Targeted Association Rule Mining in Data Cubes

A Project Report
Submitted in partial fulfilment of the
requirements for the Degree of
Master of Engineering
in
Computer Science and Engineering

by

Shrutendra K Harsola

Computer Science and Automation
Indian Institute of Science
BANGALORE – 560 012

June 2012

©Shrutendra K Harsola

June 2012

All rights reserved

TO

My Family

Acknowledgements

I am deeply grateful to Prof. Jayant Haritsa for his unmatched guidance, enthusiasm and supervision. He has always been a source of inspiration for me. I have been extremely lucky to work with him.

Also, I am thankful to Dr. Prasad Deshpande (IBM India Research Labs) for introducing me to this problem, and for all those long discussions and suggestions. It had been a great experience to work with him.

My sincere thanks goes to my fellow labmates for all the help and suggestions. Also I thank my friends who made my stay at IISc pleasant, and for all the fun we had together.

Finally, I am indebted with gratitude to my parents for their love and inspiration that no amount of thanks can suffice. This project would not have been possible without their constant support and motivation.

Abstract

Data mining techniques are usually applied over entire datasets. But the localized behavior of subset of data can be very different from that of aggregate behavior of dataset. Our goal is to mine patterns in these localized subsets of data. We looked at the problem of mining association rules in this framework. In this project, we deal with multidimensional market basket data i.e. in addition to set of customer purchase items, each transaction also has dimension attributes associated with it. Based on these dimension attributes, transactions can be visualized as distributed over cells of an n-dimensional cube. Our goal is to mine targeted association rules in this cube space. A targeted association rule is of the form $\{X \to Y\}_R$*, where* R *is a region in cube and* $X \to Y$ *is traditional association rule in region* R*.*

Firstly we explain two basic algorithms: RelaxedSup and TOARM developed on the lines of previous works. Then, we discuss the idea of bottom-up aggregation and cubing which are used to design CellUnion Algorithm. Then, the ideas of interleaving and credit based pruning are discussed. These ideas are incorporated to design efficient algorithm called IceCube Algorithm. We evaluated the performance of algorithms on both synthetic and real datasets. Experiments on these datasets show that the RelaxedSup and TOARM algorithms are the worst. CellUnion Algorithm is better than these two. IceCube Algorithm always provides best performance and the performance improvement is even more for large datasets and complex cubes.

Contents

Acknowledgements **i**

Abstract **ii**

Keywords **vii**

1 Introduction **1**
1.1 Organization of Thesis 2
1.2 Contributions 2

2 Problem Definition **4**
2.1 Notations 6

3 Approaches **8**
3.1 RelaxedSup Algorithm 9
3.2 TOARM Algorithm 12
3.3 CellUnion Algorithm 13
3.4 IceCube Algorithm 14
3.4.1 Credit Based Pruning 14
3.4.2 Interleaving Idea 15
3.4.3 IceCube Algorithm: Example 17
3.5 Implementation Details 20
3.5.1 Mechanism for Counting Candidates at Cell level 20
3.5.2 Generating Candidates: gen-cand($F_{k-1}(C), S_k(C)$) 21
3.5.3 Cubing Algorithm 21

4 Experimental Evaluation **25**
4.1 Synthetic Data Generator 25
4.1.1 IBM Quest Market Basket Synthetic data Generator 25
4.1.2 Multi-dimensional Market-Basket Synthetic data generator 26
4.2 Oracle (God's) Algorithm 27
4.3 Results: Synthetic Data 28
4.4 Results: Real Data 32

5 Related Work **37**

6 Conclusions **40**

References **41**

List of Tables

2.1 Notations 7

4.1 Parameters to IBM Quest Data Generator 25
4.2 Parameters to Multi-dimensional Data Generator 26
4.3 Targeted Patterns 33

List of Figures

2.1 Hierarchy on Location dimension 5
2.2 Hierarchy on Time dimension 5
2.3 Cube 6

3.1 RelaxedSup Algorithm 10
3.2 TOARM Algorithm 12
3.3 CellUnion Algorithm 13
3.4 IceCube Algorithm 17
3.5 GenCand 18
3.6 Survivor Sets 19
3.7 IceCube Example 19
3.8 Trie 20
3.9 Gen-cand: Join 22
3.10 Gen-cand: Prune 23
3.11 Lattice of views 24

4.1 Oracle Algorithm 28
4.2 Synthetic datasets 29
4.3 Execution Time 29
4.4 Time taken 30
4.5 Number of candidates counted 31
4.6 Peak Main Memory Utilization 32
4.7 Feasibility (Average Processing Time per Region) 33
4.8 Scalability 34
4.9 Number of Targeted Itemsets 34
4.10 Execution Time (DBLP) 35
4.11 Number of candidates counted (DBLP) 35
4.12 Peak Main Memory Utilization (DBLP) 36

Keywords

data cube; association rule mining; frequent pattern mining; localized rules; targeted rules

Chapter 1

Introduction

Data mining techniques involve analyzing data to extract interesting patterns (clustering, frequent pattern mining) or to build models representative of data (classification, regression). But most of the times the aggregate behavior may not faithfully represent the behavior of subspaces of data i.e. behavior of data in localized subspaces can be very different from that of aggregate behavior. For example, a company may spend different amount of money for advertisement campaigns on different mediums (television, internet, newspaper etc) based on their effectiveness. But the best advertisement medium depends on product category, geography, target consumer etc. Internet might be best medium in USA but not in India. Similarly, it may be best to promote beauty products on television but internet might be the best medium for promoting smartphones. We will look at the problem of association rule mining / frequent pattern mining in this framework.

The problem of mining association rules over market basket data was introduced in [2]. Market basket data contains a set of transactions, where each transaction is a set of items. Association rule mining is used to find a set of association rules of form $X \rightarrow Y$, where X and Y are disjoint set of items. For example, customers who buy shoes also buy socks: $shoes \rightarrow socks$. Support and confidence are used to determine the importance of rules. Support of a rule is defined as $p(X \cup Y)$. While confidence of a rule $X \rightarrow Y$ is defined as $p(Y/X)$. Goal of association rule mining is to find

all rules that satisfy user-given minimum support and minimum confidence threshold. Applications of association rule mining include customer behavior analysis, intrusion detection, bioinformatics, association classification etc.

Several algorithms have been proposed in literature for mining association rules [2] [3] [8] [14]. But all of these algorithms mine association rules over aggregate data. If association rule mining is done over all transactions, many rules that apply only to a subset of transactions may be missed out. Some rules that are valid only for a particular customer segment may not show up because they might not satisfy the support criteria at aggregate level (i.e. wrt to all transactions). For example, sale of winter apparels may be very high in cold regions like Jammu and Kashmir, but this pattern may be missed if mining is done over India. We want to extend traditional association rule mining to capture such targeted rules that apply to a specific customer segment. [1] [13] defines localized association rules as rules that apply to a subset of transactions. We want to adopt this notion of localized support.

1.1 Organization of Thesis

First step of association rule mining is to find frequent patterns of form $\{I\}$, where I is a set of items which satisfies support criteria. Generating rules from frequent pattern which satisfies confidence criteria, is straightforward as explained in [2]. So, we will just concentrate on mining frequent patterns. The report is organized as follows: Chapter 2 formally defines the problem. In Chapter 3, we explain basic algorithms: RelaxedSup and TOARM, followed by our new algorithms CellUnion and IceCube. All algorithms are experimentally evaluated in Chapter 4. Chapter 5 gives related work in this area, followed by conclusions in Chapter 6.

1.2 Contributions

In summary, our contributions are as follows:

- Defining the problem of mining targeted rules in data cubes.
- Designing a set of algorithms to compute targeted rules.
- Experimental evaluation of proposed algorithms on real and synthetic datasets showing that IceCube algorithm is the best practical algorithm among all.

Chapter 2

Problem Definition

Traditional frequent pattern mining ignores the additional information about transactions like at what time or location transaction was recorded, or information about customer who purchased the items in transactions. We call these additional attributes in a transaction as dimensions. For example, time and location are dimensions. Assume that our data has n dimensions. Then, each transaction is of the form $T = \{i_1, \ldots, i_k; d_1, \ldots, d_n\}$, where $i_1, \ldots, i_k$ are the items present in transaction and $d_1, \ldots, d_n$ are values of n dimensions. For example, {*pen, ink, pencil; Q3, Mysore*} is a transaction with 3 items: pen, ink and pencil, whereas Q3 and Mysore are values of Time and Location dimension. Parent child hierarchies are defined on each dimension. Hierarchy in each dimension is modeled as a tree. Sample hierarchies on Location and Time dimension are shown in Figures 2.1 and 2.2. Hierarchy on Location dimension implies that Bangalore and Mysore comes under Karnataka, while Karnataka and M.P. comes under India. In the hierarchy shown in Figure 2.1, all nodes have same fanout. But we do not require this condition. Our approaches will work even for hierarchies having different fanout for different nodes. Only assumption is that all leaf nodes should be at same level.

Transactions with dimension information can be visualized as distributed over cells of an n-dimensional cube. Transactions will fall in different cells of cube based on their dimension values as shown in Figure 2.3.

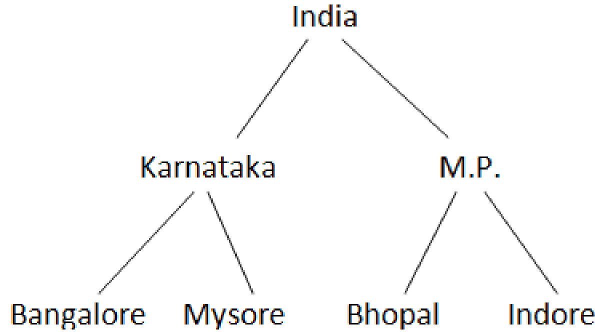


Figure 2.1: Hierarchy on Location dimension

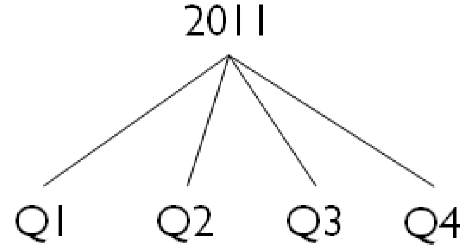


Figure 2.2: Hierarchy on Time dimension

Let $D_1, \ldots, D_n$ be domain of n dimensions. All nodes of a tree hierarchy form the domain of that dimension. For example, in above hierarchy, domain of Location = {India, Karnataka, M.P., Bangalore, Mysore, Bhopal, Indore}. Let D be the cartesian product of domains of all dimensions i.e. $D = D1 \times D2 \times \cdots \times Dn$. Then, any $C = (c_1, c_2, \ldots, c_n) \in D$ defines a cell in the cube, if $\forall i, c_i$ is a leaf in i^{th} dimension's hierarchy. And, any $R = (r_1, r_2, \ldots, r_n) \in D$ defines a region in the cube, if $\exists i$ s.t. r_i is not a leaf in i^{th} dimension's hierarchy. For example, $R_1 = (Q2, Karnataka)$ defines a region having two cells: $(Q2, Bangalore)$ and $(Q2, Mysore)$. All such regions will be convex. But all convex shapes in cube are not regions. For example, a convex shape having two cells $(Q1, Mysore)$ and $(Q1, Bhopal)$ is not a region according to our definition.

Let minsup be user-specified minimum support threshold as percentage. Our goal is to find out all the targeted frequent patterns which are of the form: $\{I\}_R$ where, R is a region in cube and I is a traditional frequent pattern in the subset of transactions that fall in region defined by R. Pattern I should satisfy minsup threshold with respect to subset to transactions falling in region defined by R. For example, if 10000 transactions fall in region defined by R and support is 10%, then an itemset will be frequent in R if it appears in atleast 1000 transactions out of 10000 transactions. Suppose there are 20000 transactions in region $(Q3, M.P.)$, support is 10% and items raincoat and umbrella occurs together in 2340 transactions. Then, an example of targeted frequent itemset is: $\{raincoat, umbrella\}_R$, where R = {Time: Q3, Location: M.P.}

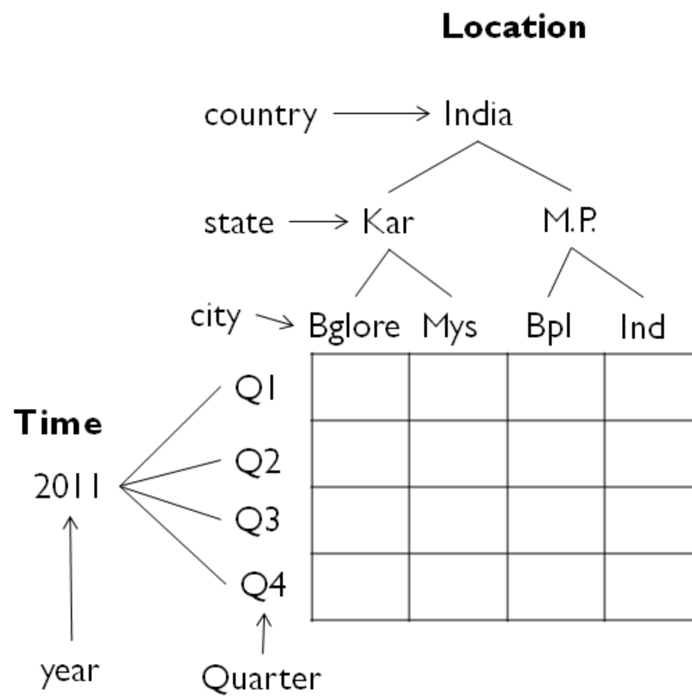


Figure 2.3: Cube

2.1 Notations

Table 2.1 gives the notations which will be used in later chapters for presenting the ideas and algorithms. Some of these notations will become clear in following chapters.

$minsup$	user specified minimum support
n	number of dimensions
$D_1, \ldots, D_n$	domain of n dimensions
D	Cartesian product of domains $D = D1 \times D2 \times \cdots \times Dn$
$C = (c_1, \ldots, c_n)$	defines a cell in cube, if $\forall i, c_i$ is a leaf in i^{th} dimension's hierarchy.
$R = (r_1, \ldots, r_n)$	defines a region in cube, if $\exists i$ s.t. r_i is not a leaf in i^{th} dimension's hierarchy.
$count(I, R)$	number of transactions in region R that contains itemset I
$nodes_i$	number of nodes in hierarchy tree of dimension i
$leaves_i$	number of leaves in hierarchy tree of dimension i
$levels_i$	number of levels in hierarchy tree of dimension i
N	No. of cells in cube $N = leaves_1 * \cdots * leaves_n$
$C_1, \ldots, C_N$	cells in cube
$ntrans(R)$	No. of transactions in region R
$total_trans$	Total no. of transactions in cube
$L_k(C_i)$	Local frequent itemsets of size k in cell C_i
$S_k(C_i)$	Survivor set of size k in cell C_i
$F_k(C_i)$	Foreign set of size k in cell C_i

Table 2.1: Notations

Chapter 3

Approaches

Firstly, we will explain the RelaxedSup Approach which flattens the dimension hierarchy and then, try to use the traditional association mining algorithm but with relaxed support. Next, we will explain TOARM algorithm which is an extension of algorithm proposed in [16] for our problem. Then, we will explain the CellUnion Algorithm based on ideas of bottom-up aggregation and cubing. Then, ideas of interleaving and credit based pruning will be explained which are incorporated to design the efficient IceCube Algorithm. All these approaches assume input and output formats as explained below:

Input:

- set of transactions of form
 $T = \{i_1, \ldots, i_k; d_1, \ldots, d_n\}$.
- hierarchy trees for all dimensions.
- user specified minimum support as percentage (*minsup*).
- local frequent itemsets in all cells $\{L(C_1), \ldots, L(C_N)\}$, where $L(C_i)$ is set of frequent itemsets in cell C_i.

Output:

- set of frequent itemsets of form $\{I\}_R$, where R defines a region in cube and I is a frequent itemset

3.1 RelaxedSup Algorithm

We have already discussed that when mining is done over entire data, itemsets which are frequent at some cell/region level are missed out because they do not satisfy support criteria over entire data. One intuitive way to solve this problem is to use relaxed support w.r.t. the region having minimum number of transactions. Let R_{min} be the region having minimum number of transactions and $total_trans$ be the total number of transactions in cube. Then, percentage relaxed support is defined as:

$$relaxed_support = \frac{ntrans(R_{min}) * minsup}{total_trans} \tag{3.1}$$

Running mining algorithm with this relaxed support will ensure that itemsets which are frequent in any region will show up in output. But there are two problem with this:

- For any itemset in output, we will not be able to tell the region(s) where it is frequent.
- Since number of transactions in different regions are widely different and we set relaxed support w.r.t. the smallest region. So, many spurious itemsets will get generated in bigger regions. These itemsets are frequent w.r.t. relaxed_support but are not frequent w.r.t. *minsup* and hence are not useful.

To solve these two problems, we extend each transaction by adding dimension attributes to it as items. Then, we apply mining on these extended transactions with relaxed support. Then, filtering is applied on output itemsets to remove spurious itemsets and to figure out region(s) where an itemset is frequent. Firstly, for any itemset in output, we figure out the region using dimension attributes which were added as items. Then, count of itemset is checked against the minimum support for that region to decide whether itemset is frequent in that region or not. Pseudocode of the algorithm is given in Figure 3.1.

RelaxedSup Algorithm :

1. *Extend transactions:* For each transaction T
   - Extend the transaction by adding dimension information to transaction
   - i.e. if T is in cell $C = (c_1, c_2, \ldots, c_n)$,
   - then for i = 1 to n,
     - add c_i and all ancestors of c_i to T
2. *Compute relaxed support:* Find the relaxed support w.r.t. to smallest region
   - R_{min} = Region having minimum no. of transactions
   - total_trans = total no. of transactions in cube
   - $relaxed_support = \frac{(ntrans(R_{min}) \times minsup)}{total_trans}$
3. *Do mining:* Run association mining algorithm on extended transaction file with relaxed_support.
   - Let F_{ext} be frequent itemsets generated
4. *Filter and generate output:* For each itemset $I \in F_{ext}$
   - Separate out basket items and dimension items from I.
     - Let I_b be part of I having basket items and I_d be part of I having dimension items.
   - If there is exactly one dimension item in I_d from each dimension, then
     - Find out region R from these dimension items.
     - if I is frequent in R wrt original support i.e. $count(I, R) \geq ntrans(R) * minsup$, then output the targeted frequent itemset $\{I_b\}_R$
   - Otherwise,
     - Drop I

Figure 3.1: RelaxedSup Algorithm

Analysis: Suppose a frequent itemset I = $\{i_1, \ldots, i_k$, Karnataka, Mysore, Q3$\}$ is produced in output, then this itemset I will simply be ignored in post-processing step because it contains 2 items from Location dimension. Itemset I is ignored to avoid same targeted frequent itemset being reported multiple times. Since I is frequent, its subsets $\{i_1, \ldots, i_k$, Karnataka, Q3$\}$ and $\{i_1, \ldots, i_k$, Mysore, Q3$\}$ will also be frequent and will produce the appropriate targeted frequent itemset, if any.

For association mining, we used the implementation of Apriori algorithm by Bart Goethals [12] which has been shown to be efficient. Then, we further improved the RelaxedSup Algorithm by changing the basic Apriori code. We applied following optimizations:

- An itemset in output having 2 or more dimension items from same dimension does not generate any targeted frequent itemset (It is simply discarded during filter step). So, we try to remove such itemsets as early as possible during mining process. During candidate generation step in Pass 2 of Apriori algorithm, we do not generate those candidates which have two dimension items from same dimension. So, no 2-itemset will show up in output having two dimension items from same dimension. Actually, it also ensures that no k-itemset ($k > 2$) will show up in output having two dimension items from same dimension because its 2-subset with those two same dimension items is not frequent.

- We do not need actual counts of itemsets having only dimension items. But we can't remove them because they will combine with other itemsets during next passes and generate frequent itemsets. So, during counting step in every pass, we do not explicitly count candidates having all dimension items. Instead we simply assume their counts to be 100%.

Even after these optimizations, this approach is highly inefficient due to following reasons:

- Relaxed support is usually very low and hence number of itemsets produced in

TOARM Algorithm:
For each region R in cube

- Let $C_1, C_2, \ldots, C_m$ be cells contained in region R
- *Compute union:* Compute union of local frequent itemsets from all cells contained in R.
  - $\mathcal{U}_R = L(C_1) \cup \cdots \cup L(C_m)$
- *Count:* Count all the itemsets of $\mathcal{U}_R$ over region R i.e. $\forall I \in \mathcal{U}_R$, compute $count(I, R)$.
- $\forall I \in \mathcal{U}_R$,
  - if $count(I, R) \geq ntrans(R) * minsup$, output targeted frequent itemset $\{I\}_R$

Figure 3.2: TOARM Algorithm

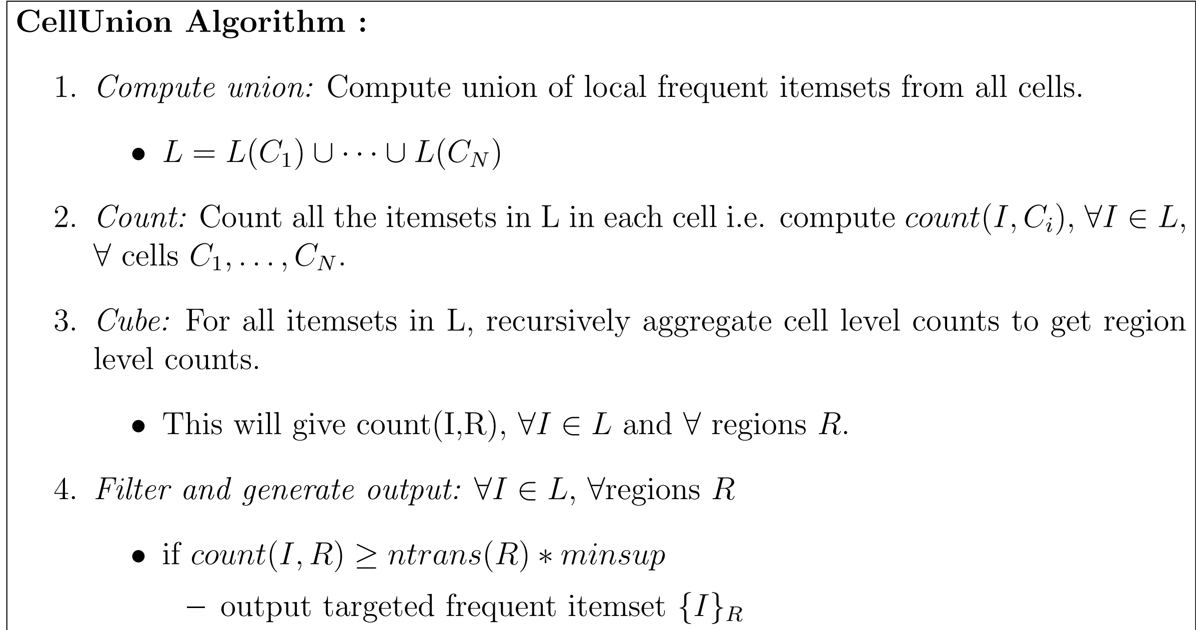
output is very large. So, mining algorithm takes lot of time. Though many of these itemsets will be discarded during filtering step.

- Extending transactions by adding dimension attributes increases the length of transactions which increases the time taken for counting candidate itemsets.

3.2 TOARM Algorithm

We can easily observe that an itemset can't be frequent in a region if it is not frequent in any of the cells contained in that region. TOARM approach [16] used this property for online association rule mining in data cubes. TOARM initially computes the local frequent itemsets in all cells. Then, given a region R, it computes the union, $\mathcal{U}_R$, of frequent itemsets of cells contained in region R. Then, it counts itemsets in $\mathcal{U}_R$ in region R. Frequent itemsets in region R are then generated by checking the counts against the support threshold in region R.

This algorithm can be extended for our problem. Our goal is to find frequent itemsets in all regions in cube. So, we will repeatedly call the TOARM algorithm for each region R in cube. The pseudocode of this approach is given in Figure 3.2.



CellUnion Algorithm :

1. *Compute union:* Compute union of local frequent itemsets from all cells.
   - $L = L(C_1) \cup \cdots \cup L(C_N)$
2. *Count:* Count all the itemsets in L in each cell i.e. compute $count(I, C_i)$, $\forall I \in L$, $\forall$ cells $C_1, \ldots, C_N$.
3. *Cube:* For all itemsets in L, recursively aggregate cell level counts to get region level counts.
   - This will give count(I,R), $\forall I \in L$ and $\forall$ regions R.
4. *Filter and generate output:* $\forall I \in L$, $\forall$regions R
   - if $count(I, R) \geq ntrans(R) * minsup$
     - output targeted frequent itemset $\{I\}_R$

Figure 3.3: CellUnion Algorithm

3.3 CellUnion Algorithm

An itemset I can't be frequent in a region R, if it is not frequent in any of the cells contained in region R. In other words, an itemset can be frequent in a region only if it is frequent in atleast one of the cells in that region. So, if we take union of local frequent itemsets from all cells (let's call this union L). Then, any itemset which is not element of this union L, can't be frequent in any region. And set of frequent itemsets in any region R will be a subset of this union L. So, for any region R, we need counts of all itemsets in union L to find out the itemsets which are frequent in region R. But it will be inefficient to do counting separately for each region. So, we will count itemsets in union L in all cells. Then, counts in any region can be computed by aggregating the counts from cells contained in region. Algorithm designed based on this idea is given in Figure 3.3.

Analysis: Counting of itemsets i.e. Step 2 is done efficiently using a Trie. Also, Step 3 is done efficiently using a dense cubing algorithm. Further details of Steps 2 and 3 are given in Sections 3.5.1 and 3.5.3. Only Step 2 is dependent on both number of transactions in dataset and number of frequent itemsets in each cell. All other steps are

only dependent on number of frequent itemsets in each cell. But with this approach, union size can become really big and hence counting time can get large. So, our next approach will focus on decreasing the number of itemsets to be counted in each cell.

3.4 IceCube Algorithm

The IceCube algorithm (Interleaved Credit Elimination Cube Mining Algorithm) is based on two ideas: credit based pruning and interleaving of counting and cubing. These ideas are explained below:

3.4.1 Credit Based Pruning

We will use the concept of credit as defined in [7]. For each cell, we have set of frequent itemsets and their counts. Let sup(C) be absolute support threshold for cell C i.e. sup(C) = minsup * ntrans(C). If an itemset I is not frequent in cell C, then $count(I, C) \leq sup(C) - 1$. We define credit of an itemset I in cell C as:

$$credit(I, C) = \begin{cases} count(I, C) - sup(C), & \text{if I is frequent in cell } C \\ -1, & \text{otherwise} \end{cases} \tag{3.2}$$

Thus, credit of an itemset in a cell C gives the extra count of itemset above the minimum support threshold in that cell. Hence, credit will be zero or positive for frequent itemsets in that cell and negative for non-frequent itemsets. Since we don't know the actual counts of non-frequent itemsets, we assume the maximum possible count which is $(sup(C) - 1)$. Hence, the credit for non-frequent itemsets is taken as -1. Credit of an itemset I in region R is sum of credit of I in all cells C contained in region R.

$$credit(I, R) = \sum_{C \in R} credit(I, C) \tag{3.3}$$

Similar to that for cell, credit of an itemset I in a region R will be non-negative, only if I is frequent in R. So, any itemset I can't be frequent in a region R, if credit(I,R) is negative. This insight can be used to avoid counting itemset I in some cells. We need to count itemset I in a cell C, only if there exists a region R enclosing C in which I can possibly be frequent. So, we don't need to count itemset I in cell C, if credit(I,R) is negative for all regions R containing C. Set of itemsets to be counted in each cell can be generated as follows:

1. *Compute union:* Compute union of local frequent itemsets from all cells.
   - $L = L(C_1) \cup \cdots \cup L(C_N)$
2. *Compute cell level credits:* Compute credit for all the itemsets in L in each cell i.e. compute $credit(I, C_i)$, $\forall I \in L$, $\forall$ cells $C_1, \ldots, C_N$.
3. *Cube to get region level credits:* Recursively aggregate cell level credits to get region level credits.
   - This will give credit(I,R), $\forall I \in L$ and $\forall$ regions R.
4. *Generate survivor sets:* $\forall I \in L$ and $\forall$ regions R, if $credit(I, R) \geq 0$, then
   - Add I to $S(C)$, $\forall$ cells C contained in region R, where $S(C)$ is survivor set for cell C.

So, instead of counting the whole union L in each cell, we just need to count their corresponding survivor sets in each cell. All survivor sets will always be subset of complete union L. So, by using credit based pruning we can reduce the number of candidates to be counted in each cell.

3.4.2 Interleaving Idea

CellUnion algorithm firstly calculated union of local frequent itemsets of all lengths. This union was the set of candidate itemsets which needs to be counted in each cell. Then,

we counted this whole set of candidates in one shot and then performed cubing for all of these itemsets.

IceCube algorithm interleaves the counting and cubing of candidate itemsets of different sizes. We will follow the pass by pass approach i.e. we will first generate candidates 1-items which needs to be counted. Then, we will count these candidate 1-items and cube the counts to generate frequent 1-items in all regions. Then, we will generate candidate 2-itemsets, count them and cube them to get frequent 2-itemsets in all regions. And so on. So, instead of doing counting and cubing at one shot, we have interleaved the counting and cubing steps. While generating candidate set to be counted, CellUnion algorithm only had the information about local frequent itemsets. So, size of candidate set generated by baseline approach is huge. But with pass by pass method of interleaving approach, in addition to local frequent itemsets, we also have information about k-frequent itemsets in all regions, before generating candidates of size (k+1). So, the number of candidates generated is less as compared to CellUnion algorithm.

Now will define the notion of foreign set of a cell. Foreign k-set of cell C_i, denoted by $F_k(C_i)$, includes k-itemsets which are frequent in some region enclosing cell C_i.

Lemma 1. *An itemset I of size k needs to be counted in cell C, only if*

1. *$I \in S_k(C)$ i.e. I is element of survivor set of cell C, and*
2. *all (k-1)-subsets of I are elements of $F_{k-1}(C)$*

Proof. Itemset I needs to be counted in cell C, if it can possibly be frequent in a region R which contains cell C. Condition 1 is straightforward. As shown in Section 3.4.1, if $I \notin S_k(C_i)$, then it can't be frequent in any region R containing cell C. So, we don't need to count I in cell C.

We will prove the converse of condition 2. Assume Z is a (k-1)-subset of I and $Z \notin F_{k-1}(C)$. Then, by definition of foreign set, Z is not frequent in any region R containing cell C. So, for any region R containing C, I can't be frequent in R because one of its subset Z is not frequent in R. So, we don't need to count I in cell C. Conversely, I needs to be counted in cell C, if all (k-1)-subsets of I are elements of $F_{k-1}(C)$. □

IceCube Algorithm :

1. Generate survivor sets $\{S_1(C_1), \ldots S_1(C_N),\ S_2(C_1), \ldots S_2(C_N), \ldots\ldots\}$ for each cell using credit based pruning (as explained in Section 3.4.1).
2. Pass k is as follows:
   - For each cell C_i
     - *If* $(k == 1)$, *then*
       $cand_k(C_i) = S_k(C_i)$
       else
       $cand_k(C_i) = \text{gen-cand}(F_{k-1}(C_i), S_k(C_i))$
     - Count all itemsets in $cand_k(C_i)$ in cell C_i
   - *Cube:* Recursively aggregate cell level counts of itemsets to get counts at various regions.
   - if $count(I, R) \geq ntrans(R) * minsup$, then
     - Output targeted frequent itemset $\{I\}_R$
     - Add I to foreign sets, $F_k(C)$ of all cells C contained in region R.

Figure 3.4: IceCube Algorithm

Pseudocode of IceCube algorithm is given in Figure 3.4. In each pass k, for each cell C, we use gen-cand function to generate candidates of size k to be counted in that cell C. Gen-cand function uses Lemma 1 and Apriori property [3] used in Apriori-gen function in Apriori algorithm. Pseudocode of gencand function is given in Figure 3.5.

3.4.3 IceCube Algorithm: Example

Assume a simple cube having 4 cells as shown in Figure 3.6. It has 5 regions: R_1 containing C_1, C_2, R_2 containing C_3, C_4, R_3 containing C_1, C_3, R_4 containing C_2, C_4, R_5 containing C_1, C_2, C_3, C_4, Let each cell has 100 transactions and support is 4%.

Local frequent 2-itemsets with their counts are given in Figure 3.6(a). Credits for these itemsets will be $credit(I_1I_2, C_1) = 4$, $credit(I_1I_3, C_2) = 3$ and $credit(I_2I_3, C_3) = 0$. All other cell level credits are -1. Aggregating them will give following non-negative region level credits: $credit(I_1I_2, R_1) = 3$, $credit(I_1I_2, R_3) = 3$, $credit(I_1I_2, R_5) = 1$, $credit(I_1I_3, R_1) = 2$, $credit(I_1I_3, R_4) = 2$, $credit(I_1I_3, R_5) = 0$. All other region level

GenCand($F_{k-1}(C), S_k(C)$) :

1. **JOIN:** This step will generate candidates of size k by joining $F_{k-1}(C)$ with itself. Also, the generated candidate should be element of $S_k(C)$.

   insert into $cand_k(C)$
   (**select** $p.items_1, \ldots, p.item_{k-1}, q.item_{k-1}$
   from $F_{k-1}(C)$ p,$F_{k-1}(C)$ q
   where $p.items_1 = q.item_1$
   ......
   and $p.item_{k-2} = q.item_{k-2}$
   and $p.item_{k-1} < q.item_{k-1}$
   intersect
   select * from $S_k(C)$)

2. **PRUNE:** This step will remove those candidates for which some of its (k-1)-subsets are not elements of $F_{k-1}(C)$

   forall itemsets $I \in cand_k(C)$ **do**
   forall (k-1)-subsets s of I **do**
   if ($s \notin F_{k-1}(C)$) **then**
   delete I from $cand_k(C)$

Figure 3.5: GenCand

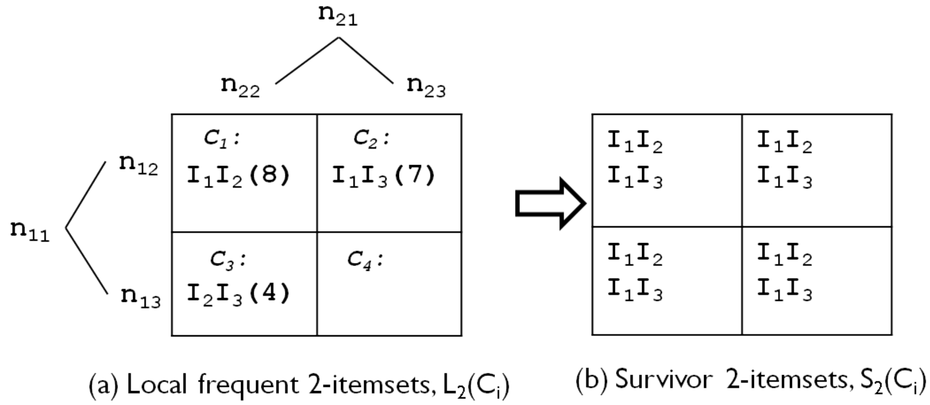


Figure 3.6: Survivor Sets

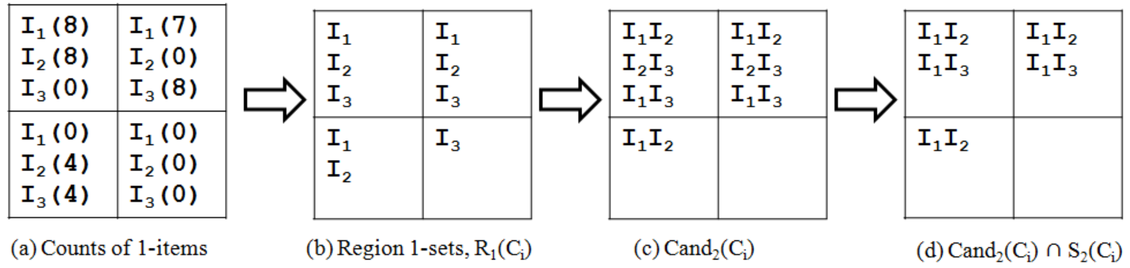


Figure 3.7: IceCube Example

credits are -1. Mapping them back to cells generate survivor sets as shown in Figure 3.6(b).

Cell level counts of 1-items are given in Figure 3.7(a). Aggregating them will generate following region level frequent 1-items: $R_1 = \{I_1(15), I_2(8), I_3(8)\}$, $R_2 = \{\}$, $R_3 = \{I_1(8), I_2(12)\}$, $R_4 = \{I_3(8)\}$, $R_5 = \{\}$. Mapping these frequent 1-items to cells generates region 1-sets as shown on Figure 3.7(b). 2-candidates that gets generated are shown in Figure 3.7(c). Intersecting these candidates with survivor sets gives final set of candidates counted by IceCube Algorithm, as shown in Figure 3.7(d). Overall, CellUnion would count a total of 3*4=12 itemsets whereas IceCube Algorithm counts only 5 itemsets.

3.5 Implementation Details

3.5.1 Mechanism for Counting Candidates at Cell level

We use trie which is a main memory data structure used for counting candidates and has been shown to have better performance as compared to other traditional structures used for counting in association rule mining algorithms [5] [4]. Trie is a rooted directed tree. Root is at depth 0 and each node at depth d point to zero or more children nodes at depth (d+1). Each edge in the trie represents an item. Children of a node are kept in lexicographical order of items they represent. Each node N of trie represent an itemset which is the concatenation of items in the path from root to node N. A counter is kept at each leaf node which stores the count of itemset represented by the leaf.

Initially we are given a set of candidates to be counted. Firstly, each candidate in the set is sorted in lexicographical order of items. Then, a trie is created from the set of candidates to be counted. An example trie for set of candidates ACD, ACF, ADF, AFG, CFG is shown in Figure 3.8. In this trie, node 7 represents the itemset ACD, node 8 represents itemset ACF and so on.

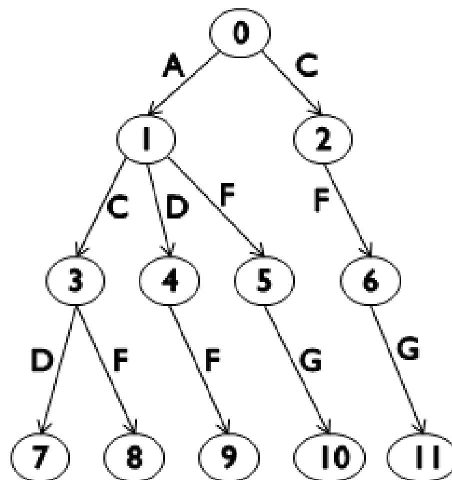


Figure 3.8: Trie

For counting candidates of size k, we will first create a trie with counters at leaf nodes. Then, we will take transactions one by one. For each transaction, we will traverse the trie

searching for each ordered k-subset of transaction. For every match, we will increment the counter at corresponding leaf node.

In case of CellUnion algorithm, where candidates of all sizes are counted in one shot, we will keep a counter at each node of trie. Then, while traversing trie we will increment counters for each node that is visited during traversal.

3.5.2 Generating Candidates: gen-cand($F_{k-1}(C), S_k(C)$)

Use of trie, simplifies the process of generating candidates in IceCube algorithm. For join step of gen-cand function, firstly we will construct a trie using itemsets in foreign set $F_{k-1}(C)$. Then, for each leaf node N in trie, we will take all its right sibling edges and add them as children of N. Then, we will remove those candidates from trie which are not present in $S_K(C)$. This completes the join step of candidate generation. Assume for some cell C in pass k=3, foreign set $F_2(C) = \{AC, AD, AF, CD, CF\}$ and survivor set $S_3(C) = \{ACD, ADF\}$. Then, join step is shown in Figure 3.9. Firstly, trie is constructed having itemsets from $F_2(C)$. Then, candidates are generated. For example, nodes 4 and 5 (which represent items D and F) are right siblings of node 3. So, two new nodes are added as children of node 3 which represent items D and F. Similarly, children for other nodes are created. Then, node 9 and 11 are deleted because they represent itemsets ACF and CDF, which are not elements of survivor set $S_3(C)$.

In prune step, for each k-candidate in trie, we will remove those candidates for which some of its (k-1)-subsets are not present in trie. Then, we will recursively remove those leaf nodes which are at depth less than k. Prune step is shown in Figure 3.10. Node 10 (which represent itemset ADF) is deleted because one of its subset DF is not present in trie. Then, other leaf nodes are deleted recursively which are at depth less than 3.

3.5.3 Cubing Algorithm

Given cell level counts of itemsets, cubing algorithm efficiently computes region level counts. One simple approach is that for each region, aggregate counts from its constituent

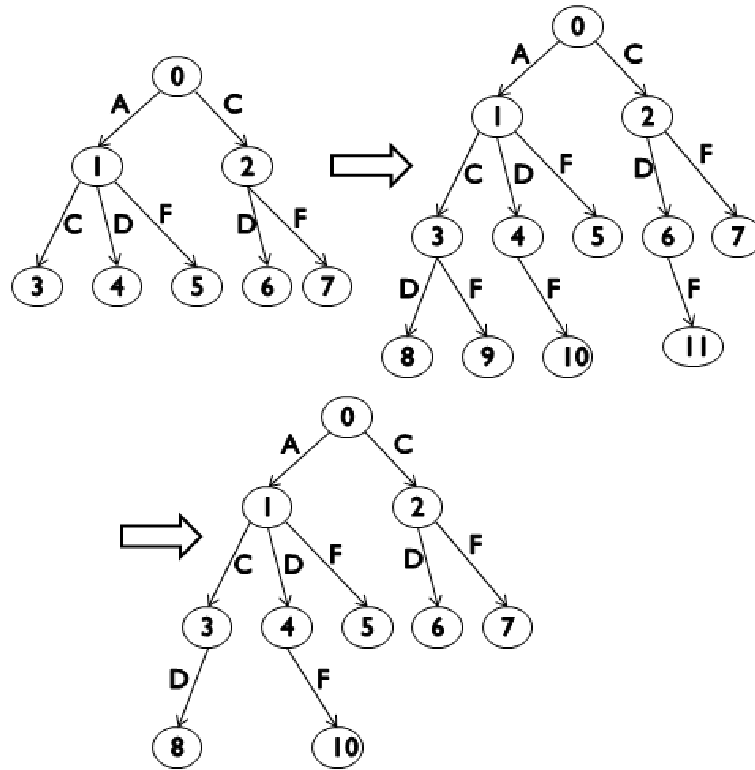


Figure 3.9: Gen-cand: Join

cells. But this approach is inefficient because it does not utilize the fact that counts for bigger regions can be computed from its constituent smaller regions instead of always computing from cell level.

Efficient cubing algorithms have been proposed in [7] and [15]. We will adapt these ideas to design an efficient cubing algorithm suited for our purposes. We will firstly define the notion of view and view lattice as defined in [9]. $V = (v_1, v_2, \ldots, v_n)$ defines a view in cube, where v_i represents a level in i^{th} dimension's hierarchy. Total number of views will be, $N_v = levels_1 * levels_2 * \cdots * levels_n$, where $level_i$ is number of levels in i^{th} dimension's hierarchy. A view $V' = (v'_1, v'_2, \ldots, v'_n)$ is called a child of view $V = (v_1, v_2, \ldots, v_n)$, if $\exists k$ s.t. $v_i = v'_i, \forall i \neq k$, and v'_k is child of v_k in k^{th} dimension's hierarchy. It implies that view V' can be computed using view V. This parent-child relationship results in a lattice of views as shown in Figure 3.11. (quarter,city) is the bottom view or root view or cell level view in this lattice. Some of the views have multiple parents. We will

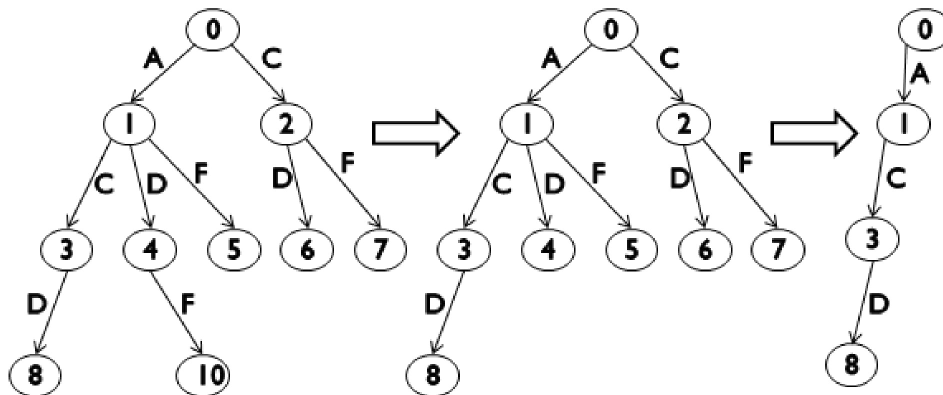


Figure 3.10: Gen-cand: Prune

convert this lattice into a tree by ensuring that each view has a single parent view. If a view has multiple parents, we will compute it from the parent view having minimum number of regions. All other edges are removed. For example, $(year, state)$ can be computed from either $(quarter, state)$ which has 8 regions or from $(year, city)$ which has 4 regions. Since $(year, city)$ has lesser number of regions, we will remove other edge i.e. between $(quarter, state)$ and $(year, state)$. Tree edges are shown bold in Figure 3.11. This pruning ensures that we always calculate a view from a parent which requires minimum amount of aggregation (additions). So, calculation of any view is optimal in terms of number of aggregation operations required. This implies that our approach is optimal in terms of number of aggregation (addition) operations required to compute all views.

After converting the lattice to a tree, for an itemset we will read the cell level counts in bottom view. Then, we will traverse the tree in depth first manner (DFS) and calculate each view from its parent view. Memory allocated to a view can be freed once all its children views have been computed. That means, atmost we have to keep views in an entire path from root view view to leaf view in memory. So, if we assume that fanout of any node in dimension hierarchy is atleast 2, then number of regions in any view is atmost half of number of regions in its parent view. This ensures that maximum memory required by our algorithm is twice the size of the bottom view.

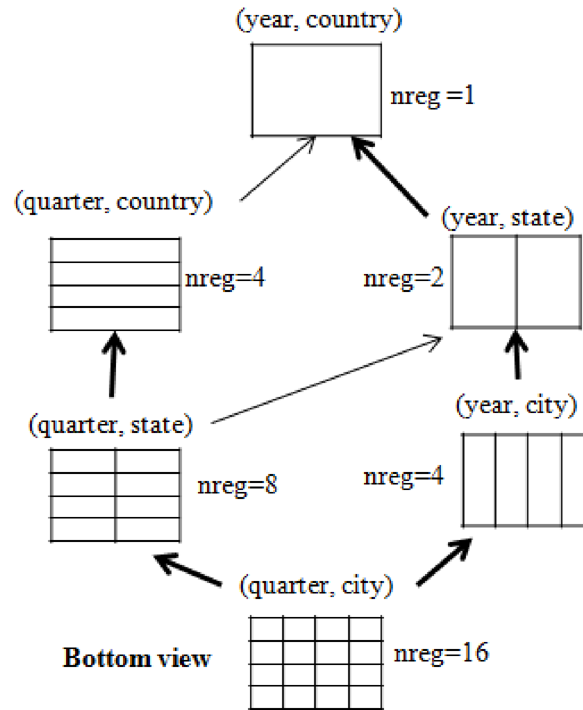


Figure 3.11: Lattice of views

But usually counts of itemset in many cells might be zero. So, we might be unnecessarily aggregating the zeros. Also, available memory may be far greater than that required for cubing a single itemset. So, instead of cubing a single itemset at a time, we will read the counts of multiple itemsets at a time in bottom view. For each cell, we will maintain a list of itemsets and their counts, sorted according to itemsets. Itemsets with zero counts are just ignored and not kept in this list. Now, calculating a view from parent view requires merging these sorted lists s.t. for same itemsets in 2 list, counts are aggregated. With this approach, maximum main memory required is 2 * (size of root view) * (number of distinct itemsets read in root view). Based on available main memory, we can decide the number of itemsets to be read in bottom view at a time.

Chapter 4

Experimental Evaluation

4.1 Synthetic Data Generator

IBM Quest Market Basket Synthetic Data Generator was developed by Agrawal et al. [3] for generating synthetic market basket datasets. But it generates transactions without dimension attributes. So, we used this Quest data generator code [11] as base to develop a multidimensional market basket synthetic data generator.

4.1.1 IBM Quest Market Basket Synthetic data Generator

IBM Quest data generator takes the parameters as given in Table 4.1. It firstly generates a set L_q of large (frequent) itemsets having $|L_q|$ itemsets. Average count of itemsets in L_q is inversely proportional to $|L_q|$. Size of each itemset in L_q is picked from a Poisson distribution with mean equal to $|I_q|$.

Then, transactions are generated using this set L_q. Size of each transaction is picked

$\lvert D_q \rvert$	number of transactions
$\lvert T_q \rvert$	avg. size of transactions
$\lvert I_q \rvert$	avg. size of maximal potentially large itemsets
$\lvert L_q \rvert$	number of maximal potentially large itemsets
N_q	number of items

Table 4.1: Parameters to IBM Quest Data Generator

Parameter	Description
$\|D\|$	number of transactions
$\|T\|$	avg. size of transactions
$\|I\|$	avg. size of maximal potentially large itemsets
$\|L\|$	number of maximal potentially large itemsets
$items_cell$	number of unique items to add to each cell
num_reg	number of random regions to consider
$items_reg$	number of items to add to each random regions
n	number of dimensions in cube
$l_1, \ldots, l_n$	number of levels in hierarchy of each dimension
$f_{i,1}, \ldots f_{i,l_i-1}$	fanout at each level of all dimensions $i = 1 \ldots n$

Table 4.2: Parameters to Multi-dimensional Data Generator

from a Poisson distribution with mean equal to $|T_q|$. Then, large itemsets from L_q are added to transaction until its size is exceeded.

4.1.2 Multi-dimensional Market-Basket Synthetic data generator

We developed Multi-dimensional Market-Basket Data generator. It takes the parameters as given in Table 4.2. Synthetic data generation has 2 phases:

- **Hierarchy Generation:**

  Hierarchy in each dimension is modeled as a tree. Following algorithm is used to generate tree hierarchy for dimensions:

  Algorithm:

  For each dimension $i = 1 \ldots n$

      For each level $j = 1 \ldots (l_i - 1)$

          Generate children for each node at level j of ith dim (No. of children of a node at level j in dimension i is equal to fanout of that level $f_{i,j}$)

- **Data Generation:**

  To model the real life behavior we need to maintain the uniqueness of each cell. For example, customer purchasing behavior is different in M.P. and Karnataka or

in summer and winter. Also, there must be some similarity between related cells. For example, customer purchasing behavior in Bangalore and Mysore have some similarities because both of them are in Karnataka. Our synthetic data generator should take care of these behaviors.

We first generate set of items for each cell. Let $B_1, \ldots, B_N$ be set of basket items generated for each of N cells. Set of items (B_i) will then be used to generate transactions for cell C_i. To model real life scenario each set B_i should have some unique items and sets that belong to related cells should have some common items.

Algorithm:

For $i = 1 \ldots N$

Add *items_cell* unique items to set B_i

For $j = 1 \ldots num_reg$

Generate a set P of *items_reg* unique items

Pick a random region R from cube.

For $k = 1 \ldots N$

If cell C_k comes under region R, then

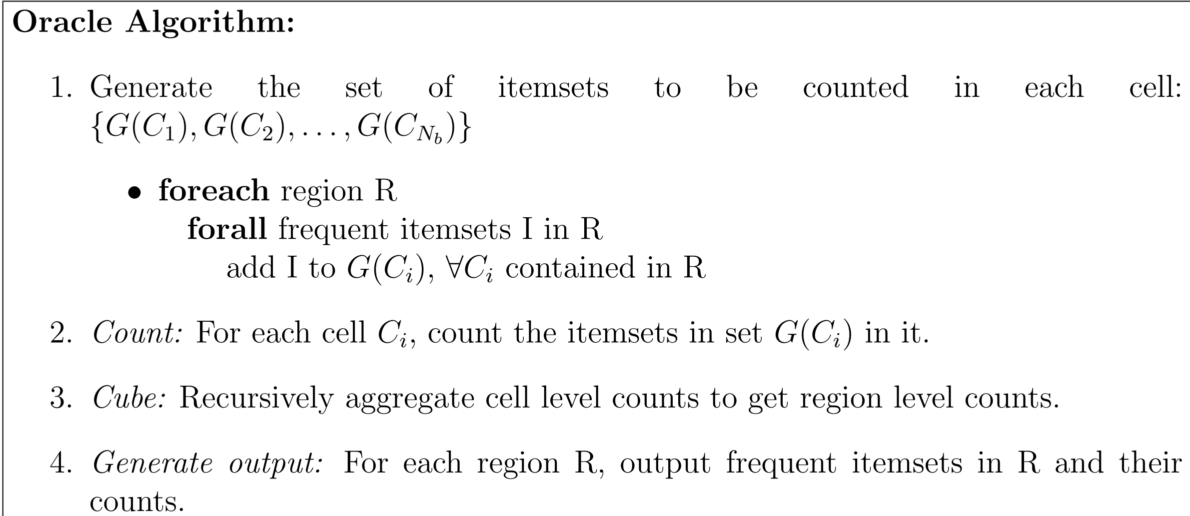
$B_k = B_k \cup P$

For $i = 1 \ldots N$

Call IBM Quest data generator to generate transactions for cell C_i using items in set B_i with following parameters $|T_q| = |T|, |I_q| = |I|, |L_q| = |L|, N_q = |B_i|, |D_q| =$ value picked from Poisson distribution with mean $|D|/N_b$

4.2 Oracle (God's) Algorithm

V. Pudi et. al. [14] proposed the idea of designing the Oracle Algorithm (God's Algorithm), as a method to get a lower bound on the performance. We borrowed this idea and designed an Oracle Algorithm for our problem. We assume that Oracle Algorithm



Oracle Algorithm:

1. Generate the set of itemsets to be counted in each cell: $\{G(C_1), G(C_2), \ldots, G(C_{N_b})\}$
   - **foreach** region R
     forall frequent itemsets I in R
     add I to $G(C_i)$, $\forall C_i$ contained in R
2. *Count:* For each cell C_i, count the itemsets in set $G(C_i)$ in it.
3. *Cube:* Recursively aggregate cell level counts to get region level counts.
4. *Generate output:* For each region R, output frequent itemsets in R and their counts.

Figure 4.1: Oracle Algorithm

knows the identities of frequent itemsets in all regions. It just need to gather the counts of these itemsets (which are known to be frequent) at cell level and aggregate cell level counts to get region level counts. This is an impractical algorithm and it is clear that any practical algorithm will have to do atleast this much amount of work. So, this gives us an idea of maximum opportunity available for performance improvement. Pseudocode of Oracle algorithm is given in Figure 4.1.

4.3 Results: Synthetic Data

We used the multi-dimensional market basket synthetic data generator to generate 5 synthetic datasets with varying number of dimensions, hierarchies and fanouts. Characteristics of these 5 datasets are shown in Figure 4.2. We used $support = 0.2 * density$ for our experiments. All algorithms were implemented in C++. All experiments were conducted on a Sun Ultra 24 quad core machine with 8 GB of RAM, running on the Ubuntu 10 operating system.

Figure 4.3 shows that the time taken by RelaxedSup algorithm for datasets D1 and D2 is 5-7 times more than TOARM algorithm and an order of magnitude or two more than other algorithms. Also, RelaxedSup algorithm takes even greater time as number

Data	No. of Trans	Avg. trans length	No. of Dim	Levels in dim hierarchy	Fanout	No. of cells	No. of regions	Density
D1	10M	8	2	3	3	81	87	0.013
D2	10M	8	2	3	4	256	185	0.023
D3	10M	8	2	3	5	625	336	0.039
D4	10M	8	2	4	3	729	871	0.052
D5	10M	8	3	3	3	729	1468	0.038

Figure 4.2: Synthetic datasets

of regions increases. Hence, we could give execution times of RelaxedSup only for D1 and D2 as it did not complete for other datasets in reasonable time frame. So, we will not show results for RelaxedSup algorithm in any of the graphs from now on.

Dataset	RelaxedSup (sec)	TOARM (sec)	CellUnion (sec)	IceCube (sec)	Oracle (sec)
D1	10921	2160	830	356	19
D2	30100	4095	1908	301	22

Figure 4.3: Execution Time

Figure 4.4 shows the execution time of various algorithms on log-scale for the five synthetic datasets. We can clearly see that TOARM algorithm is highly inefficient. For each region in cube, TOARM algorithm separately computes itemsets to be counted and then counts them. This lead to many itemsets being counted redundantly thus making it inefficient.

CellUnion algorithm is much better than TOARM algorithm because it removes the inefficiency due to redundant counting of itemsets. But still it computes union over whole cube and then count the itemsets in union in all cells. For complex cubes, this union can become huge. Hence, it takes large time for complex cubes.

IceCube algorithm is always better than TOARM and CellUnion algorithms. The performance gap increases with increase in cube complexity. Performance of IceCube

algorithm is an order of magnitude better than CellUnion algorithm for datasets D3, D4 and D5. But there is still a significant gap between IceCube algorithm and Oracle algorithm. So, theoretically there remains scope for improvement over IceCube algorithm. But Oracle is an impractical algorithm and given the exponential space of data cube, performance comparable to Oracle algorithm is difficult to achieve.

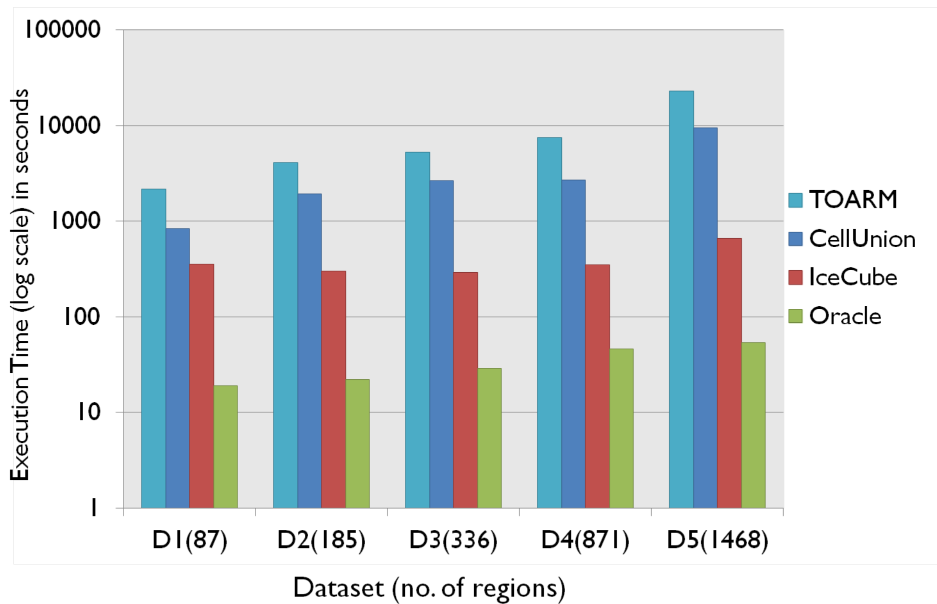


Figure 4.4: Time taken

Figure 4.5 shows the number of candidates counted by different approaches. We can see that number of candidates counted by IceCube algorithm is an order of magnitude less than TOARM algorithm for all datasets and an order of magnitude less than CellUnion algorithm for complex cubes.

Figure 4.6 gives the maximum main memory utilization of different algorithms for the synthetic datasets. Peak main memory utilization for TOARM and CellUnion algorithm is similar and is slightly less for IceCube algorithm.

Figure 4.7 shows the average time taken per region for the synthetic datasets. We can see that for IceCube algorithm, time taken per region is less than 4 seconds for all datasets and it is less than 1 second for datasets having larger number of regions in cube.

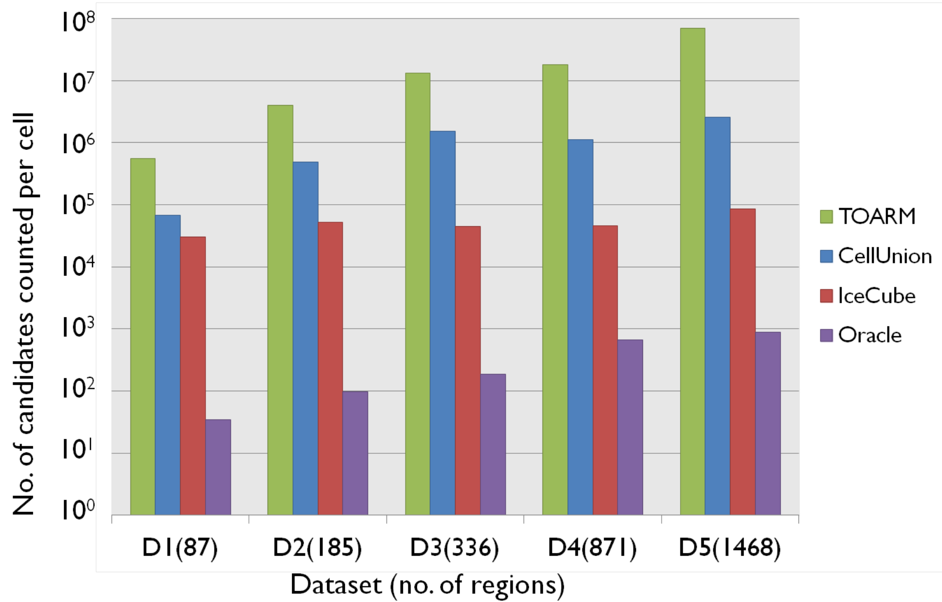


Figure 4.5: Number of candidates counted

Figure 4.8 shows the scalability of algorithms. We took dataset D3 and increased the number of transactions upto 100 million and measured performance of various algorithms. Graph shows that all algorithms scales linearly with increase in dataset size. But the slope of graph for IceCube algorithm is very less as compared to that of TOARM and CellUnion algorithms. Slope of graph for Oracle algorithm even lesser than that of IceCube algorithm.

We measured the average number of frequent itemsets for regions of different sizes, where region size is calculated as number of cells in the region. Figure 4.9 shows the graph for dataset D3. We can see that average number of frequent itemsets is more for smaller regions. This confirms the presence of many localized frequent patterns. These localized patterns are discovered by our approach but will be missed by any global mining algorithm.

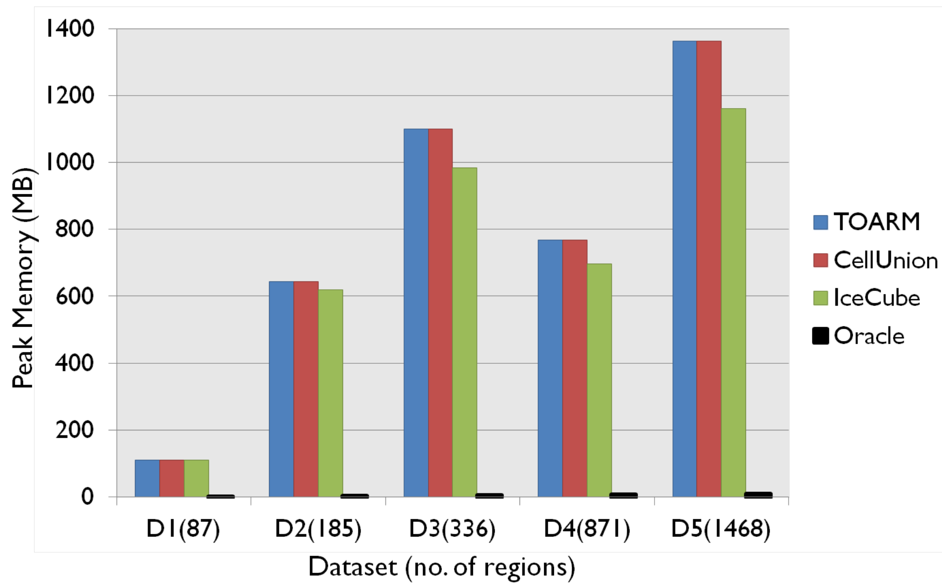


Figure 4.6: Peak Main Memory Utilization

4.4 Results: Real Data

We have taken inproceedings records from DBLP data [10] and created transactions and dimension hierarchy out of them. Paper title, authors and author affiliation are taken as basket items to build transaction. Conference and year are used as two dimensions of cube. Total of 153 computer science conferences are taken and are classified according to their areas like OS, Database etc. This generated cube having a total of 3366 cells, 1305 regions and 203K transactions. We replicated data in each cell 50 times and thus generated 10.15M transactions. Support value of 5% is used for mining.

Figure 4.10 shows the time taken by various algorithms for generated DBLP dataset. IceCube algorithm performs significantly better than TOARM and CellUnion algorithms. Also, performance of IceCube algorithm is within 4 times of Oracle algorithm for DBLP dataset.

Figure 4.11 shows the number of candidates counted by different algorithms for DBLP dataset. Candidates counted by IceCube algorithm is more than two orders of magnitude less than both TOARM and CellUnion algorithm.

Figure 4.12 gives the maximum main memory utilization of different approaches for

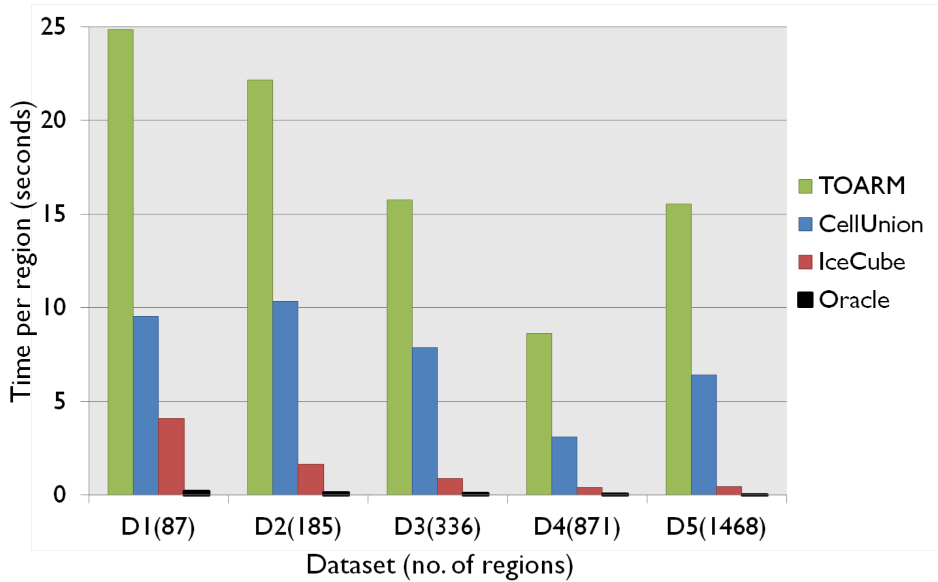


Figure 4.7: Feasibility (Average Processing Time per Region)

DBLP dataset. Peak memory utilization of TOARM and CellUnion algorithms are similar and is slightly lower for IceCube algorithm.

Table 4.3 lists some of the targeted frequent itemsets generated from DBLP dataset. For example, first rule indicates that NUS (National University of Singapore) had many papers in SIGMOD conference 2005. While second rule says that NUS had strong presence in all DASFAA conferences.

Region	**Targeted Pattern**
(2005, sigmod)	NUS
(ALL, dasfaa)	NUS
(2010, srds)	cloud computing
(2009, dasfaa)	xml search keyword
(2000,wcre)	program tools structured demonstration comprehension UnivOfVictoriaCanada margaret-anne-storey susan-elliott-sim

Table 4.3: Targeted Patterns

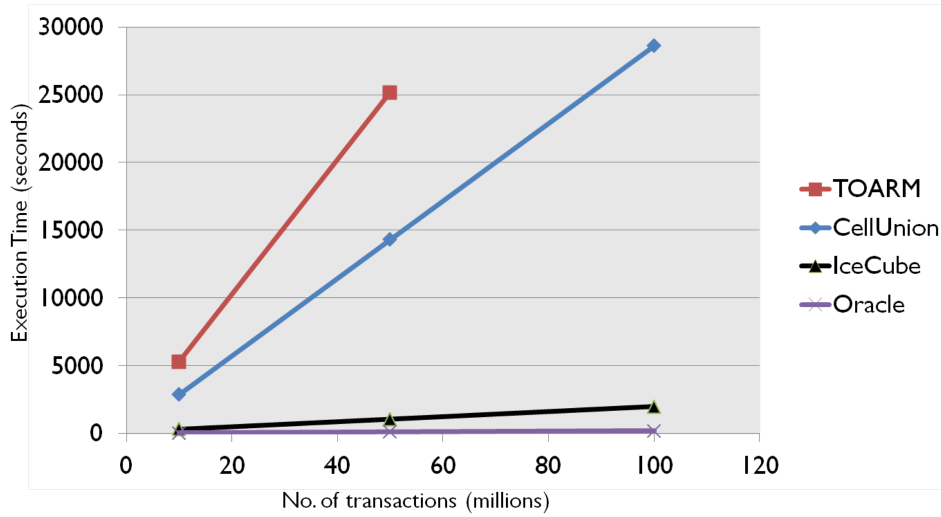


Figure 4.8: Scalability

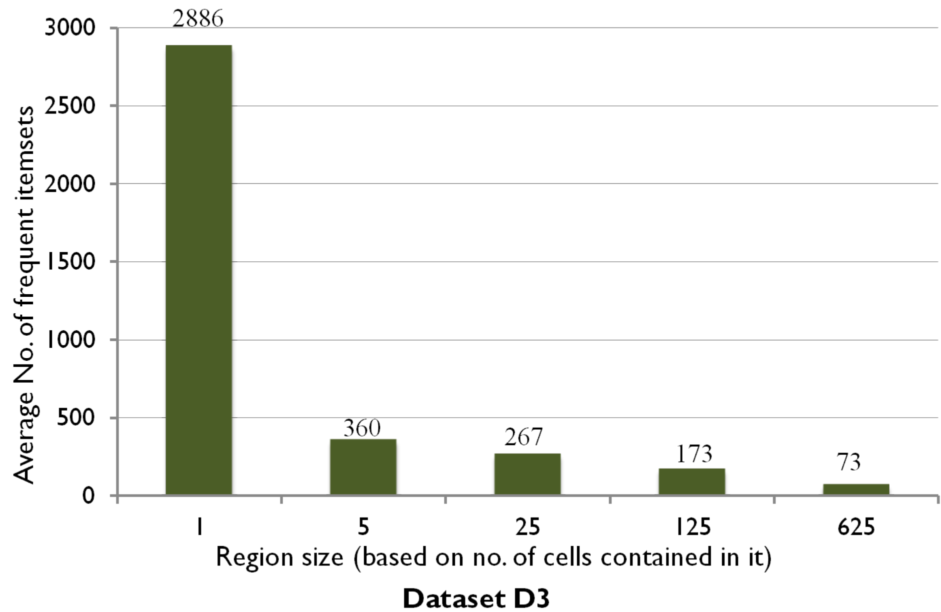


Figure 4.9: Number of Targeted Itemsets

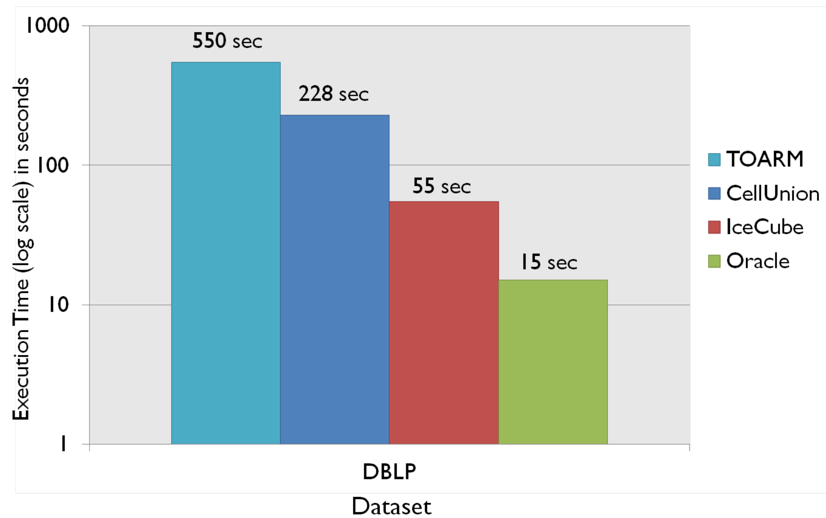


Figure 4.10: Execution Time (DBLP)

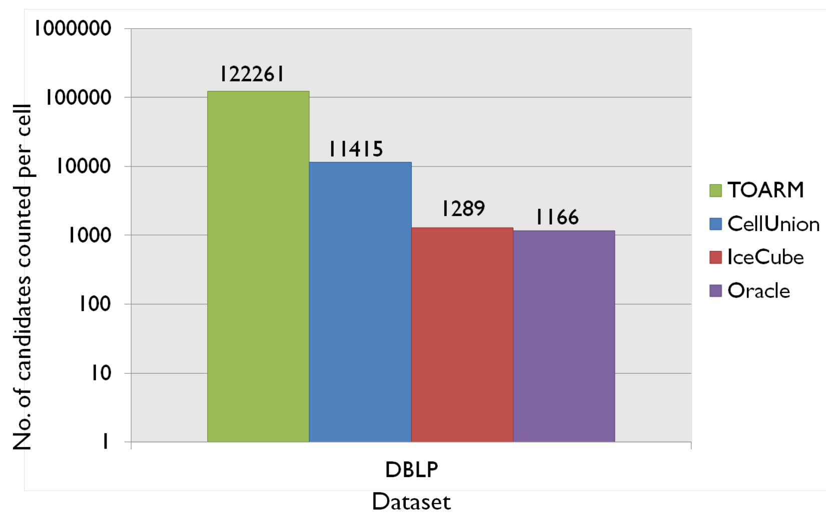


Figure 4.11: Number of candidates counted (DBLP)

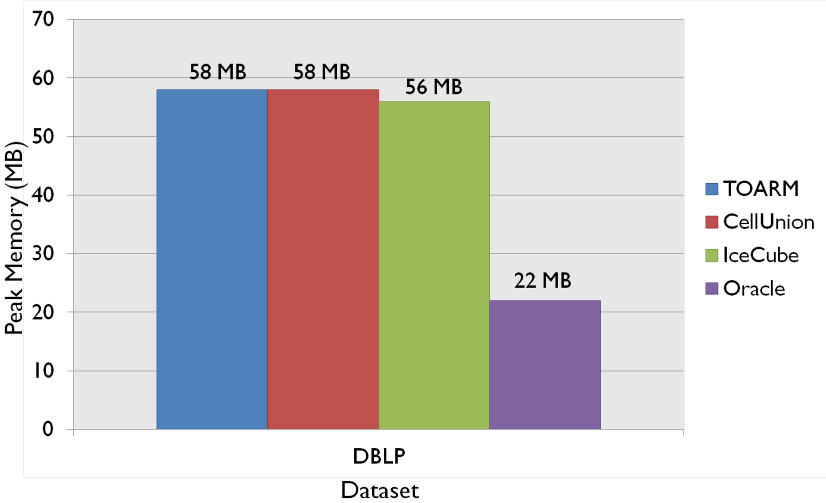


Figure 4.12: Peak Main Memory Utilization (DBLP)

Chapter 5

Related Work

Association rule mining problem was firstly introduced by Agrawal et al. [2]. They proposed the first algorithm for mining association rules, known as AIS algorithm. Apriori algorithm proposed in [3] was one of the earliest and popular algorithm for mining association rules. Apriori algorithm iteratively generates candidate itemsets and frequent itemsets from cardinality 1 to k, where k is the largest frequent itemset cardinality. It uses the property that all subsets of a frequent itemset are also frequent. It uses hash tree for counting candidate itemsets.

Han et al. [8] proposed frequent pattern tree (FP-tree) and a FP-tree based algorithm for mining frequent patterns. FP-tree is a compact representation of transaction database which stores information about frequent patterns. It uses pattern growth approach for mining frequent itemsets and does not explicitly do candidate set generation which is costly.

V. Pudi et. al. [14] addressed the issue of space available for improvement for association rule mining algorithms. They designed an Oracle algorithm which knows the identities of frequent itemsets in advance and only required to gather their counts to complete mining process. Then, they proposed ARMOR algorithm whose performance is within twice of Oracle algorithm.

But none of the above approaches deal with multidimensional market basket data.

Wang et al. [16] addressed the problem of online mining of association rules in multidimensional market basket data and proposed an approach called TOARM. It initially computes and stores cell level frequent itemsets (along with their counts) based on some minimum support (localized support). At query time, it requires user to specify the window (region) for which frequent itemsets are desired. Then, it generates candidate itemsets based on stored itemsets for user given query window. Then, it counts these candidate itemsets in query window and generates frequent itemsets.

Das et al. [7] also looked at the similar problem as that of [16]. They proposed a method called RMV to find out the minimal number of itemsets to be counted and stored for each cell so that original transaction data need not be visited for any multidimensional query window. For any user given query window (region), they don't require to do any counting of itemsets. Instead they just have to aggregate the cell level counts of itemsets which is already available to them.

But both of these approaches do not scale for our problem, where goal is to find out frequent itemsets in all regions.

Chen at al. [6] proposed prediction cubes which is used for exploratory data analysis. Each cell in prediction cubes describes a predictive model trained on data belonging to that cell. They propose a technique based on model decomposition to efficiently compute the prediction cube. Our problem formulation is on the lines of prediction cube but for market basket data.

Concept of localized support, as used by us, has previously been addressed in Aggarwal et al. [1] and Nasraoi et al. [13].

Aggarwal et al. [1] uses the concept of localized association rules. They develop a clustering algorithm to segment the transactions into disjoint clusters. Similar transactions (based on its constituent items) are put into same cluster. Then, association mining is done locally on each of these clusters. Clustering is done so as to increase the number of frequent patterns generated. But they don't take multidimensional data and dimension attributes into consideration. Clusters generated by their approach may not align with cell/region boundaries. Also clusters need not be convex regions in cube. That means

same cluster may contain transactions from different unrelated parts of cube. So, the association rules generated by their approach do not target a specific customer segment.

Nasraoi et al. [13] maps association mining as a criterion guided optimization problem. They try to find localized frequent itemsets with error tolerance in support and itemset matching. They call such itemsets as Localised Error Tolerant Frequent Pattern (LET-FP). But they also do not take the dimensions and hierarchies into consideration and hence do not guarantee targeted rules.

Chapter 6

Conclusions

In this project, we extended traditional association rule mining to the framework of data cubes. We defined the problem of mining targeted association rules using notion of localized support. An efficient IceCube algorithm was designed which incorporates ideas of interleaving, credit based pruning and cubing. Experimental evaluation on both synthetic and real datasets showed that IceCube algorithm gives good performance. The good performance is achieved due to reduction in redundant counting of candidates. We also showed that algorithm scales linearly with increase in dataset size and also performs well for complex cubes. Currently we only studied association rule mining in the framework of data cubes. Future direction could be to extend other data mining techniques for data cubes.

References

[1] C. C. Aggarwal, C. Procopiuc, and P. S. Yu, *Finding localized associations in market basket data*, Knowledge and Data Engineering (2002).

[2] R. Agrawal, T. Imielinski, and A. N. Swami, *Mining association rules between sets of items in large databases.*, SIGMOD'93, 1993, pp. 207–216.

[3] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules in large databases.*, VLDB'94, 1994, pp. 487–499.

[4] F. Bodon, *A fast apriori implementation.*, Informatics Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary.

[5] F. Bodon and L. Ronyai, *Trie: an alternative data structure for data mining algorithms*, Computers and Mathematics with Applications, 2003.

[6] B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan, *Prediction cubes.*, VLDB'05, 2005, pp. 982–993.

[7] M. Das, D. Padmanabhan, P. M. Deshpande, and R. Kannan, *Fast rule mining over multi-dimensional windows*, SDM'11, 2011.

[8] J. Han, J. Pei, and Y. Yin, *Mining frequent patterns without candidate generation.*, SIGMOD'00, 2000, pp. 1–12.

[9] V. Harinarayan, A. Rajaraman, and J. D. Ullman, *Implementing data cubes efficiently*, SIGMOD'96, 1996.

[10] The DBLP Computer Science Bibliography. http://dblp.uni trier.de/xml/.

[11] IBM Quest Market-Basket Synthetic Data Generator. http://www.cs.nmsu.edu/~cgiannel/assoc_gen.html.

[12] Apriori implementation by B. Goethals. http://adren.ua.ac.be/~goethals/software.

[13] O. Nasraoui and S. Goswami, *Mining and validating localized frequent itemsets with dynamic tolerance.*, SDM'06, 2006.

[14] V. Pudi and J. R. Haritsa, *Armor: Association rule mining based on oracle.*, FIMI., 2003.

[15] K. A. Ross and D. Srivastava, *Fast computation of sparse datacubes.*, VLDB'97, 1997, pp. 116–125.

[16] C.-Y. Wang, S.-S. Tseng, and T.-P. Hong, *Flexible online association rule mining based on multidimensional pattern relations.*, Inf. Sci. (2006), 1752–1780.