14 ⇒ (7): \{b_1, e_2, i_2, j_2, k_2, l_2, m_2\}
145 ⇒ (4): \{b_1, i_2, k_2, m_2\}
15 ⇒ (6): \{a_1, b_1, g_2, i_2, k_2, m_2\}
2 ⇒ (14): \{a_2, b_1, c_2, d_2, e_2, f_2, g_2, h_2, i_1, j_2, k_2, l_2, m_1, n_2\}
23 ⇒ (9): \{b_1, c_2, e_2, f_2, g_2, h_2, k_2, l_2, n_2\}
234 ⇒ (8): \{b_1, c_2, e_2, f_2, h_2, k_2, l_2, n_2\}
2345 ⇒ (6): \{b_1, c_2, f_2, h_2, k_2, n_2\}
235 ⇒ (7): \{b_1, c_2, f_2, g_2, h_2, k_2, n_2\}
24 ⇒ (10): \{a_2, b_1, c_2, e_2, f_2, h_2, j_2, k_2, l_2, n_2\}
245 ⇒ (6): \{b_1, c_2, f_2, h_2, k_2, n_2\}
25 ⇒ (7): \{b_1, c_2, f_2, g_2, h_2, k_2, n_2\}
3 ⇒ (14): \{a_1, b_1, c_2, d_1, e_2, f_2, g_2, h_2, i_2, j_1, k_2, l_2, m_2, n_2\}
34 ⇒ (11): \{b_1, c_2, d_1, e_2, f_2, h_2, i_2, k_2, l_2, m_2, n_3\}
345 ⇒ (9): \{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\}
35 ⇒ (12): \{a_1, b_1, c_2, d_1, e_2, f_2, g_2, h_2, i_2, j_1, k_2, m_2, n_2\}
4 ⇒ (14): \{a_2, b_1, c_2, d_1, e_2, f_2, g_1, h_2, i_2, j_2, k_2, l_2, m_2, n_2\}
45 ⇒ (9): \{b_1, c_2, d_1, f_2, h_2, i_2, k_2, m_2, n_2\}
5 ⇒ (14): \{a_1, b_1, c_2, d_1, e_1, f_2, g_2, h_2, i_2, j_1, k_2, l_1, m_2, n_2\}

Figure 4.13: Colossal closed itemsets with \textit{mincard} = 7

There are a total of 15 colossal closed itemsets found for the dataset in Table 4.1 with \textit{mincard} = 7, as shown in Figure 4.13.
CHAPTER 4. DisClose: MINING COLOSSAL CLOSED ITEMSETS

4.5.2 Space and time analysis of DisClose

For a table $T$ with $n$ rows and $m$ dimensions where each dimension has a maximum of $k$ distinct items (attribute values), without considering the rowset values, the space complexity of the transposed table, $T'$ is $O(mn)$. Using the cardinality threshold, $\text{mincard}$, means that further memory space is not required for all itemsets with a size of less than the specified constraint. This means that for a transposed table, $T'$ with $km$ rows, additional memory does not need to be used for nodes that contain itemsets of less than $(km-\text{mincard})$ because the search will stop at an itemset with size $(km-\text{mincard})$. The space complexity of the CR-tree depends on the column value of the transposed table. For a transposed table, $T'$ with $m$ dimensions, the CR-tree requires $O\left(\frac{n(n+1)}{2}\right)$ as the number of nodes for each level decreases by 1 as the level of the tree increases.

For time complexity, the transformation from table $T$ to $T'$ requires $O(kmn)$ time to collect the rids for each distinct item. The process of building the CR-Tree involves, time complexity of at most $O[n(n-1)]$ because there are at most $n$ rids in $T'$ and each rid in the CR-Tree has at most $(n-1)$ children to be searched. At each node, DisClose needs to process each itemset with $O(1)$ time, thus a total of $O(km)$ at most.

4.6 Summary

This chapter presents the proposed colossal closed itemset mining algorithm, DisClose, which implements the data structure that has been proposed in order to efficiently discover colossal closed itemsets based on the proposed search strategy and closedness-checking method. The used of the method is explained through illustrative examples.

The following chapter analyses the performance of DisClose on several synthetic and real high-dimensional datasets. The effectiveness of the algorithm will
be compared to several state-of-the art algorithms by adapting similar requirements to enable a fair comparison.
Chapter: 5

Experimental Evaluation

This chapter presents the results of the performance study of DisClose and confirms that the program design has been realized.

This chapter begins with Section 5.1, which introduces the environment in which algorithm DisClose was implemented. This includes the programming language applied and the specification of the machine on which the algorithm was tested. This section also provides a list of state-of-the art algorithms selected for comparison purposes and the datasets that were selected for evaluation use.

Section 5.2 provides an evaluation of the performance of DisClose on synthetic datasets from different points of view such as the effect of the change of mincard, the number of dimensions, the number of rows and the cardinality of each dimension.

Section 5.3 provides an evaluation of the performance of DisClose on real datasets. Descriptions of the selected real datasets are also provided. This section also provides the discretization method that has been applied to the real application datasets.

Finally Section 5.4 summarizes the chapter.
CHAPTER 5. EXPERIMENTAL EVALUATION

5.1 Experimental Setting

Algorithm DisClose was implemented using C++. The set of experiments was performed on a PC with a 2.66 GHz Intel Core 2 Quad CPU Q9400 with 4.00 GB RAM and 150 GB hard disk.

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 discussed in the previous chapter. These algorithms are:

(i) **FP-Close** (Grahne and Zhu, 2003). This is a representative of the column enumeration-based algorithms, which won the FIMI’03 best implementation award. The implementation of FP-Close was obtained from the developer Christian Borgelt’s website through his implementation of FP-Growth, which has the option to discover closed frequent itemsets (Borgelt, 2005).

(ii) **CARPENTER** (Pan et al., 2003): This is a representative of bottom-up row enumeration-based algorithms. Carpenter searches the tree from the smallest rowset and builds larger rowset values. The source of implementation was also downloaded from Christian Borgelt’s website (Borgelt, 2011).

(iii) **D-Miner** (Besson et al., 2005): This is a representative of constraint-based mining algorithms, which use the minimum cardinality threshold as one of the constraints for their search strategy. For the algorithm D-Miner, the source of the implementation was downloaded from the author’s website (Besson et al., 2005).

(iv) **TTD-Close** (Liu et al., 2009): This algorithm is a representative of the top-down row enumeration search based set of algorithms. The search begins from the largest rowset value and moves its way down the search tree. For algorithm TTD-Close, the source of the implementation was obtained from its authors.
CHAPTER 5. EXPERIMENTAL EVALUATION

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

5.1.1 Challenges in Comparisons

Amongst the selected algorithms listed above, only D-Miner has been found to apply the minimum cardinality threshold, mincard. As stated previously, D-Miner is a constraint-based algorithm which uses the minimum support, minsup, and the minimum cardinality, mincard, thresholds to discover concepts (closed itemsets). As the objective of this study is to discover colossal closed itemsets, direct comparison can be made with DisClose if the minimum support, minsup threshold is set to 0. This means that D-Miner will only search the large cardinality (colossal) closed itemsets.

Other existing itemset mining algorithms – particularly those that find closed itemsets, which includes FP-Close, CARPENTER, and TTD-Close – are routinely presented with running times given for varying thresholds of support. As DisClose begins by searching and storing the colossal closed itemsets using the cardinality threshold, a direct comparison to these previous techniques is difficult. If one of the previous algorithms were given a support threshold greater than 1, it would certainly not find many of the largest-cardinality closed itemsets. Similarly, if DisClose were given a cardinality threshold bigger than 1, it would certainly not find many of the most frequent closed itemsets. The only way to compare the algorithms is to present both with a threshold of 1, essentially asking each algorithm to find all closed itemsets. A strength of DisClose is that it bypasses the huge number of small-cardinality, high-frequent closed itemsets and focuses almost immediately on potentially valuable closed itemsets (especially for high-dimensional data). This type of complete closed itemsets search does not address the true intent of either DisClose or the existing closed itemset mining algorithms.
CHAPTER 5. EXPERIMENTAL EVALUATION

One approach is 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. Hence, DisClose was implemented in two versions that satisfy both conditions, in order for fair comparisons to be made.

5.2 Synthetic Datasets

The synthetic datasets were specifically constructed based on the implementation of the selected algorithms in order to obtain fair comparison between the methods. Synthetic datasets were used to test the performance of DisClose with the selected algorithms in terms of different aspects such as the effect of the change of mincard, the number of dimensions, the number of tuples and the cardinality of each dimension.

Synthetic datasets have been generated randomly using the IBM Quest Synthetic Data Generator based on three main parameters: the number of dimensions, the number of rows and the average length of the itemset in the dataset. To represent the synthetic dataset, the label T#L#N# is used, where T# is the number of rows, L# is average length of the itemset, and N# is the number of dimensions.
CHAPTER 5. EXPERIMENTAL EVALUATION

In reality, if a synthetic dataset was randomly generated several times with the equivalent parameters, this dataset will produce a different set of values every time. Hence, this will cause the answer set to be different each time the synthetic dataset is analyzed. However, in this thesis, only one dataset is generated for each parameter settings. This is because the objective of this thesis is to observe the efficiency of the algorithm proposed as compared to other selected algorithms in order to discover the colossal closed itemsets as opposed to differences in the number of colossal closed itemsets that could be discovered in each dataset.

5.2.1 Dimensionality (columns)

To test the performance of the four algorithms with respect to the number of dimensions, three datasets were generated with 4 K, 6 K and 10 K dimensions, respectively. Each dataset contains the same row value of 100 and an average itemset length of 2000.

5.2.1.1 DISCLOSE VS. D-MINER

Figure 5.1(a)-(c) shows the effect of changing the dimensionality on the runtime of DisClose with D-Miner based on the minimum cardinality threshold, mincard. In this set of experiments, DisClose presents better performance than D-Miner for all datasets.

It can be seen from Figure 5.1(a) that at a higher cardinality threshold, the differences 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 itemsets with mincard = 700. For D-Miner, after mincard = 2100, the algorithm took more than 12 hours to discover the colossal closed itemsets. The percentage of density of values present in T100L2000N4000 dataset is 50%. With the same number of rows and average size value of the itemset, as the number of dimensionality increases, the dataset becomes less dense.
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.1: The effect of changing dimensionality with $\text{mincard}$
CHAPTER 5. EXPERIMENTAL EVALUATION

The percentage of density of values present in T100L2000N6000 is 33.36%. Similar behaviour can be seen in Figure 5.1(b). Here, DisClose is able to discover colossal closed itemsets with mincard = 300. However, D-Miner still requires more than 12 hours mining itemsets with cardinality less than 2100 hence values for cardinality less than 2100 are not included in the figure.

For dataset T100L2000N10000, the percentage density of values present is 20.03%. Therefore, it can be observed in Figure 5.1(c) that DisClose is able to reach a much smaller cardinality itemset where mincard = 70. Similar to previous datasets, the execution of D-Miner requires more than 12 hours mining itemsets with cardinality less than 2100.

It is observed in Figure 5.1 that, with the changes in dimensionality (and also changes in density), DisClose performs better than D-Miner in discovering the colossal closed itemsets. DisClose also scales well as the execution time increases with the changes in the cardinality threshold for the changes in the dimensionality of the datasets.

5.2.1.2 DISCLOSE VS. FP-CLOSE, CARPENTER AND D-Miner

Figures 5.2, 5.3 and 5.4 respectively compare DisClose with FP-Close, CARPENTER and D-Miner by varying the number of attributes. As shown in Figure 5.2(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. There are a total of 27,994,019 colossal closed itemsets found when mincard = 600.

Figure 5.2(b) shows that as minsup decreases, the runtime of the three algorithms increases. However, the algorithms are only able to operate up to minsup = 95 before running out of memory. CARPENTER has the best execution time among these algorithms. It is more than twice as fast as FP-Close and more than 13 times faster than D-Miner.
This is due to the fact that the dataset is very dense and there exists a huge number of closed frequent itemsets with large cardinality. There are 4,908,256 closed itemsets discovered when $\text{minsup} = 95$ with the largest cardinality of closed
itemsets equal to 120. \textit{D-Miner} took the most time to discover the closed itemsets at this threshold.

As expected, Figure 5.3(a) shows a similar result i.e. as the mincard threshold decreases the execution time required for \textit{DisClose} to discover the colossal
closed itemsets increases. The T100L2000N6000 dataset is less dense, compared to dataset T100L2000N4000. As a result, DisClose is able to reach a smaller cardinality threshold of 200 with a maximum support threshold of 10. There are a total of 56,541,298 closed itemsets discovered at mincard = 200.

Figure 5.3(b) shows that only FP-Close and D-Miner can reach minsup = 81. FP-Close outperforms both D-Miner and CARPENTER. There are 309,914,567 closed itemsets discovered when minsup = 81 with the largest cardinality of the closed itemsets being equal to 20. CARPENTER can only reach minsup = 87. As expected, the time required to discover the closed itemsets increases as the minimum support decreases.

The T100L2000N10000 dataset is less dense as compared to the previous two datasets. This is shown in Figure 5.4(a), where DisClose is able to reach closed itemsets at a much lower mincard threshold. DisClose discovers a total of 215,610,238 closed itemsets with a lowest value of mincard = 10. The closed itemsets with mincard = 10 are found to have the maximum support threshold value of 24. The time required to discover the closed itemsets increases as the mincard threshold decreases.

For Figure 5.4(b), only FP-Close and D-Miner can reach lower support thresholds with values of 42 and 46 respectively before running out of memory. FP-Close performs better than D-Miner as D-Miner’s performance degrades for minsup values greater than 46. The decrease of the performance of D-Miner can be explained by the enormous influence of the high number of closed itemsets in this data. The runtime for CARPENTER could not be displayed as the algorithm can only reach minsup = 70. There are 1,483,324,975 closed itemsets discovered when minsup = 42 with the largest cardinality of closed itemsets equal to 13.

5.2.1.3 DisClose vs. TTD-Close

Figure 5.5 shows the results between DisClose and TTD-Close on the dataset 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.

Figure 5.5(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 \( \text{mincard} = 1700 \). There are a total of 78,717,638 closed itemsets that exists when \( \text{mincard} = 1700 \) having the maximum support of 6.

Figure 5.5(b) shows that \( TTD\text{-}Close \) could only reach \( \text{minsupt} = 97 \) with a total of closed itemsets of 58,505. \( TTD\text{-}Close \) runs out of memory probably due to the existence of larger cardinality itemsets at smaller \( \text{minsupt} \) thresholds. This shows
that for dense dataset, even at a high minsup threshold, the size of itemsets can become very large.

Figure 5.6(a) shows that as the number of dimensions increases, by increasing the mincard value, the time required to discover the colossal closed itemsets also increases. The number of closed itemsets is 50 times more when mincard = 2400 than when mincard = 2500, hence the precipitous increase in run

![Graph showing the relationship between mincard and time]  
(a)

![Graph showing the relationship between minsup and time]  
(b)

Figure 5.6: Comparison on T100L2000N6000 with TTD-Close
**CHAPTER 5. EXPERIMENTAL EVALUATION**

time. *DisClose* can reach closed itemsets with $\text{mincard} = 2400$. There are a total of 79,393,410 closed itemsets that exists when $\text{mincard} = 2400$ having a maximum support of 6.

Figure 5.6(b) shows that *TTD-Close* could still only reach $\text{minsup} = 97$ with a total number of closed itemsets of 21,451. This is due to the existence of colossal closed itemsets at a high support threshold.

Figure 5.7(a) shows similar behaviour as the number of dimension increases. *DisClose* is able to reach closed itemsets with $\text{mincard} = 5200$. There are a total of 77,556,906 closed itemsets that exists when $\text{mincard} = 5200$ having a maximum support of 5.

Figure 5.7(b) shows that *TTD-Close* may still only reach $\text{minsup} = 97$ with a total of closed itemsets of 117,251. *TTD-Close* runs out of memory due the existence of large cardinality itemsets at smaller $\text{minsup}$ thresholds. This shows that on very dense dataset, even at high $\text{minsup}$ threshold, the size of itemsets can become large.

### 5.2.2 Number of rows

To test the runtime of these algorithms with respect to the number of rows, two more datasets have been generated; one containing 150 rows and the other 200 rows, while the dimension is 4 K and the cardinality is 10. Figure 5.8(a)-(c) shows the effect on execution time by increasing the number of rows using the $\text{mincard}$ threshold.

To obtain a convenient comparison, the result for T100L2000N4000 is provided again here. Figure 5.8(b) shows that as the $\text{mincard}$ value decreases, the time required to discover the colossal closed itemsets increases. At the same $\text{mincard}$ value of 600, *DisClose* requires more time to discover colossal closed itemsets. There are a total of 227,674,614 closed itemsets that exists at $\text{mincard} = 600$. This shows that as the number of rows increases, the total number of colossal
closed itemsets also increases, thereby requiring more time to mine the dataset. DisClose still outperforms D-Miner when the mincard threshold approaches 2000.

Similar behaviour is also shown in Figure 5.8(c) where, with the increase in the number of rows, more time is required for DisClose to reach the smaller mincard threshold. In this case, the algorithm is able to reach mincard = 700. The difference in the execution time between DisClose and D-Miner is very small at high cardinality threshold.
However, when the mincard threshold is lowered, DisClose clearly outperforms D-Miner.

Figure 5.8: The effect of changing the number of rows on the runtime
CHAPTER 5. EXPERIMENTAL EVALUATION

This set of experiments also shows this trend: as the number of rows increases, the computation time for both algorithms increases. DisClose also scales well, as the execution time increases with the changes in the mincard threshold when increasing the number of rows in the dataset.

5.2.2.1 DISCLOSE VS. FP-CLOSE, CARPENTER AND D-MINER

Figure 5.9 compares DisClose with FP-Close, Carpenter and D-Miner on synthetic dataset T150L2000N4000. The percentage of density of values that exist for this dataset is 50%. Figure 5.9(a) shows that beginning with the largest closed itemsets, DisClose is able to discover colossal closed itemsets with the maximum support of 10. There are 227,674,613 colossal closed itemsets found when mincard = 600.

Figure 5.9(b) shows that as minsup decreases, the runtime of the three algorithms increases. Apparently, the algorithms can only reach minsup = 96, and Carpenter runs the fastest. It is observed that even at lower support threshold the existence in the number of large cardinality closed frequent itemsets is huge. D-Miner took the most time to discover the closed itemsets at this support threshold. There are 4,707,487 closed itemsets discovered at minsup = 97 with the largest cardinality of closed itemsets being equal to 85.

Figure 5.10 compares DisClose with FP-Close, Carpenter and D-Miner as the number of rows increases. With the same percentage density of true values of 50%, DisClose is able to discover 36,297,315 colossal closed itemsets when mincard = 600, having a maximum support of 8. This is shown in Figure 5.10(a).

As for the other three algorithms, Figure 5.10(b) shows that the algorithms can only reach minsup = 97, having a maximum cardinality of 68.
Figure 5.9: Comparison on T150L2000N4000 with *FP-Close*, *CARPENTER* and *D-Miner*
CHAPTER 5. EXPERIMENTAL EVALUATION

5.2.2.2 DISCLOSE VS. TTD-CLOSE

The testing of algorithm TTD-Close on both T150L200N4000 and T200L2000N4000 datasets has taken more than one day in order to obtain the initial result. Hence, the outcome of the experiments is not displayed. However, the results
CHAPTER 5. EXPERIMENTAL EVALUATION

Both figures show that DisClose is still able to mine the dense datasets. It is observed that the algorithm scales well, as the time taken to discover the colossal closed itemsets increases with the increase in the row size of the datasets.
5.2.3 Cardinality

In the previous groups of experiments described above, the average length of each dataset is set to 2000, which means that each dimension of each dataset has on average 2 K distinct values. To test the performance of the different cardinalities, two more datasets have been generated that correspond to 1.5 K distinct values and 2.5 K distinct values. The other parameters remain unchanged: that is to say, the number of dimension is 4 K and the number of tuples is 100.

Figure 5.13(a)-(c) compares DisClose and D-Miner using the mincard threshold on synthetic dataset T100L1500N4000. In this set of experiments, the results show that as the cardinality value of the datasets increases, the value of the mincard that DisClose could reach also increases. Intuitively, with respect to cardinality pruning, a lower mincard results in an increase in run time. DisClose also scales well, as the execution time increases with the changes in the dimensionality of the datasets. The results show the DisClose still performs better than D-Miner as the magnitude of the cardinality changes.

In Figure 5.13(a), the T100L1500N4000 dataset contains the percentage density of true values of 38%. Therefore, by reducing the average length of the itemset, DisClose is able to reach closed itemsets until mincard = 200. In addition, D-Miner could reach a much lower cardinality threshold of 1700 when the cardinality value decreases.

As for Figure 5.13(c), as the number of average length increases, the percentage density of true values also increases to 64%. Hence, DisClose is able to reach colossal closed itemsets at a much larger mincard threshold of 1100. As for D-Miner, the mincard threshold that the algorithm could reach also increases to 2600.

5.2.3.1 DisClose vs. FP-Close, Carpenter and D-Miner

Figures 5.14 and 5.15 together demonstrate the effect of changing L. Figure 5.14 compares DisClose with FP-Close, CARPENTER and D-Miner as the average length (cardinality) decreases.
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.13: The effect of changing cardinality on the runtime
CHAPTER 5. EXPERIMENTAL EVALUATION

As compared to Figure 5.2, DisClose is able to operate up to mincard = 200 having a maximum support of 9. This is shown in Figure 5.14(a). Figure 5.14(b) shows that for the three algorithms, all could reach a lower minsup threshold. Amongst the three algorithms, FP-Close performs the best with the value of minsup
= 86. The maximum cardinality of the closed itemsets discovered at this support threshold is 22. As for D-Miner, at \( \text{minsup} = 87 \), the maximum cardinality of the closed itemsets is also 22. CARPENTER could reach at \( \text{minsup} = 89 \) with the largest closed itemset having the maximum cardinality of 20. As the \( \text{minsup} \) threshold decreases, the runtime of the three algorithms also increases. This is because the dataset is very dense, and the number of closed frequent itemsets that require checking increases substantially. As an example, the difference in the closed itemsets discovered between \( \text{minsup} = 86 \) and \( \text{minsup} = 87 \) is almost 42 million.

Figure 5.15 shows the results obtained as the number of average length increases. It is observed that for the T100L2500N4000 dataset, the cardinality of the closed itemsets is very large. In Figure 5.15(a), D-Miner discovered colossal closed itemsets until \( \text{mincard} = 1100 \). At this \( \text{mincard} \) threshold, the maximum support value for colossal closed itemsets is 15.

Figure 5.15(b) shows that for the three algorithms, the lowest \( \text{minsup} \) value reached is 95 by D-Miner. This indicates that T100L2500N4000 is relatively denser than the T100L1500N4000 dataset. It is also observed that the maximum cardinality of the closed itemsets that exists at this \( \text{minsup} \) value is 481. CARPENTER and FP-Close can reach \( \text{minsup} = 96 \) with maximum cardinality = 477 and \( \text{minsup} = 97 \) with maximum cardinality = 472, respectively.

### 5.2.3.2 Disclose vs. TTD-Close

Figures 5.16 and 5.17 together demonstrate the effect of changing L. Figure 5.16 compares the results of DisClose and TTD-Close on the dataset T100L1500N4000. With the reduction in the average itemset length, Figure 5.16(a) shows that DisClose discovers closed itemsets at a lower \( \text{mincard} \) value of 1500. The maximum support value at this threshold is 6 which is the same support value for the dataset T100L2000N4000 shown in Figure 5.2.

Figure 5.16(b) shows that TTD-Close can reach \( \text{minsup} = 96 \) with a total of closed itemsets of 292,150. TTD-Close runs out of memory probably due to the
existence of larger cardinality itemsets at smaller \textit{minsup} thresholds. This shows that for dense dataset, even at high \textit{minsup} threshold, the size of the itemsets can be significantly large.

Figure 5.15: Comparison on T100L2500N4000 with \textit{FP-Close}, \textit{CARPENTER} and \textit{D-Miner}
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.16: Comparison on T100L1500N4000 with TTD-Close
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.17 compares the results of \textit{DisClose} and \textit{TTD-Close} as the average itemset length increases. Figure 5.17(a) shows that with the same maximum support value of 6, \textit{DisClose} is able to discover colossal closed itemsets with a \textit{mincard} value of 2200. Figure 5.17(b) shows that \textit{TTD-Close} can reach \textit{minsup} = 97 with a slight increases in processing time as compared to the result in Figure 5.2(b), before it runs out of memory.

![Comparison on T100L2500N4000 with TTD-Close](image)

Figure 5.17: Comparison on T100L2500N4000 with \textit{TTD-Close}
5.3 Discussions of Results from Synthetic Datasets

Based on the different aspects of the characteristics of the synthetic datasets used to test the algorithms, discussions of the results of the evaluation is as follows:

- **Dimensionality**
  It has been observed that by keeping the \( L \) value constant, increasing the number of dimension increases the sparseness of the datasets. Therefore, results show that as the number of dimension increases, \textit{DisClose} can reach to a much lower cardinality threshold. Using only the cardinality threshold, \textit{D-Miner} can only reach a \textit{mincard} value of 2100 for all the datasets. As \textit{D-Miner} was designed to discover closed itemsets using more than one constraint, it encounters the problem of having to mine only large cardinality itemsets. Comparatively, \textit{FP-Close}, \textit{CARPENTER} and \textit{D-Miner}, may reach a lower support threshold as the number of dimension increases. It is also observed that \textit{CARPENTER} performs better if large cardinality itemsets exist in the dataset. However, if the itemsets are shorter then \textit{FP-Close} performs best. In terms of the support threshold, it is shown that all three algorithms cannot reach closed itemsets at low support. This is an advantage of \textit{DisClose}, as it is able to discover the many colossal closed itemsets that exist at the lower end of the support spectrum. Comparing \textit{DisClose} and \textit{TTD-Close}, the way the algorithms identifies the itemsets in the datasets leads to the discovery of larger cardinality closed itemsets. The number of colossal closed itemsets discovered by \textit{DisClose} increases as the number of dimension increases. \textit{TTD-Close} can only reach a constant large \textit{minsup} value as the number of dimension increases. Further, \textit{TTD-Close} suffers from insufficient memory for all the datasets.

- **Number of rows**
  It is observed that as the number of row increases \textit{DisClose} spends more time discovering colossal closed itemsets at the same \textit{mincard} threshold. A
further increase will lead to \textit{DisClose} reaching a much higher cardinality threshold. Similarly to the results in changing the number of dimensions, as the number of row increases, \textit{D-Miner} can only reach a constant \textit{mincard} value of 2100. Further to the reason given above, \textit{D-Miner} was designed to discover closed itemsets with several values of thresholds; hence, increasing the number of rows does not affect its performance. For \textit{FP-Close}, \textit{CARPENTER} and \textit{D-Miner} the \textit{minsup} threshold increases as the number of rows increases. It can also be observed that larger cardinality closed itemsets exist at a lower support value, and hence \textit{CARPENTER} performs the best in this situation. In constrast, \textit{FP-Close} performs best when the cardinality of the closed itemsets that exist is small. However, none of the three algorithms are able to reach a low support threshold; this is achieved by \textit{DisClose}, as the results show that it is able to find colossal closed itemsets at low support thresholds. Increasing the row values results in a denser dataset for \textit{TTD-Close} hence it has been unable to provide a result due to insufficient memory. However, in this case \textit{DisClose} can discover colossal closed itemsets with \textit{mincard} = 1800 and \textit{mincard} = 1900 which have a maximum support of 5.

- \textbf{Cardinality}

  The increase in the average number of itemsets in the dataset results in an increased \textit{mincard} threshold for \textit{DisClose}. This also affects the impact of the increase of \textit{mincard} threshold for \textit{D-Miner}. In addition to the increase in the average number of itemsets, the occurrence of colossal closed itemsets also increases. This can be observed in the result, where, although the size of the closed itemsets is large, the maximum support threshold that can be discovered by \textit{DisClose} is also high. However, for \textit{FP-Close}, \textit{CARPENTER} and \textit{D-Miner}, as the average size of the itemset increases, the \textit{minsup} threshold also increases. It may be observed that due to the existence of large
cardinality closed itemsets, **CARPENTER** performs the best amongst the three algorithms. A similar outcome has also been obtained from **TTD-Close**.

### 5.4 Real Datasets

Four real high-dimensional (microarray) datasets have been selected for evaluation. Table 5.1 presents the four datasets in terms of their number of rows and columns. These datasets were obtained from the Orange website\(^3\), which provides open source data visualization and analysis for data mining (Demsar et al. 2004).

#### 5.4.1 Discretization

As all the real datasets contain continuous attribute values, itemsets that occur frequently in the data are essentially non-existent. The data must be discretized in order for meaningful (closed) itemsets to exist.

The discretization method applied to the selected real datasets is the *over-expressed cut-off threshold*. Given a user-defined parameter \( p \) and the maximum value of the row, \( \text{max} \), a value of row, \( v \), is over-expressed if

\[
v > \text{max} - \left( \frac{p}{100} \right) \text{max}
\]

The result gives a 1 in the boolean matrix if it satisfies the condition above.

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Instances (Rows)</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLBCL</td>
<td>77</td>
<td>7070</td>
</tr>
<tr>
<td>Leukemia</td>
<td>72</td>
<td>5147</td>
</tr>
<tr>
<td>Lung</td>
<td>203</td>
<td>12,600</td>
</tr>
<tr>
<td>MLL</td>
<td>72</td>
<td>12,533</td>
</tr>
</tbody>
</table>

---

\(^3\) [http://orange.biolab.si/datasets](http://orange.biolab.si/datasets). Last accessed: 26th April 2012
and a 0 otherwise. The application of the method is available from the same website that provides the implementation of D-Miner.

As stated at the beginning of the chapter, the implementation of the selected algorithms varies in terms of how these algorithms were designed to identify the items in the dataset. FP-Close, CARPENTER and D-Miner algorithms were designed to identify each item based on the value present for each attribute of the dataset. TTD-Close, however, corresponds to every single value that represents each attribute of the data.

Therefore, in the case of the real datasets selected, after applying the discretization method, the algorithm FP-Close, CARPENTER and D-Miner will identify all the items that are represented with a 1 value. For TTD-Close, all the items that are represented with 1 and 0 are accounted for during the mining process. DisClose is designed to satisfy both of these conditions in order for fair comparisons to be made.

### 5.4.2 DLBCL Data

Diffuse large B-cell lymphomas (DLBCL) and follicular lymphomas (FL) are two B-cell lineage malignancies that have very different clinical presentations, natural histories and response to therapy. However, FLs frequently evolve over time and acquire the morphologic and clinical features of DLBCLs and some subsets of DLBCLs have chromosomal translocations characteristics of FLs (Demsar et al., 2004).

This dataset contains 58 example of DLBCL and 19 examples of FL. After applying the discretization method, the percentage of density of true values (results that contain 1) is 72%. As can be seen in Figure 5.18, by applying the mincard threshold, the computation time and the count of closed itemsets are extremely highly-correlated. This suggests that DisClose runs in linear time relative to the number of closed itemsets discovered. There are 28,455,770 closed itemsets of
cardinality 900. DisClose also performs better than D-Miner which can only reach cardinality 1400 after 24 hours of running time.

5.4.2.1 **DisClose vs. FP-Close, Carpenter and D-Miner**

Figure 5.19(a) shows that the discovery of closed itemsets by DisClose reaches the minimum cardinality of 900 and has a maximum support of 14. This is a noteworthy ratio of approximately 18% of the 77 rows. The time required to reach this threshold was around two hours. At a further smaller mincard threshold, DisClose requires more than 12 hours to discover the closed itemsets.

Figure 5.19(b) shows the runtime of FP-Close, Carpenter and D-Miner, at different minsup values. As the minsup decreases, the runtime for D-Miner increases dramatically, while the runtime for both FP-Close and Carpenter remains relatively stable. This figure also indicates that FP-Close can reach the lowest minsup value of 58 with the maximum cardinality of 21. This is because the size of the itemsets that exist at this higher support threshold is relatively small. Hence, FP-Close performs better for this kind of situation. Both Carpenter and D-Miner, can reach minsup = 63. The maximum cardinality of the itemset that exists at this support threshold is 13. Although Carpenter is faster than D-Miner, it ran
out of memory after $\minsup = 63$. This is because CARPENTER still needs to search the rowset space in which the size of each rowset is less than $\minsup$, although it will not satisfy the $\minsup$. $D$-Miner took a much longer time in order to discover the closed itemsets after $\minsup = 63$. 

Figure 5.19: Comparison on DLBCL with $FP$-$Close$, CARPENTER and $D$-Miner
5.4.2.2 DisClose vs. TTD-Close

Figure 5.20 compares between DisClose and TTD-Close. As the method requires every single item in the dataset to be identified, DisClose needs to discover larger cardinality itemsets. Figure 5.20(a) shows that DisClose can reach closed itemsets.
with $\text{mincard} = 1700$ before it runs out of memory. There are a total of 224,128 colossal closed itemsets discovered with the maximum support of the itemset equal to 4. As for TTD-Close, Figure 5.20(b) shows that the algorithm can reach $\text{minsup} = 73$ before it runs out of memory.

5.4.3 Leukemia Data

This dataset is probably the most famous gene expression cancer dataset (Pomeroy et al., 2002), containing information on gene-expression in samples from human acute myeloid (AML) and acute lymphoblastic leukemias (ALL). The original research is one of the first to show a new approach to cancer classification based on gene expression monitoring by DNA microarrays.

There are 5147 genes (columns) and 72 samples (rows) from the originally measured 6817 probe sets after removing genes (columns) where at least one measurement was unavailable. The data was discretized using the over-expressed cut-off threshold, which results in the dataset containing the density of true values of 84%.

In Figure 5.21 it is noted that there are very few closed itemsets with large cardinality. Not surprisingly, in Figure 5.21, the time required by DisClose to find these colossal closed itemsets is very small. DisClose uses less than 10 minutes to find all closed itemsets with $\text{mincard} = 200$. However, for D-Miner this takes more
than 18 minutes to discover closed itemsets when $mincard = 4400$.

**5.4.3.1 DISCLOSE VS. FP-CLOSE, CARPENTER AND D-MINER**

Figure 5.22(a) shows that DisClose can reach $mincard = 200$ which contains closed itemsets with maximum support of 9. Figure 5.22(b) shows almost the same situation as Figure 5.19(b). What is different is that each algorithm can reach a much smaller $minsup$ value. As the size of itemsets that exists in this dataset is small, $FP$-$Close$ performs best.

![Diagram a](image)

![Diagram b](image)

Figure 5.22: Comparison on Leukemia with $FP$-$Close$, Carpenter and $D$-Miner
5.4.3.2 DISCLOSE VS. TTD-CLOSE

In comparison with TTD-Close, Figure 5.23(a) shows that DisClose is able to discover closed itemsets with mincard = 1300 which contains the largest support value of 5. As for TTD-Close, Figure 5.23(b) shows the algorithm can reach minsup = 65.

Figure 5.23: Comparison on Leukemia with TTD-Close
5.4.4 Lung Cancer Data

This dataset was originally built to distinguish between four different lung tumors (adenocarcinomas, small-cell lung carcinomas, squamous cell carcinomas, carcinoids) and normal lung tissue on the basis of DNA expression signatures (Pomeroy et al., 2002). There are 12,600 genes (columns) and 203 samples (rows). The data was discretized using the over-expressed cut-off threshold, which results in the dataset containing the density of the true values of 35%.

Figure 5.24 shows a precipitous increase in run time between the thresholds of 4000 and 4500 of mincard. This is because the differences in the numbers of colossal closed itemsets discovered between these thresholds amounts to 5878 closed itemsets. A total of 6932 closed itemsets were discovered by DisClose at mincard = 4000 before running out of memory. This is probably due to the large cardinality size of the itemsets discovered. As for D-Miner the algorithm took more than 400 seconds to discover the colossal closed itemsets at mincard = 6000.

Figure 5.24: Comparison of mincard thresholds on Lung with D-Miner
5.4.4.1 DisClose vs. FP-Close, Carpenter and D-Miner

Figure 5.25(a) shows that \( \text{mincard} = 4000 \), the maximum support value that exists in the colossal closed itemsets discovered by DisClose is 10. Figure 5.25(b) shows that Carpenter performs best amongst the three algorithms as the cardinality of the
itemsets that exist in this dataset is large. However, CARPENTER is able to reach only $\text{minsup} = 96$, due to memory running out. $FP$-Close cannot run to completion after $\text{minsup} = 96$. $D$-Miner is able to reach $\text{minsup} = 98$; however, the algorithm requires more than 19 hours to discover the closed itemsets.

### 5.4.4.2 DisClose vs. TTD-Close

For the lung dataset, it is not possible to obtain the runtime of $TTD$-Close as it runs out of memory for the initial support threshold value of 100. This shows that the lung dataset is both large and dense. $TTD$-Close needs to build a large $FR$-tree to cater for the huge number of existing closed frequent itemsets. However, $DisClose$ is able to reach $\text{mincard} = 7500$ with the closed itemset having the maximum support of 11. This is shown in Figure 5.26. There are a total of 10,858,767 closed itemsets discovered when $\text{mincard} = 7500$. This shows well that $DisClose$ is able to skip the huge numbers of small closed frequent itemsets and directly discovers colossal closed itemsets with sufficient support thresholds.

![Figure 5.26: Comparison on Lung with DisClose](image)
CHAPTER 5. EXPERIMENTAL EVALUATION

5.4.5 MLL Data

Mixed-lineage leukemias (MLL) are a subset of human acute lymphoblastic leukemias with a chromosomal translocation involving the mixed-lineage leukemia gene (Armstrong et al., 2002). This dataset was originally prepared to show that the differences in gene expression are robust enough to classify leukemias correctly as MLL, acute lymphoblastic leukemia (ALL) or acute myelogenous leukemia (AML).

There are 12,533 genes (columns) and 72 samples (rows). The data was discretized using the over-expressed cut-off threshold, which results in the dataset with the density of true values of 73%.

Using the mincard threshold, Figure 5.27 compares between DisClose and D-Miner. The figure shows that DisClose is able to reach mincard = 1400 with a total of 37,407,863 closed itemsets discovered. However, D-Miner can only manage to reach closed itemsets when mincard = 4100.

![Figure 5.27: Comparison of mincard thresholds on MLL with D-Miner](image)
CHAPTER 5. EXPERIMENTAL EVALUATION

5.4.5.1 DISCLOSE VS. FP-CLOSE, CARPENTER AND D-MINER

Figure 5.28(a) shows that when mincard = 1400, the maximum support value that exists in the colossal closed itemsets discovered by DisClose is 12. Figure 5.28(b) shows that CARPENTER performs the best amongst the three algorithms, although it
reaches only a $\text{minsup} = 92$. It has also been observed that when $\text{minsup} = 92$ and a run time of less than 300 seconds, large cardinality itemsets exist having the maximum cardinality of 461.

In the evaluation process, $\text{FP-Close}$ and $\text{D-Miner}$ can reach $\text{minsup} = 95$, however, both algorithms require more than 3 hours to discover the closed itemsets at lower support thresholds.

5.4.5.2 DISCLOSE VS. TTD-CLOSE

Figure 5.29 compares between $\text{DisClose}$ and $\text{TTD-Close}$. Figure 5.29(a) shows that $\text{DisClose}$ is able to reach at $\text{mincard} = 4100$ with the closed itemset having a maximum support of 9. There are a total of 84,198 closed itemsets discovered at $\text{mincard} = 4100$. Figure 5.29(b) shows that $\text{TDD-Close}$ is able to reach $\text{minsup} = 68$ with the total of closed itemsets discovered of 606,654.

5.4.6 Memory consumption

As memory consumption is an important characteristic of an algorithm, a number of datasets (Leukemia, MLL and Lung) were evaluated in order to assess memory cost. Memory usage is assessed in terms of changes in the number of rows and the number of columns. The largest amount of memory used by each algorithm on each real dataset is presented in Figure 5.30.

Figure 5.30(a) shows the memory consumption of $\text{DisClose}$ and $\text{D-Miner}$ when discovering colossal closed itemsets using the $\text{mincard}$ threshold. For MLL and Lung, $\text{D-Miner}$ uses the largest amount of main memory. This is because changes in both the number of columns and the number of rows affect $\text{D-Miner}$ as the algorithm searches for closed itemsets based on the size of columns and rows. For the Leukemia dataset, although $\text{D-Miner}$ consumes less memory than $\text{DisClose}$, the difference between the results is minimal. As the number of columns is larger in MLL dataset, in terms of memory usage $\text{DisClose}$ increases more than 2 times and for $\text{D-Miner}$ more than 10 times than compared with memory usage for the
Leukemia dataset. However, the increase in the amount of memory is substantial when the number of rows is larger in the Lung dataset, in terms of memory usage. DisClose uses more than three times the memory it uses for MLL. As for D-Miner, the amount of memory increases more than four times.
Figure 5.30: Memory usage comparison using the maximum mincard and minsup thresholds
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.30(b) shows the memory consumption of DisClose, FP-Close, CARPENTER and D-Miner. Except for DisClose which uses the mincard threshold, FP-Close, CARPENTER and D-Miner uses the minsup threshold. The amount of memory used by DisClose is nearly consistent with the changes in the threshold as it does not require the previously mined colossal closed itemsets to be retained in main memory. DisClose uses the least amount of memory amongst all of the algorithms. Note though that this is not really a fair comparison, as DisClose begins with large cardinality itemsets and other algorithms begin with small cardinality itemsets. This shows that other algorithms require a large amount of memory to process and store closed frequent itemsets from the highest support threshold.

It is also observed in Figure 5.30(b) that for FP-Close, the effects of changing the number of columns also affects memory consumption. The greater number of columns in the MLL dataset causes 13 times more memory consumption compared with that used for the Leukemia dataset. As for the row enumeration-based algorithm, CARPENTER, it consumes the largest amount of memory on the Lung dataset, as it contains the largest number of rows. D-Miner, however, uses less memory compared to FP-Close and CARPENTER. Nevertheless, the result shown represents the amount of memory used at the higher support threshold, as compared to FP-Close and CARPENTER, as D-Miner is unable to reach to a lower support value.

The same reasoning can be made for TTD-Close on the Leukemia data in Figure 5.30(c). Although the amount of memory is reasonably small as compared to DisClose, the reason is that TTD-Close can only reach a higher support threshold before running out of memory. However, for MLL data, TTD-Close is able to reach a much lower support value, and hence requires more memory. As for the Lung data, results cannot be compared between the two algorithms as TTD-Close is unable to mine the data. Similar to DisClose, TTD-Close also constantly changes memory usage, as the value of the support threshold changes, as TTD-Close does not need to maintain previously mined closed itemsets.
5.5 Discussion on Results on Real Datasets

Overall, the results of the experiments on real world high-dimensional datasets illustrate that DisClose can use the mincard constraint to prune the search space and discovers colossal closed itemsets with reasonable support thresholds. As for support-based algorithms, it has been demonstrated that each of the algorithms is unable to discover closed itemsets that exists at a lower support threshold.

The performance of support-based algorithms also varies with the dataset. It is observed that FP-Close performs the best when the cardinality of the closed itemsets existing in the dataset is small. This shows that FP-Close, being a column enumeration-based algorithm works well when the size of the itemsets is small. CARPENTER performs the best for large cardinality itemsets as its closed itemset mining is based on the number of rows. D-Miner shows that it can identify closed itemsets based on both the cardinality of the itemsets as well as on the support constraint. However, mining for closed itemsets using either one of these constraints does not well evidence the purpose of D-Miner as a constraint-based mining algorithm. Further, D-Miner performs the worst in applying both thresholds independently, as compared to other algorithms.

In terms of memory consumption, it is observed that for both DisClose and TTD-Close, changes in the memory usage are proportional to changes in the threshold values. It is observed that DisClose requires a small amount of memory to discover the colossal closed itemsets. As for FP-Close, the increase in the number of columns results in an increase in the memory consumption. This is because the algorithm is required to recursively build the FP-tree as well as stores multiple CFI-trees. Although CARPENTER is a row enumeration-based search algorithm, as the support threshold decreases, the algorithm is required to store a larger conditional transposed table that does not satisfy the constraint, hence, increasing the memory cost.
5.6 Summary

This chapter has reported on an experimental evaluation of the proposed algorithm, DisClose in comparison to selected state-of-the-art algorithms, namely FP-Close, D-Miner, CARPENTER and TTD-Close. The algorithms were compared on two types of dataset, synthetic and real. DisClose has demonstrated that it is able to discover colossal closed itemsets with a reasonable support threshold that other algorithms are unable to reach. The algorithm scales well with changes in the number of dimensions, the number of rows and the average cardinality of the itemset in the dataset using the mincard constraint. Further, DisClose consumes comparatively small amounts of memory in order to perform the search for colossal closed itemsets.
Chapter: 6

Conclusions and Future Work

This chapter presents a summary of the proposed colossal closed itemset mining algorithm - *DisClose*, the main findings and research contribution and possible future research direction.

Section 6.1 gives a summary of the proposed colossal closed itemset mining algorithm – *DisClose*.

Section 6.2 presents the main findings and contribution of the research work.

Finally, Section 6.3 presents some directions for future research.

6.1 Summary

It has been shown that the characteristics of high-dimensional datasets provide a significant challenge to itemset mining algorithms especially in the discovery of colossal closed itemsets. High-dimensional datasets, characterised by relatively large number of columns as compared to the number of rows, are also relatively dense. With these characteristics, the existing column enumeration-based algorithms suffer from an exponential increase in running time as the average row length increases. For example, if \( n \) is the maximum row size, there are \( 2^n \) potential frequent itemsets. A study undertaken by Pasquier et al. (1999), addresses this by reducing the number
of frequent itemsets by enumerating the closed frequent itemsets. Although closed itemsets produces a smaller set of results as compared to discovering frequent itemsets, mining based on the number of existing columns in high-dimensional datasets still suffers from increasing runtime as well as memory cost.

Another breed of algorithms was then proposed, which can overcome the drawback of a column enumeration-based search. These algorithms mine the high-dimensional datasets from the perspective of the row enumeration search space. The two main approaches that have been proposed are the bottom-up and top-down search of the row enumeration tree respectively. Row enumeration-based search has been shown to be efficient in mining closed itemsets in high-dimensional datasets. However, most existing algorithms rely on the minimum support threshold as the search constraint. This means that these algorithms discover closed itemsets beginning from the largest number of rows (i.e. the most frequent closed itemsets). Mining from the most frequent end of the support spectrum results in the discovery of many small cardinality closed itemsets. The existence of these small cardinality closed itemsets makes it difficult for the algorithms to reach the large cardinality itemsets due to insufficient memory space. In addition, as the algorithms move further down the support threshold, the performance in run-time of these algorithms decreases.

As high-dimensional datasets tend also to be dense, items in this dataset are more correlated and longer itemsets exist. Thus, discovery of these large cardinality (colossal) closed itemsets may provide interesting insights into the datasets. A study made by Zhu et al. (2007) attempted to discover colossal closed itemset by skipping through several levels of the column enumeration tree and approximating the number of colossal closed itemsets discovered. However, this method does not generally discover a complete set of colossal closed itemsets.

In order to attempt the discovery of the complete set of colossal closed itemsets, the study in this thesis has proposed that the search begins from the largest (colossal) itemset that exists in the dataset, and builds smaller itemsets.
CHAPTER 6. CONCLUSIONS AND FUTURE WORK

The two main stages in the process are to discover the colossal itemsets and then identify whether the colossal itemsets discovered are closed. Before entering these stages, a transposed version of the high-dimensional dataset was obtained. The implementation of the transposed table assists in reducing the complexity of the original dataset.

In the first stage of the mining process, two strategies were proposed in order to obtain the desired colossal itemsets. The first strategy was to find a method to search the dataset. The aim of this search is to begin the discovery from the largest cardinality (colossal) itemset that exists in the dataset. It is recognized that due to the characteristics of high-dimensional datasets, mining based on row enumeration reduces the number of searches as compared to the number of searches based on column enumeration. As the largest itemset usually occurs in a single transaction, the search follows the bottom-up approach of the row enumeration tree. The second strategy proposed was a refinement that introduces a constraint in the mining process in order to limit the search space. As the focus of the search is to identify the colossal itemsets, this study proposed the use of the minimum cardinality threshold, \textit{mincard}. Itemsets that do not satisfy the desired threshold size are discarded. The search space of the dataset can be safely pruned using the \textit{mincard} constraint, because of its anti-monotone property.

The second stage of the mining process is to identify whether the colossal itemsets discovered are closed. A novel closedness checking method has been proposed, based on the unique generator which identifies the closed itemset using only one of the rowset values. This is because for every closed itemset, there is only one unique rowset value in which this rowset value is a closed rowset. The advantage of the proposed closedness-checking method is that when the same colossal closed itemset is discovered in one of the rows, further checking with other rows is not required. Furthermore, identification of a closed itemset is performed during the search process. Hence, this removes the requirement for a separate phase to check and compare with previously discovered closed itemsets.
The methods proposed from the two main stages of the mining procedures were then implemented using the Compact-Row Tree (CR-Tree). The aim of CR-Tree is to provide a compact representation of the dataset in order to perform the mining process. CR-Tree helps to reduce the amount of memory consumed during the search process. This is evident in the evaluation where memory consumption for DisClose is 14 times less than TTD-Close in the MLL dataset. The size of the tree depends only on the number of rows in the dataset mined. Several pruning strategies were also proposed to further optimize the search. This helped to reduce the generation of the same candidate itemsets.

The procedures described above have been realized in the DisClose algorithm. The performance of DisClose was evaluated in comparison with selected state-of-the-art algorithms on both synthetic and real datasets. The results obtained show that DisClose is able to discover colossal closed itemsets with reasonable support threshold values that other support-based algorithms cannot discover. DisClose performs better than D-Miner, a constraint-based algorithm, in applying the mincard threshold to discover colossal closed itemsets. DisClose scales well with changes in the parameters of the datasets using the mincard constraint. The algorithm also proves that it consumes a small amount of memory during the search process. As an example, DisClose uses less than half of the memory consumed by D-Miner, 5 times less than FP-Close and 72 times less than the memory use by CARPENTER on Leukemia dataset.

6.2 Research Contributions

The approaches to the design of DisClose addresses the research question introduced in Chapter 1 and a number of associated research issues. This section returns to these research questions and issues and reflects upon the work carried out. In the following, the research contributions of this thesis are presented by discussing each
of the identified research issues in turn, as well as the manner in which the proposed research addresses each individual issue.

6.2.1 Search Strategy for Colossal Closed Itemsets

This section questions: ‘To bypass these small frequent itemsets, can the search begin with the largest cardinality itemsets?’

It can be seen that searching a high-dimensional dataset based on row enumeration reduces the amount of search space, as compared to a column enumeration-based approach, due to the characteristics of the dataset having relatively fewer rows compared to columns. Row-enumeration based search is divided into two approaches; bottom-up and top-down. The top-down search strategy is shown to perform better than bottom-up when using the support threshold as the search constraint. However, by applying the bottom-up search of the row enumeration tree results in the *direct extraction of large cardinality closed itemsets which in turns avoids the search for small cardinality closed itemsets in high-dimensional data*.

6.2.2 Implementation of Constraint Measure

This section questions: ‘To identify large cardinality itemsets, is it possible to identify a method that can acquire and utilize and alternative threshold other than the support threshold?’

It may be seen that using the *minsup* threshold discovers the itemsets by limiting the search space based on the number of occurrences. The existing number of small itemsets at the frequent end of the support spectrum tends to be huge; hence, discovering the large itemsets is a problem. As the search begins from the largest itemset size, smaller itemsets are built by applying an *alternative threshold based on the cardinality of itemsets*. 
6.2.3 Closedness-checking Method

This section questions: ‘To reduce the memory space and processing time, can the closed itemsets be identified during the search thus avoiding the necessity of having to check whether the closed itemsets have already been discovered?’

The result of investigating how to identify closed itemsets leads to the development of a closedness-checking method, suited to the proposed search strategy. Attempting to identify the closed itemsets during the search process without having to repeatedly check the closed itemsets already discovered leads to the idea of a unique generator. The implementation of the unique generator results in discovery of the closed itemset using only one of the rowset values and allows the algorithm to identify colossal closed itemset without repeated checking of the result set.

6.2.4 Data Structure – CR-Tree

This section questions: ‘Are there ways to avoid generating unnecessary candidate itemsets thus reducing memory space usage and computation time?’

Consideration of this led to the development of the Compact Row-Tree (CR-Tree) data structure. The CR-Tree provides a compact representation of the itemsets during the search process. It is observed that checking and storing redundant candidate itemsets leads to an increase in the amount of computation time as well as memory cost. With the aid of the several optimization strategies proposed during the search of the CR-Tree, the amount of memory used is reduced of memory space, as only potential itemsets are stored and those deemed redundant are removed. Eliminating unnecessary itemsets also facilitates the reduction of computation time, as a smaller number of intersections occur between the itemsets.
CHAPTER 6. CONCLUSIONS AND FUTURE WORK

6.2.5 Algorithm to Discover Colossal Closed Itemsets

This section questions: ‘Is there an efficient way to represent the results in order to compare and demonstrate the strengths and weaknesses of any proposed new method?’

Combining the individual steps discussed in the above sections leads to the DisClose algorithm. Experiments conducted on both synthetic and real datasets shows that DisClose is a suitable algorithm to enable discovery of the colossal closed itemsets that exists in high-dimensional datasets, based on a specified cardinality threshold. In addition, DisClose appears to be scalable to changes in the number of dimensions, the number of tuples and the cardinality of each dimension of the dataset as the value of the cardinality threshold changes.

6.3 Future Work

The work described in this thesis has sparked a number of promising directions for future research outlined below.

- Examining the colossal closed itemsets with known classes in the datasets. After discovering the colossal closed itemsets, a question arises which regards to whether there is a link between the colossal closed itemsets, their corresponding support sets, and the set of samples that they identify. Many of these datasets were originally constructed with the goal of running classifier algorithms on them. Do any of the closed itemsets actually identify the known classes of example in the datasets? Hence, methods to examine whether the colossal closed itemsets discovered matches with the known classes or provide new insights into the classification problem are therefore seen as an important avenue for future research.

    Studies can be made into which classifier algorithms are suited to deal with colossal closed itemsets. Some classifier algorithms use more than the confidence threshold in order to efficiently obtain the classification rules.
It is possible to add the \textit{mincard} threshold to these algorithms in order to obtain classification rules that are based on the size of the itemsets in the rule generation.

- **Combining \textit{DisClose} with other state-of-the-art algorithms.** It seems that \textit{DisClose} will perform poorly when using a low \textit{mincard} threshold on dense datasets. This may be observed in the experimental results obtained. As low \textit{mincard} values typically mean high support, existing algorithms may be better in traversing these regions. Top-down row enumeration algorithms such as \textit{TTD-Close} have shown good performance in discovering closed itemsets based on \textit{minsup} constraints on high-dimensional datasets. Thus, one avenue for future work would be to explore ways of blending \textit{DisClose} with previous \textit{minsup}-based algorithms to produce a superior overall algorithm for closed itemset mining on high-dimensional datasets.

The ability to discover the entire closed itemsets in the datasets provided does not mean that it provides the entire answer set, as some of the information may be redundant to the user. However, knowing that the algorithm is able to extract all the closed itemsets without having to face the difficulty of memory usage as well as run-time can be valuable. One option is to merge with the \textit{TTD-Close} algorithm. Since the algorithm searches the dataset from the largest rowset value, it has the advantage of discovering closed itemsets at the frequent end of the support spectrum. It is possible to hybridize the \textit{FR-Tree} structure proposed in \textit{TTD-Close} with the \textit{CR-Tree} proposed in order to achieve the discovery of closed itemsets in high-dimensional dataset. Since both \textit{DisClose} and \textit{TTD-Close} do not require access to already discovered closed itemsets, the memory requirement to store the entire closed itemsets will not be an issue.

- **Comparison with \textit{Pattern-Fusion} algorithm.** During the course of this work, the author was unable to obtain the \textit{Pattern-Fusion} program (either source code or executable). It would be interesting to compare \textit{DisClose} with
the Pattern-Fusion algorithm. This may be done in terms of both run-time and the closed itemsets that may be missed by Pattern-Fusion. The Pattern-Fusion algorithm’s aim is also to discover colossal closed itemsets. However, that algorithm searches the high-dimensional datasets by approximating the number of colossal closed itemsets discovered. It essentially skips several closed itemsets in order to reach the desired colossal ones.

Identification of the differences in the colossal closed itemsets discovered by Pattern-Fusion and DisClose could give insight as to whether the colossal closed itemsets missed by Pattern-Fusion are actually significant colossal closed itemsets or that those discovered by DisClose are redundant colossal closed itemsets. Another interesting approach is to combine both of the algorithms and see if the method applied in DisClose can help to fill missing colossal closed itemsets in Pattern-Fusion. A method can be proposed so that colossal closed itemsets that have been discovered by Pattern-Fusion will not be discovered by DisClose. This could lead to a decrease in run-time since the same colossal closed itemsets would not be discovered more than once.

- **Speeding-up the search process.** It may be possible to speed up the search, if DisClose were treated as a constraint-based mining algorithm and given two parameters: mincard and minsup, similar to D-Miner. D-Miner performs itemset mining based on the term ‘formal concept’. The current DisClose algorithm essentially begins from the region where itemsets have the support of one. It then searches the colossal closed itemsets following a bottom-up search of the row enumeration tree. Of interest here is what would be the effect of a higher minsup threshold, for example, minsup = 8; would DisClose find all support 4 itemsets and combine them to explore support 8 itemsets without generating size 5, 6, and 7- hence saving time?
• **Discovering the meaning behind discovered colossal closed itemsets.** In this thesis it has been observed that the number of colossal closed itemsets that may be discovered is huge. However, the question then arises as to what these colossal closed itemsets actually mean? Are all the colossal closed itemsets discovered of value? To answer these questions, further work could focus on finding the true meaning of these answer sets.

One approach may be to acquire an in-depth knowledge on the dataset in order to provide further understanding of the answer sets obtained. Another approach may be to obtain an already existing set of answers and to compare these with the answers discovered through the implemented algorithm.

Overall, this thesis has shown that large cardinality (colossal) closed itemsets can be directly extracted from high-dimensional datasets. The main hypothesis was that such itemsets could be derived efficiently by using a strategy that began the search from the largest itemset and progressively built smaller itemsets. We proved that this was indeed the case. The proposed algorithm *DisClose*, which consists of the closedness-checking method using the unique generator, has also shown that it can correctly and efficiently identify the closed itemsets. This is further assisted by use of the *CR-Tree* data structure.
Bibliography


International workshop on open source data mining: frequent pattern mining implementations, pages 77-86.


