

Fast Rule Mining Over Multi-Dimensional Windows

Mahashweta Das* Deepak P† Prasad M Deshpande‡ Ramakrishnan Kannan§

Abstract

Association rule mining is an indispensable tool for discovering insights from large databases and data warehouses. The data in a warehouse being multi-dimensional, it is often useful to mine rules over subsets of data defined by selections over the dimensions. Such interactive rule mining over multi-dimensional query windows is difficult since rule mining is computationally expensive. Current methods using pre-computation of frequent itemsets require counting of some itemsets by revisiting the transaction database at query time, which is very expensive. We develop a method (RMW) that identifies the minimal set of itemsets to compute and store for each cell, so that rule mining over any query window may be performed without going back to the transaction database. We give formal proofs that the set of itemsets chosen by RMW is sufficient to answer any query and also prove that it is the optimal set to be computed for 1 dimensional queries. We demonstrate through an extensive empirical evaluation that RMW achieves extremely fast query response time compared to existing methods, with only moderate overhead in pre-computation and storage.

1 Introduction

Association Rule Mining (ARM) is an indispensable tool for discovering potentially meaningful knowledge in large databases. Ever since it was introduced [1, 2], it has been widely researched upon [3, 4]. ARM uncovers latent relations between items in a database such that the occurrence of certain items increases the likelihood of the occurrence of certain other items in the same transaction. For multi-dimensional data, the focus of a mining task can be a view, a subset of transactions, specified at query time [5, 6].

EXAMPLE 1. Consider a global retail chain and its transaction database. Let location and time be two of the attributes associated with each transaction. The multi-dimensional grid representation of the database with one parameter per grid dimension (each transaction held in the appropriate cell), is illustrated in Figure 1. Both the attributes have an inherent hierarchical structure;

location India can be the child node of the larger Indian Subcontinent or South Asia node. Now, the South Asia Manager may be interested in quarterly trends (association rules) for his region. For Q3 2008, this corresponds to mining rules from transactions in window W1 that spans several cells. The North America Manager may be interested in December sales trends (window W2). Such tasks are often exploratory and query windows are varied repeatedly until significant actionable patterns are discovered. This motivates a system to support querying over any arbitrary window specification.

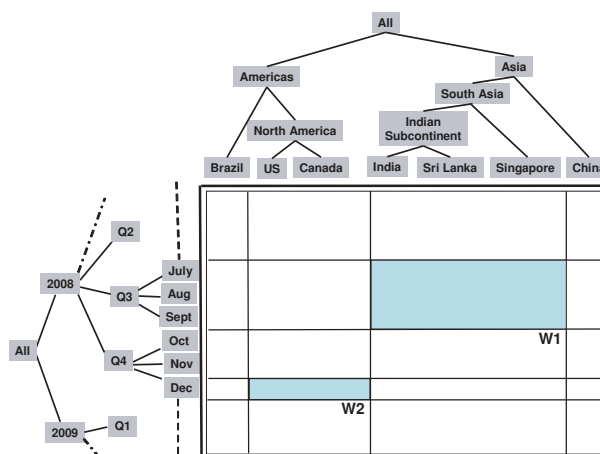


Figure 1: Query Window Selection.

Rule mining algorithms are computationally intensive and have response times of the order of hundreds of seconds [7, 6], making such interactive analysis difficult. Existing methods for online rule mining such as TOARM [6] pre-computes and stores frequent itemsets based on some pre-specified minimum support, which eventually require counting of itemsets by revisiting the transaction database at query time.

We address the problem of efficient rule-mining over windows specified at query-time; we present a technique that completely avoids database scans during query time processing. Our contributions are:

- A method for handling 1- d query windows using provably optimal storage, that retrieves rule mining

*University of Texas at Arlington

†IBM Research - India, Bangalore

‡IBM Research - India, Bangalore

§IBM India Software Lab, Bangalore

results without scanning the transaction database at query time.

- *RMW (Rule Mining over Windows)*, its generalization to handle multi-dimensional query windows.
- An extensive empirical evaluation illustrating the effectiveness of our approach against the state-of-art in providing extremely fast query response time.

2 Related Work

Mining association rules from transaction databases has been a topic of much research [1, 2]. Improvements to the basic ARM scheme in [2] optimize the expensive first phase of computing frequent itemsets (e.g., [8, 9]). Query-time view specifications can be considered to be an instance of constrained ARM [10]; query windows here can be modeled as a specific type of data and knowledge constraints. However, in spite of these constraints and optimizations, ARM is still expensive. Supporting interactive analysis over different view specifications requires some form of pre-computation, which is the approach taken in this paper.

Itemset discovery over pre-specified mining views is well-studied [11]. Our work is closer in spirit to incremental mining algorithms that efficiently maintain the set of association rules over a dynamic database. They maintain previously mined patterns (including those that are slightly short of achieving the minimum support) so that re-processing of the database need not be performed for each update [12, 13, 14]; similar techniques have been used in counting over data streams [15]. Incremental mining is like querying over a 1- d window spanning the entire database, cells ordered temporally. The problem we address is more general since we allow arbitrary window specifications. Another source of variation in mining specifications is changing user specified *minimum support*. Algorithms capable of handling such variations work by pre-computing association rules for a lesser support and maintaining them efficiently so that rules satisfying a higher user-defined support can be retrieved efficiently [16, 17]. Results of previous queries and could be cached and reused [18], the mining view being still static.

Many studies have addressed the issue of performing data mining tasks on multi-dimensional data cubes [19, 20, 21, 22, 23]. In a prediction cube [22], a predictive model is built over pre-specified views (cells), summarizing the data in that view. They focus on different predictive models and not on association rule mining. Mining association rules from data cubes can be classified into three: intra-dimensional, inter-dimensional and hybrid [20]. Intra-dimensional association rule cover repetitive predicates from a single

dimension, whereas inter-dimensional association rules are mined from multiple dimensions without repetition of predicates in each dimension. We consider intra-dimensional mining on views defined by the other dimensions. Cubegrades [21] are a generalization of association rules that identify significant changes that affect measures when a cube is modified through specialization, generalization or mutation. All these methods generate and compute the frequent itemsets at query time, making interactive analysis infeasible. We avoid the expensive step by pre-computing the candidate itemsets, requiring only an aggregation at query time.

Pre-computation to enable interactive query response times is a popular OLAP technique [24, 25]. These exploit the distributive properties of many aggregate functions to answer queries by aggregating pre-computed results. However, using pre-computed results for data mining is not straightforward since the mining operator does not have the properties of distributive aggregate functions. This has been addressed in TOARM [6], the method we compare against. It uses pre-computed results to handle query-time mining view specifications; it requires counting of some candidates by scanning the transaction database at query time.

3 Preliminaries

Consider a database \mathcal{D} of n transactions, $\{T_1, T_2, \dots, T_n\}$. Each transaction consists of an itemset and some attributes associated with it. In Example 1, the attributes are *time* and *location* of purchase. These attributes can be used for selecting various subsets of the database on which the mining task can be applied. *W1* corresponds to *location* values in *SouthAsia* and *time* values in *Q3*. Consider m such attributes, $\{A_1, A_2, \dots, A_m\}$; transaction T_j is then represented as $\{I_j, A_{1,j}, A_{2,j}, \dots, A_{m,j}\}$ where I_j is the corresponding itemset and $A_{i,j}$ is the value of the i^{th} attribute in the j^{th} transaction. An example transaction having two items and has time and location as attributes could be $\{\{Item1, Item5\}, 16 : 59, Bangalore\}$.

One-Dimensional Window Consider an attribute A_i and a pre-specified ordering of values for that attribute $[v_{i,1}, v_{i,2}, \dots, v_{i,p}]$. We define a query window as a specification of the form $(A_i, [x, y])$, $y \geq x$ that denotes the selection of a subset $\mathcal{D}_{A_i[x,y]}$ of \mathcal{D} .

$$\mathcal{D}_{A_i[x,y]} = \{T_j \in \mathcal{D} | A_{i,j} \in \{v_{i,x}, v_{i,x+1}, \dots, v_{i,y}\}\}$$

Thus, $\mathcal{D}_{A_i[x,y]}$ contains all those transactions that have the value for the i^{th} attribute in between the x^{th} and y^{th} values (both inclusive) in the pre-defined ordering of values for A_i . For example, the pre-

defined ordering for the time attribute could just be the chronological order of transaction days; a window in such a case would denote a contiguous sequence of times (e.g., a quarter, a month, a fortnight etc). For attributes that do not have an inherent order, the hierarchy of values can be used to define an order. Any choice of a single internal node in a hierarchy would lead to an intuitive window in an ordering of values that keeps siblings together. The choice of the internal node *South Asia* leads to a window involving *India*, *Sri Lanka* and *Singapore*. Such hierarchies are quite common in multi-dimensional OLAP databases.

Multi-Dimensional Windows A multi-dimensional query window is a combination of one-dimensional windows over a set of attributes, having one window per chosen attribute. Consider two one-dimensional query windows $(A_{i_1}, [x_1, y_1])$ and $(A_{i_2}, [x_2, y_2])$, $i_1 \neq i_2$. The *mining view* would then be composed of:

$$\mathcal{D}_{A_{i_1}[x_1, y_1], A_{i_2}[x_2, y_2]} = \mathcal{D}_{A_{i_1}[x_1, y_1]} \cap \mathcal{D}_{A_{i_2}[x_2, y_2]}$$

For example, W_1 in Figure 1 represents a combination of two 1- d windows denoted by their intersection. When a multi-dimensional window is specified over a subset of available attributes, we would then be choosing all values of the attributes not specified in the window. When we choose just the *South Asia* node, we would choose transactions in South Asia across all values of the time attribute.

4 Problem Statement

Consider such a transaction database with attributes for each transaction (e.g., location, time etc). For any **arbitrary multi-dimensional window (over the attributes) and support specified by the user at query time**, we intend to compute frequent itemsets from transactions within that window at real-time. These would be processed using the confidence criterion to produce association rules. More specifically, we address the problem of pre-computing and storing enough itemset frequency counts so that the transaction database would not have to be consulted at query time.

The number of possible query windows is exponential in the number of dimensions and quadratic in the number of values per dimension; thus, pre-computing frequent itemsets for every query window is infeasible. TOARM-style processing, on the other hand, incurs lower storage at the expense of having to do counting over the database at query time. *The key thus is to identify the minimal set of itemsets to compute and store for each cell, so that any query can be answered without going back to the database of transactions.* Our pre-processing approach would guarantee the correct re-

sults without going back to the transactions, only for support values greater than a specific threshold (which is used in the pre-processing phase). Such minimum support thresholds are common in pre-processing approaches for mining [6, 26].

5 Rule Mining Over Windows

We now present our approach, Rule Mining over Windows (*RMW*) and describe pre-processing techniques for 1- d and 2- d windows (with a generalization to multiple dimensions). We also discuss query time processing.

a)	$l_1: 11$ $l_2: 15$	$l_1: 11$ $l_3: 10$		$l_4: 11$	$l_1: 10$
	C_1	C_2	C_3	C_4	C_5
b)	$l_1: 11$ $l_2: 15$	$l_1: 11$ $l_2: 8$ $l_3: 10$	$l_1: 8$ $l_2: 5$ $l_3: 9$	$l_1: 7$ $l_2: 8$ $l_4: 11$	$l_1: 10$ $l_2: 7$ $l_3: 5$

Figure 2: Pre-processing

5.1 Motivating Example Consider a 1- d space of 5 cells $C_1, C_2 \dots C_5$ (Figure 2). Let each cell have 1k transactions and the minimum support be 1%, (i.e., a count of 10); Figure 2a shows the frequent itemsets in each cell. Now, for each cell, itemset pair $[C, I]$ where I is not frequent in C , we determine whether C could be part of a window in which I is frequent. If such a window exists, we count and store the support of I in C so that we can provide its exact support of I for the corresponding query; else, we omit counting it. This omission reduces storage cost, at the same time, guaranteeing that any query can be answered directly from stored counts (without query time support counting). For $[C_3, I_2]$, $[C_1 - C_3]$ is a window having C_3 where I_2 can be frequent. The count of I_2 in C_1 is 15, and the counts in C_2 and C_3 can at most be 9 (its lesser than 10, otherwise I_2 would be frequent in those cells), giving a maximum support of 33 which is greater than the minimum support required of 30 (1% of 3k). Thus the support of I_2 needs to be computed in C_3 . Had the count of I_2 been 11 in C_1 , the upper bound (i.e., $11+9+9=29$) would be below the threshold, thus not necessitating counting of its support in C_3 . Similar reasoning can help determine all the potential itemsets across cells (Figure 2b).

In Figure 2, we count the support of I_2 in C_3 based on the reasoning above. Upon counting, we may realize that the actual support is 3. Given this additional information, we know that I_2 cannot be frequent in the window $[C_1 - C_3]$ since the refined upper bound is now

15+9+3 = 27, which is lesser than the required support of 30. Thus, we do not need to store the support of I_2 in C_3 since I_2 cannot be frequent in a window involving C_3 . Such refined upper bounds after support counting are used for further filtering.

5.2 Handling 1-Dimensional Windows Consider a specific attribute A and an ordering of values, $[v_1, v_2, \dots, v_p]$. For notational convenience, we denote the subset of \mathcal{D} that takes the value v_k for the attribute A (i.e., $\mathcal{D}_{A[k,k]}$) by \mathcal{D}_k . Let the pre-specified minimum support criterion for a mining task be denoted by μ (expressed as a fraction). Thus, an itemset \mathcal{I} is frequent in \mathcal{D}_k if it has an absolute support of at least $\mu * |\mathcal{D}_k|$. We define a notion of *credit*, $\mathcal{C}_{\mathcal{I},k}$ for an itemset \mathcal{I} and a choice of value v_k :

$$(5.1) \quad \mathcal{C}_{\mathcal{I},k} = \text{Support}(\mathcal{I}, \mathcal{D}_k) - \mu * |\mathcal{D}_k|$$

$\text{Support}(\mathcal{I}, \mathcal{D}_k)$ denotes the support of \mathcal{I} in the set of transactions \mathcal{D}_k . Credit denotes the excess or shortage of support of an itemset \mathcal{I} in \mathcal{D}_k . Frequent itemsets have a positive credit that can be used for neighboring cells, whereas infrequent itemsets have a negative credit that can use up the additional credit from neighboring cells. A *positive credit indicates that the itemset is frequent in that cell*. The cumulative credit for a range $[k_1, k_2], k_1 \leq k_2$ is:

$$(5.2) \quad \mathcal{C}_{\mathcal{I},[k_1,k_2]} = \sum_{i=k_1}^{k_2} \mathcal{C}_{\mathcal{I},i}$$

The following theorem follows from this definition:

THEOREM 5.1. *An itemset \mathcal{I} is frequent in a query window $[k_1, k_2]$ if and only if $\mathcal{C}_{\mathcal{I},[k_1,k_2]} \geq 0$.*

Now, we define a notion of directional cumulative credit when $k_1 \leq k_2$ in a recursive fashion as follows:

$$(5.3) \quad \mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2} = \begin{cases} \max\{0, \mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2 - 1}\} + \mathcal{C}_{\mathcal{I},k_2}, & k_2 > k_1 \\ \mathcal{C}_{\mathcal{I},k_1}, & k_2 = k_1 \end{cases}$$

$$(5.4) \quad \mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2} = \begin{cases} \max\{0, \mathcal{C}_{\mathcal{I},k_1 + 1 \leftarrow k_2}\} + \mathcal{C}_{\mathcal{I},k_1}, & k_1 < k_2 \\ \mathcal{C}_{\mathcal{I},k_2}, & k_1 = k_2 \end{cases}$$

At any point, only a positive cumulative credit is carried forward; else it is *reset* to 0. $\mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2}$ is the maximum credit any query window starting at or beyond k_1 and ending at k_2 can have. The following properties follow from the definition since only positive credits are carried forward:

PROPERTY 5.1. $\mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2} \geq \mathcal{C}_{\mathcal{I},[k_1,k_2]}$; $\mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2} \geq \mathcal{C}_{\mathcal{I},[k_1,k_2]}$

PROPERTY 5.2. $\mathcal{C}_{\mathcal{I},k'_1 \rightarrow k_2} \geq \mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2}$ if $k'_1 \leq k_1$; $\mathcal{C}_{\mathcal{I},k_1 \leftarrow k'_2} \geq \mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2}$ if $k'_2 \geq k_2$

PROPERTY 5.3. $\mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2} = \mathcal{C}_{\mathcal{I},[k',k_2]}$ where k' is the last point of reset. Also, $\mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2} = \mathcal{C}_{\mathcal{I},[k_1,k']}$ for the reverse direction with the last reset at k' .

Now, suppose we have pre-computed the supports for some itemsets in each cell. Let $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_p\}$ denote the pre-computed results, where \mathcal{P}_k denotes the pre-computed itemsets and their supports for \mathcal{D}_k . Let \mathcal{P}_k have the property that it includes all the frequent itemsets in \mathcal{D}_k , i.e. $\mathcal{P}_k \supseteq \mathcal{F}_k$ where \mathcal{F}_k denotes the frequent itemsets of \mathcal{D}_k . The credit estimate based on the pre-computed set \mathcal{P}_k would then be:

$$(5.5) \quad \mathcal{C}_{\mathcal{I},k}^{\mathcal{P}_k} = \begin{cases} \text{Support}(\mathcal{I}, \mathcal{D}_k) - \mu * |\mathcal{D}_k|, & \text{if } \mathcal{I} \in \mathcal{P}_k \\ \lceil \mu * |\mathcal{D}_k| \rceil - 1 - \mu * |\mathcal{D}_k|, & \text{otherwise} \end{cases}$$

$\mathcal{C}_{\mathcal{I},k}^{\mathcal{P}_k}$ denotes an upper bound of the value of the actual credit $\mathcal{C}_{\mathcal{I},k}$. In cases when $\mathcal{I} \in \mathcal{P}_k$, we know the exact value of its support; in other cases, the support of \mathcal{I} , being an integer, can at most be $\lceil \mu * |\mathcal{D}_k| \rceil - 1$, since \mathcal{I} cannot be frequent (since $\mathcal{P}_k \supseteq \mathcal{F}_k$). Thus we have:

PROPERTY 5.4. $\mathcal{C}_{\mathcal{I},k} \leq \mathcal{C}_{\mathcal{I},k}^{\mathcal{P}_k}$

The cumulative scores and the directional cumulative scores are estimated based on the pre-computed results using equations 5.1, 5.3 and 5.4, the only difference being that $\mathcal{C}_{\mathcal{I},k}^{\mathcal{P}_k}$ is used instead of $\mathcal{C}_{\mathcal{I},k}$. We denote these as $\mathcal{C}_{\mathcal{I},[k_1,k_2]}^{\mathcal{P}}$, $\mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2}^{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2}^{\mathcal{P}}$ respectively. This leads to the following:

PROPERTY 5.5. $\mathcal{C}_{\mathcal{I},[k_1,k_2]}^{\mathcal{P}} \geq \mathcal{C}_{\mathcal{I},[k_1,k_2]}$

PROPERTY 5.6. $\mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2}^{\mathcal{P}} \geq \mathcal{C}_{\mathcal{I},k_1 \rightarrow k_2}$

PROPERTY 5.7. $\mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2}^{\mathcal{P}} \geq \mathcal{C}_{\mathcal{I},k_1 \leftarrow k_2}$

We now present a property of such cumulative credits and then, the two phases of our approach.

THEOREM 5.2. *If $(\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{P}} + \max\{0, \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{P}}\})$ is negative, there does not exist $k_1, k_2, k_1 \leq k \leq k_2$ such that \mathcal{I} is frequent in $\bigcup_{i=k_1}^{k_2} \mathcal{D}_i$, i.e., \mathcal{I} would be infrequent in every window involving k .*

Proof. Let there exist $k_1, k_2, k_1 \leq k \leq k_2$ such that \mathcal{I} is frequent in $\bigcup_{i=k_1}^{k_2} \mathcal{D}_i$. We have:

$\mathcal{C}_{\mathcal{I},[k_1,k_2]} \geq 0$ (Theorem 5.1)

$\mathcal{C}_{\mathcal{I},[k_1,k]} + \mathcal{C}_{\mathcal{I},[k+1,k_2]} \geq 0$ (using Equation 5.2)

$\mathcal{C}_{\mathcal{I},k_1 \rightarrow k} + \mathcal{C}_{\mathcal{I},k+1 \leftarrow k_2} \geq 0$ (Property 5.1)

$\mathcal{C}_{\mathcal{I},1 \rightarrow k} + \mathcal{C}_{\mathcal{I},k+1 \leftarrow p} \geq 0$ (Property 5.2 and $1 \leq k \leq p$)

$\mathcal{C}_{\mathcal{I},1 \rightarrow k} + \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{P}} \geq 0$ (Property 5.6 and 5.7)

$\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{P}} + \max\{0, \mathcal{C}_{\mathcal{I},p \leftarrow k+1}^{\mathcal{P}}\} \geq 0$

This completes the proof by contradiction.

Alg. 1 Phase I Processing

1. for each $k, 1 \leq k \leq p$
 2. $\mathcal{F}'_k = \mathcal{F}_k$
 3. for each $\mathcal{I} \in \bigcup_{i=1}^p \mathcal{F}_i$
 4. for each $k, 1 \leq k \leq p$
 5. if $((\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}} + \max\{0, \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{F}}\}) \geq 0)$
 6. Count support, s , of \mathcal{I} in \mathcal{D}_k
 7. Add $[\mathcal{I}, s]$ to \mathcal{F}'_k
-

Phase I Phase I is based on the insight that an itemset that is frequent in a query window has to be frequent in at least one cell in the range. Thus, it is sufficient to consider only the frequent itemsets in each \mathcal{D}_k for pre-computation. The frequent itemsets for \mathcal{D}_k along with their supports are held in the set \mathcal{F}_k . Let $\mathcal{F} = \{\mathcal{F}_1, \dots, \mathcal{F}_p\}$ denote the pre-computed results consisting of only the frequent itemsets in each cell. The Phase I processing is outlined in Algorithm 1. It progressively builds $\mathcal{F}'_{k,s}$, supersets of corresponding set of frequent itemsets (i.e., \mathcal{F}_k s), by including the support of certain itemsets that are not frequent in the corresponding \mathcal{D}_k s. Specifically, for a particular k , it counts the frequency of every itemset \mathcal{I} where $(\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}} + \max\{0, \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{F}}\})$ is non-negative (Refer lines 5-7). This is done since there is a possibility that \mathcal{I} is frequent in a window involving k . Also, Theorem 5.2 suggests that it is sufficient to add only such itemsets to the pre-computed results. We will use the \mathcal{F}'_k s for the Phase II processing. Figure 2a shows the \mathcal{F}_k s and 2b shows the \mathcal{F}'_k s for the example.

Phase II Phase II is a filtering phase where we use the exact counts of the additional frequent itemsets computed in Phase I for further filtering. Let $\mathcal{F}' = \{\mathcal{F}'_1, \dots, \mathcal{F}'_p\}$ be the pre-computed results at the end of Phase I. Phase II is similar to Phase I; but, we use \mathcal{F}'_k s built in Phase I instead of the \mathcal{F}_k s. $\mathcal{C}_{\mathcal{I},k}^{\mathcal{F}'}$ s are tighter upper bounds of the excess support available (i.e., $\mathcal{C}_{\mathcal{I},k}^{\mathcal{F}}$) than $\mathcal{C}_{\mathcal{I},k}^{\mathcal{F}}$, since the exact support for more itemsets have been computed in \mathcal{F}'_k leading to:

PROPERTY 5.8. $\mathcal{C}_{\mathcal{I},k}^{\mathcal{F}'} \leq \mathcal{C}_{\mathcal{I},k}^{\mathcal{F}}$

Subsequently, the cumulative credits of this phase are also upper bounded by the cumulative credits of Phase I. Phase II processing is presented in Algorithm 2. For every k , each itemset satisfying the condition checked for in Algorithm 1 is purged out of \mathcal{F}'_k (Refer line 3). Next, we discuss a strong property of modified \mathcal{F}'_k s.

THEOREM 5.3. *At the end of Phase II processing, an itemset \mathcal{I} is in \mathcal{F}'_k if and only if there exists at least*

Alg. 2 Phase II Processing

1. for each $k, 1 \leq k \leq p$
 2. for each $\mathcal{I} \in \mathcal{F}'_k$
 3. if $((\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'} + \max\{0, \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{F}'}\}) < 0)$
 4. Remove \mathcal{I} from \mathcal{F}'_k
-

one window involving k in which it is frequent.

Proof. The ‘if’ follows from Theorem 5.2. The expression in line 3 becoming negative rules out the possibility of a window involving k in which \mathcal{I} is frequent.

To prove the only if part, we will show that each \mathcal{I} in \mathcal{F}'_k at the end of Phase II would have at least one window involving k in which it is frequent; *specifically, we show that the points at which the directional cumulative credits got reset (from both directions) form the end points of one such window.* Let us consider k_1 , the latest (i.e., numerically largest) point for which the moving counter for computing $\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'}$ got reset, i.e., $\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'} = \mathcal{C}_{\mathcal{I},k_1 \rightarrow k}^{\mathcal{F}'}$. This implies that $\mathcal{C}_{\mathcal{I},1 \rightarrow k''}^{\mathcal{F}'} \geq 0$ holds for all $k'', k_1 \leq k'' < k$; a reset would have happened later than k_1 otherwise. Since the corresponding Phase I scores are higher or equal (Property 5.8), $\mathcal{C}_{\mathcal{I},1 \rightarrow k''}^{\mathcal{F}} \geq 0$ also holds for all $k'', k_1 \leq k'' < k$. This would have forced the actual supports of \mathcal{I} to be computed in line 6 of Algorithm 1 for every such k'' (and thus, to be included in \mathcal{F}'_k). Thus, $\mathcal{I} \in \mathcal{F}'_{k''}$, for all $k'', k_1 \leq k'' < k$. Thus, $\mathcal{C}_{\mathcal{I},k''} = \mathcal{C}_{\mathcal{I},k''}^{\mathcal{F}'}$ (from Equation 5.5), i.e., we have the exact counts of \mathcal{I} for all $k'', k_1 \leq k'' < k$. Thus:

$$\mathcal{C}_{\mathcal{I},[k_1,k]} = \mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'}$$

A similar condition, $\mathcal{C}_{\mathcal{I},[k,k_2]} = \mathcal{C}_{\mathcal{I},k \leftarrow p}^{\mathcal{F}'}$ follows from analogously for the credit from the other direction where k_2 is the last point of reset of $\mathcal{C}_{\mathcal{I},k \leftarrow p}^{\mathcal{F}'}$ since:

$$\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'} + \max\{0, \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{F}'}\} = \max\{0, \mathcal{C}_{\mathcal{I},1 \rightarrow k-1}^{\mathcal{F}'}\} + \mathcal{C}_{\mathcal{I},k \leftarrow p}^{\mathcal{F}'}$$

We know the actual support of \mathcal{I} in the k^{th} cell since its in \mathcal{F}_k (being in \mathcal{F}'_k). We now use these inferences in the simple proof below. \mathcal{I} being in \mathcal{F}'_k after phase 2, we have:

$$\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'} + \max\{0, \mathcal{C}_{\mathcal{I},k+1 \leftarrow p}^{\mathcal{F}'}\} > 0$$

$$\mathcal{C}_{\mathcal{I},1 \rightarrow k}^{\mathcal{F}'} + \mathcal{C}_{\mathcal{I},k \leftarrow p}^{\mathcal{F}'} - \mathcal{C}_{\mathcal{I},k}^{\mathcal{F}'} > 0 \text{ (simple rewrite)}$$

$$\mathcal{C}_{\mathcal{I},[k_1,k]} + \mathcal{C}_{\mathcal{I},[k,k_2]} - \mathcal{C}_{\mathcal{I},k} > 0 \text{ (using inferences above)}$$

$$\mathcal{C}_{\mathcal{I},[k_1,k_2]} > 0 \text{ (simple rewrite)}$$

This means that \mathcal{I} is frequent in the window $[k_1, k_2]$, which contains k , thus proving the only if part.

Figure 2b shows the itemsets filtered out from \mathcal{F}'_k s in Phase II for the example. We will outline similar, but weaker, conditions for 2-dimensional windows.

5.3 Handling 2-dimensional Windows Let us consider two attributes, A_1 and A_2 that have p_1 and p_2 distinct values respectively. Let the corresponding orderings be $[v_{1,1}, v_{1,2}, \dots, v_{1,p_1}]$ and $[v_{2,1}, v_{2,2}, \dots, v_{2,p_2}]$. $\mathcal{D}_{i,j}$ would then represent the subset of transactions that take the value $v_{1,i}$ and $v_{2,j}$ for A_1 and A_2 respectively.

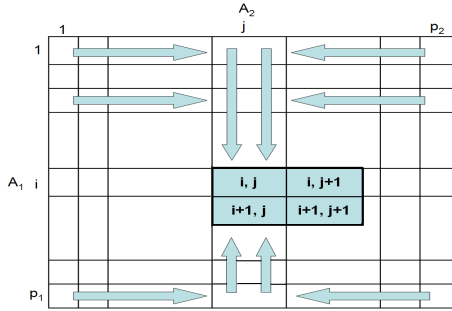


Figure 3: Phase I for 2-d windows.

Phase I Similar to Section 5.2, we compute the frequent itemsets in each cell $\mathcal{D}_{i,j}$. Let \mathcal{F} denote these pre-computed results. Now, each cell (i, j) can be approached from 4 directions, i.e., $(1, j)$, (p_1, j) , $(i, 1)$ and (i, p_2) . The first two cases consider variation in A_1 (similar to considering only A_1 in 1-d) and the rest two consider variation in A_2 . Consider the $\mathcal{D}_{i,j}$ s arranged along a 2-d grid as shown in the Figure 3. Now, for each row k , representing the value $v_{1,k}$ for A_1 , we build cumulative sums (using the same strategy as for 1-d) for each (k, j) from both directions, i.e., from 1 and p_2 . This is represented by the horizontal arrows in the Figure. Now, each cell (k, j) ($1 \leq k \leq p_1$), would have two cumulative sums, one from each direction. Let these be $\mathcal{C}_{\mathcal{I},k,1 \rightarrow j}$ and $\mathcal{C}_{\mathcal{I},k,j \leftarrow p_2}$ based on the direction of aggregation. Now, we aggregate these two scores (corresponding to the two horizontal directions) separately towards (i, j) . This leads to four cumulative scores represented as $\mathcal{C}_{\mathcal{I},1 \rightarrow i,1 \rightarrow j}$, $\mathcal{C}_{\mathcal{I},i \leftarrow p_1,1 \rightarrow j}$, $\mathcal{C}_{\mathcal{I},1 \rightarrow i,j \leftarrow p_2}$ and $\mathcal{C}_{\mathcal{I},i \leftarrow p_1,j \leftarrow p_2}$. $\mathcal{C}_{\mathcal{I},1 \rightarrow i,1 \rightarrow j}$ refers to the cumulative sums aggregated from the top row to i , of the scores $\mathcal{C}_{\mathcal{I},k,1 \rightarrow j}$, $1 \leq k \leq i$. This specific score is an upper bound of the excess credit available in any multi-dimensional window, $\mathcal{D}_{A_1[a,i],A_2[b,j]}$ for any combination of values for a and b such that $1 \leq a \leq i$, $1 \leq b \leq j$. Informally, $\mathcal{C}_{\mathcal{I},1 \rightarrow i,1 \rightarrow j}$ being negative implies that (i, j) cannot be the bottom right corner of a 2-d window in which \mathcal{I} is frequent.

The Phase I processing, on similar lines as that for the 1-d case, counts the support of every itemset \mathcal{I} in such cells (i, j) where the following sum is non-negative:

$$(5.6) \quad \mathcal{C}_{\mathcal{I},1 \rightarrow i,1 \rightarrow j}^{\mathcal{F}} + \max\{0, \mathcal{C}_{\mathcal{I},1 \rightarrow i,j+1 \leftarrow p_2}^{\mathcal{F}}\} + \max\{0, \mathcal{C}_{\mathcal{I},i+1 \leftarrow p_1,1 \rightarrow j}^{\mathcal{F}}\} + \max\{0, \mathcal{C}_{\mathcal{I},i+1 \leftarrow p_1,j+1 \leftarrow p_2}^{\mathcal{F}}\}$$

This is analogous to the condition in Algorithm 1 (Line 5). A theorem analogous to Theorem 5.2 can be proved on exactly the same lines for 2-d queries. It is based on the observation that any 2-d window involving (i, j) where \mathcal{I} is frequent can be split into four quadrants, one of which would have (i, j) as the bottom right corner. Figure 3 illustrates this idea. Consider a partitioning of the space as shown in the figure. Now, each sub-expression in Equation 5.6 takes care of sub-windows of a 2-d window involving (i, j) wholly contained within each quadrant. $\mathcal{C}_{\mathcal{I},1 \rightarrow i,1 \rightarrow j}$ is the upper bound of the excess credit in the sub-window wholly contained in the top-left quadrant (thus having the bottom right corner at (i, j)). Similarly, $\max\{0, \mathcal{C}_{\mathcal{I},1 \rightarrow i,j+1 \leftarrow p_2}^{\mathcal{F}}\}$ is an upper bound of the excess credit in the sub-window in the top-right quadrant with the bottom-left corner at $(i, j+1)$. Similar conditions take care of the other two quadrants. Negative sums from the quadrants not containing (i, j) are not propagated (due to the reset in the $\max\{\dots\}$ in the three terms) in order to consider windows that do not overlap with that corresponding quadrant. Thus, if the expression in Equation 5.6 is negative, there cannot exist a 2-d window involving (i, j) where \mathcal{I} is frequent.

The upper bound for the quadrant with (i, j) at the bottom right corner could be built by either moving right (on A_2) and then aggregating downwards (as in Figure 3) or by moving down (on A_1) first and then aggregating rightwards. These could lead to different upper bounds due to the resets. The ordering that will lead to the strongest upper bound depends on the actual counts and cannot be pre-determined. We stick to a fixed pre-chosen attribute ordering here.

Phase II Similar to the handling of 1-d query windows, the Phase II processing starts by refining the cumulative sums using the updated counts as obtained from Phase I (denoted by \mathcal{F}'). We follow a similar strategy as in Phase I to compute the refined cumulative sums, 4 per cell. Here, we exclude every itemset \mathcal{I} from $\mathcal{F}'_{i,j}$ where the following expression evaluates to a negative value:

$$(5.7) \quad \mathcal{C}'_{\mathcal{I},1 \rightarrow i,1 \rightarrow j}{}^{\mathcal{F}'} + \max\{0, \mathcal{C}'_{\mathcal{I},1 \rightarrow i,p_2 \rightarrow j+1}{}^{\mathcal{F}'}\} + \max\{0, \mathcal{C}'_{\mathcal{I},p_1 \rightarrow i+1,1 \rightarrow j}{}^{\mathcal{F}'}\} + \max\{0, \mathcal{C}'_{\mathcal{I},p_1 \rightarrow i+1,p_2 \rightarrow j+1}{}^{\mathcal{F}'}\}$$

This is exactly the condition as in Equation 5.6 with the cumulative sums replaced by their corresponding refined estimates after Phase II. Unlike the 1-d case (Theorem 5.3), we cannot guarantee the existence of a 2-d window involving (i, j) in which \mathcal{I} is frequent for

all itemsets \mathcal{I} remaining in $\mathcal{F}'_{i,j}$ at the end of Phase II processing. This can be seen from a counter-example in Figure 4. The numbers indicate the credit in each cell for an itemset \mathcal{I} . For $i = 2$ and $j = 2$, the last three components in Equation 5.7 evaluate to 2 each. The shaded areas show the regions which lead to this counts. The first component of the equation evaluates to -6 . Thus the entire equation adds up to 0, which is non-negative; itemset \mathcal{I} will remain in $\mathcal{F}'_{i,j}$ at the end of Phase II. Since the shaded regions are not aligned, they cannot be combined into a bigger 2-d window containing (i, j) in which the itemset is frequent. As can be seen, there is no other 2-d window in which the itemset \mathcal{I} is frequent. This issue of non-aligned sub-windows exists only in 2-d and higher dimensions.

		j			
		-1	-1	1	-1
i	-1	-1	-6	1	-1
	-1	1	1	1	1
	-1	1	-1	-1	-1

Figure 4: Counter-example for 2-d windows.

5.4 Handling multi-dimensional Windows We saw in the previous section that the *left* \rightarrow *right, top* \rightarrow *bottom* count for $[i, j]$ was useful in checking whether an itemset has to be stored in the cell $[i, j]$. On the other hand, the *left* \rightarrow *right, bottom* \rightarrow *top* count for $[i, j]$ would be useful to check whether an itemset has to be stored in the cell $[i-1, j]$. Thus, each combination of end points leads to a count, for every cell, that is useful to consider checks either pertaining to them, or to their neighbors. Thus, for m dimensions, we have to compute 2^m counts for each cell (per itemset); once that is done, the checks are straightforward.

We outline the approach for computing the 2^m cumulative credits in Algorithm 3. Each of the 2^m combinations could be represented as an m -bit integer. For every cell V , $C[V]$ is used to represent the array of counts, and $C[V][B]$ represents the count for a particular combination of endpoints represented by B . For each attribute, the algorithm scans the array C in row-major order with that attribute as the row two times, once in the increasing order (from v_1 to v_{p_1} in lines 4-11) and once in the decreasing order (from v_{p_1} to v_1 in lines 12-19). The array corresponding to each cell $C[V]$ doubles in size in each iteration; at the end of m iterations over all the attributes, each cell $C[V]$ holds 2^m cumulative credits, one for each choice of end-points.

Alg. 3 Computing credits for Phase I

1. for each $V \in \mathcal{V}$
 2. $C[V][0] = \text{credit of } \mathcal{I} \text{ in cell } V$
 3. for i from 1 to m
 4. for each V obtained by scanning C in row-major increasing order of A_i
 5. if $V[i] = 1$ (*lower endpoint*)
 6. for j from 0 to $2^{i-1} - 1$
 7. $C'[V][2 * j] = C[V][j]$
 8. else
 9. for j from 0 to $2^{i-1} - 1$
 10. $C'[V][2 * j] = \max(0, C'[PrevV][2 * j]) + C[V][j]$
 11. $PrevV = V$
 12. for each V obtained by scanning C in row-major decreasing order of A_i
 13. if $V[i] = p_i$ (*upper endpoint*)
 14. for j from 0 to $2^{i-1} - 1$
 15. $C'[V][2 * j + 1] = C[V][j]$
 16. else
 17. for j from 0 to $2^{i-1} - 1$
 18. $C'[V][2 * j + 1] = \max(0, C'[PrevV][2 * j + 1]) + C[V][j]$
 19. $PrevV = V$
 20. $C = C'$
-

We omit the details of the expression to check for non-negativity in both the phases due to lack of space.

6 Query Time Processing

Once the pre-processing has been done to handle m dimensional query windows, we would now be able to handle query-time specifications of windows over upto m attributes. Consider a k -dimensional window; $(A_{i_1}, [x_1, y_1]), (A_{i_2}, [x_2, y_2]), \dots, (A_{i_k}, [x_k, y_k]), k \leq m$. A cell with value combination $V = (v_1, v_2, \dots, v_m)$ would be part of the window if:

$$(x_1 \leq v_{i_1} \leq y_1) \wedge (x_2 \leq v_{i_2} \leq y_2) \dots (x_k \leq v_{i_k} \leq y_k)$$

Let \mathcal{V} be a set of all such value combinations (i.e., cells) that are included in the specified window. Further, for every cell V , let \mathcal{F}'_V be the set of all itemsets and their *credits* (excess frequency counts beyond threshold) retained after pre-processing. Since Phase I forces counting some itemsets despite being infrequent in the cell, the credits held in \mathcal{F}'_V could be negative.

We outline the query time processing in Algorithm 4. The eventual result set \mathcal{R} is initialized to ϕ and then updated to \mathcal{F}'_V when the first cell V from the window is being considered (lines 4-6). Whenever a new cell is considered, \mathcal{R} is set to its intersection with

1. $\mathcal{R} = \phi$
 2. for each $V \in \mathcal{V}$
 3. if ($\mathcal{R} = \phi$)
 4. $\mathcal{R} = \{\mathcal{I} | \mathcal{I} \in \mathcal{F}'_V\}$
 5. for each $\mathcal{I} \in \mathcal{R}$
 6. $\mathcal{I}_{credit} = getCredit(\mathcal{F}'_V, \mathcal{I})$
 7. else
 8. $\mathcal{R} = \mathcal{R} \cap \{\mathcal{I} | \mathcal{I} \in \mathcal{F}'_V\}$
 9. if ($\mathcal{R} = \phi$) break;
 10. for each $\mathcal{I} \in \mathcal{R}$
 11. $\mathcal{I}_{credit} = \mathcal{I}_{credit} + getCredit(\mathcal{F}'_V, \mathcal{I})$
 12. $\mathcal{R} = \{\mathcal{I} | \mathcal{I} \in \mathcal{R} \wedge \mathcal{I}_{credit} \geq 0\}$
-

the corresponding \mathcal{F}'_V ; the credit of itemsets in \mathcal{R} is then updated by adding the credit for those itemsets as obtained from the considered cell (lines 8-11). If the intersection at line 8 leads to an empty set, we break out of the loop, the result set being trivially empty. Finally, all itemsets having negative credits are purged out of \mathcal{R} (line 12) and the remaining are reported as results along with their corresponding counts.

The correctness of this algorithm is based on Theorem 5.2. It guarantees that any itemset that is frequent in the query window will be present in \mathcal{F}'_V for each $V \in \mathcal{V}$. This implies that the intersection of \mathcal{F}'_V s over all $V \in \mathcal{V}$ will contain all the frequent itemsets in the query window. *By way of this pre-processing technique, we are able to eliminate the need to count the support of itemsets at query time.*

Query Time Support Specifications The user may not be always interested in the same support threshold (μ) as used in pre-processing. We consider the case where the user may specify a threshold μ' at query time. When $\mu' < \mu$, we have to fall back on a counting based approach (however, pre-computed counts could be leveraged even in this case; we do not include details due to space constraints) since itemsets that have a support between μ' and μ have no pre-computed information. When $\mu' \geq \mu$, we can easily handle the case since it amounts to filtering the result set for the query with μ further as follows:

$$\mathcal{R} = \{\mathcal{I} | \mathcal{I} \in \mathcal{R} \wedge \mathcal{I}_{credit} \geq S * (\mu' - \mu)\}$$

where S is the number of transactions in the query window being considered (i.e., $S = \sum_{V \in \mathcal{V}} |\mathcal{D}_V|$ where $|\mathcal{D}_V|$ denotes the number of transactions in the cell V).

7 Handling Incremental Database Updates

Changes to such a multi-dimensional structure occur by addition or deletion of transactions in cells. For changes in the set of transactions in cell A , the itemsets whose supports have to be additionally computed come from among the frequent itemsets in A or those whose supports have been stored in A 's adjacent cells already. The same applies to addition of new values at either ends of the existing values, for an attribute (e.g., adding the transactions for the latest month, in the time attribute). This property ensures that the additional pre-computation cost is negligible for such database updates. We omit proofs due to space constraints.

8 Experimental Study

We now empirically compare our proposed approach with TOARM [6] and a naive technique that builds upon the TOARM approach, but does more pre-computation to avoid query time support counting (BL\C).

TOARM: TOARM depends on the observation that an itemset that is frequent in the query context would be frequent in at least one of the component cells of the context. Thus for a cell C_i , it pre-computes its frequent itemsets \mathcal{F}_i . At query time, it first determines the union of sets of frequent itemsets across all component cells, then applies filtering criteria, counts support of the remaining in the context of interest and finally outputs the frequent itemsets. Support counting, though an expensive operation, is necessary since the counts for a candidate itemset would not have been pre-computed for the cells in which it is not frequent.

BL\C: An itemset has to be frequent in at least one cell in the *entire space* under consideration, if it is to be frequent in any query window within the space. A simple way to leverage this observation to avoid query time counting is to pre-compute the actual supports of each such itemset for each of the cells in the entire space, i.e., to pre-computes the counts for $\cup \mathcal{F}_i$ for each cell. We refer to such an approach as BL\C (**B**ase**L**ine approach without **C**ounting); at query time, it computes the actual support in the query context for each itemset, using the pre-computed supports, followed by a simple filtering to arrive at the results.

Evaluation Measures: In an exploratory data analysis setting where trends are analyzed using changing query windows, the most crucial performance measure is response time. We also analyze the different approaches on storage required and pre-processing time.

Experimental Setup: Our experiments were run on an IBM X Series running Windows Server 2003

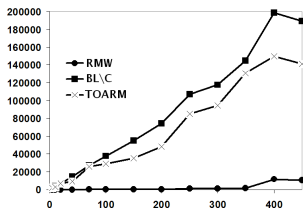


Figure 5: Response Time (ms) vs. Window Length (1- d , 1%)

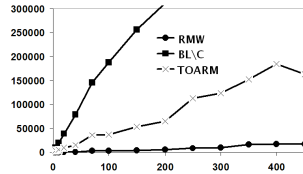


Figure 6: Response Time (ms) vs. Window Length (1- d , 0.7%)

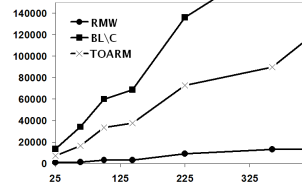


Figure 7: Response Time (ms) vs. Window Size (2- d , 1.0%)

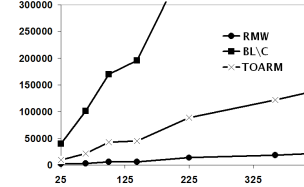


Figure 8: Response Time (ms) vs. Window Size (2- d , 0.7%)

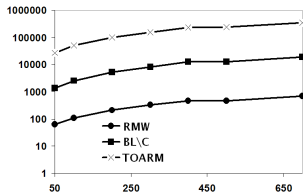


Figure 9: Response Time (ms) vs. Window Size (1- d , 0.6%) (IBM Data)

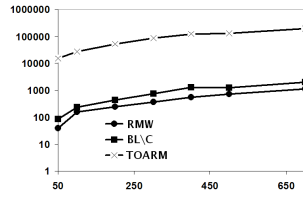


Figure 10: Response Time (ms) vs. Window Size (1- d , 1.0%) (IBM Data)

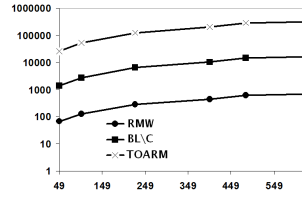


Figure 11: Response Time (ms) vs. Window Size (2- d , 0.6%) (IBM Data)

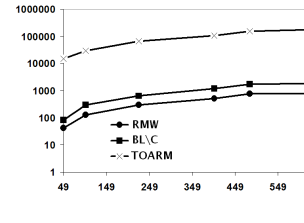


Figure 12: Response Time (ms) vs. Window Size (2- d , 1.0%) (IBM Data)

on an Intel Xeon 2.33GHz processor with 3.25GB of RAM. Pre-computed information is stored as one file per cell on disk; response time includes the time to read relevant files. *BL/C* and *RMW* require reading only as many files with pre-computed results as the query window length. *TOARM*, however, needs to read original transaction files too since it has to do counting of non-filtered candidates. For each experiment, we analyze the response time as averaged over 10 random queries whose lengths come from a normal distribution with the chosen window length as the mean and 20% of it as the standard deviation.

Datasets Real Data: The BMS-POS dataset¹ has 515,597 transactions; we split it into 515 cells of 1k transactions each with the last 597 transactions in the 516th cell for our experiments with 1- d query windows (simulating an attribute with 516 distinct values). We had to simulate attributes since the BMS-POS dataset is composed purely of transactions and does not have associated attributes. For 2- d windows, we take the first 500k transactions and create 500 cells of 1k transactions each, arranged in a grid of 25X20 (simulating two attributes with 25 and 20 distinct values).

Synthetic Data: The IBM Market Basket Data generator generates data according to parameters such as the size of the vocabulary of itemsets. They can be var-

ied across various runs to generate data of varying characteristics across cells. To analyze the changing behavior of the techniques with varying nature of transactions across cells, we create two sets of synthetic data, each containing 1024k transactions. For 1- d , we partition the dataset into 1024 cells with 1k transactions each, thus simulating an attribute with 1024 distinct values. For the 2- d scenario, we simulate two attributes with 32 distinct values each by arranging cells in a 32X32 grid.

Homogeneous Data: For the homogeneous data, we generate 1024k transactions with an average of 10 items per transactions, items having ids in the range 1 to 2000.

Heterogeneous Data: Here, we generate 1024k transactions in chunks of 1k transactions. While keeping the average length of transactions at 10, we vary the item ids across these chunks. For the i^{th} chunk, we generate transactions containing items having ids in the range $[(i-1) * D + 1, (i-1) * D + 2000]$; the first cell would contain transactions having item ids from $[1, 2000]$ whereas the 6th chunk has transactions with item ids from the range $[51, 2050]$ for $D = 10$. These chunks are preserved while doing the partitioning in the grid. In the 1- d case, the i^{th} chunk would form the i^{th} cell in the grid. These chunks are arranged in row-major fashion on the 2- d 32X32 grid.

8.1 Varying Window Lengths We now analyze response times of the various techniques against varying query window sizes. We measure the sizes of query

¹<http://fimi.cs.helsinki.fi/data/>

windows as the number of cells included within it. Thus, a $2-d$ window that covers 5 values from one dimension (say, $v_2 - v_7$) and 7 values from the other would have a size of 35 since it covers 35 cells. Since the support percentage is another source of variation, we fix the support at different values and analyze the response time behavior. For a given size, we choose the window from a normal distribution with the mean as the chosen mean and the standard deviation as 20% of it.

BMS-POS Data: For the BMS-POS dataset, we vary the $1-d$ window size from 5 to 450 fixing supports at 0.7% and 1.0%; the charts appear in Figures 6 and 5 respectively. Response times deteriorate with increasing window sizes (more cells need to be examined) and decreasing support (more frequent itemsets to consider). The charts show that RMW scales remarkably well with increasing window sizes providing response times of 10s across varying window sizes and supports. This is in sharp contrast to TOARM and BL\C that sharply deteriorate with increasing window sizes. TOARM has to do count supports of a larger number of itemsets over larger number of cells when query window sizes become larger. BL\C, on the other hand, has to aggregate counts over more cells, and thus varies linearly with window size. The charts for varying $2-d$ window sizes fixing supports at 0.7% and 1.0% respectively appear in Figures 8 and 7. Observations are similar, with *RMW outperforming other approaches by close to an order of magnitude across varying window sizes and supports.*

IBM Homogeneous Dataset: Homogeneity causes more frequent itemsets to be common across cells (thus, reducing the size of the union across cells), a favorable case for BL\C. TOARM is also benefited since it has fewer itemsets to consider counting of; this benefit is often offset by the reduced effectiveness of TOARM query-time pruning conditions. Homogeneous data presents an unfavorable case for RMW since pruning effectiveness is reduced due to many common frequent itemsets across cells. The reduced pruning at pre-processing time leaves many counts in the storage, to be considered at query time. Figures 9 and 10 plot the response time behavior (in *log-scale*) over varying $1-d$ window sizes at supports of 0.6% and 1.0% respectively on the homogeneous dataset. RMW remains a clear winner even in this setting that is unfavorable to it; it is seen to respond upto an order of magnitude faster at 0.6% support and upto twice as faster at 1.0% support. Unlike the observations in the BMS-POS dataset, BL\C outperforms TOARM on homogeneous data; this is due to the larger number of shared frequent itemsets across cells in homogeneous data (thus, leading to a smaller union of frequent itemsets across cells). Observations on $2-d$ windows are not very different from those

in $1-d$; charts for supports of 0.6% and 1.0% appear in Figures 11 and 12 respectively.

IBM Heterogeneous Dataset: Heterogeneity in the nature of transactions across cells is favorable for the RMW approach since better pruning can be affected due to fewer common itemsets across cells. We generate heterogeneous data using the IBM data generator by varying the space of itemset IDs across cells, D being a parameter larger values of which cause increased heterogeneity in generated data. Transactions in each cell is generated from a vocabulary of items, the vocabulary differing by D items between adjacent cells. We plot the response times across varying $1-d$ window sizes for a support of 1% fixing D at 10 (Refer Figure 13) and 50 (Figure 14) respectively. As expected (heterogeneous data being a favorable case for RMW), RMW responds many times faster than BL\C and TOARM approaches. The variation in behavior of TOARM with varying window sizes is worth delving deeper into; at $D=10$, when the window size in $1-d$ becomes large enough, the pruning in TOARM is efficient since there are very few itemsets in common due to inclusion of distant cells thus leading to faster responses. At $D=50$, such effects become apparent at smaller window sizes, as expected. The margins by which RMW outperforms the other approaches were seen to improve with decreasing supports; we omit charts due to lack of space.

Thus, RMW is seen to outperform both TOARM and BL\C across widely varying window sizes under different supports and heterogeneity. RMW scales very well with increasing window sizes; this underlines the effectiveness of the RMW pruning conditions.

8.2 Varying Supports We now analyze the response latency against varying support specifications on the BMS-POS dataset. Since the number of frequent itemsets typically increase tremendously even when support is reduced slightly, higher supports lead to faster response times (and smaller result sets). Figure 15 plots the response times for the BMS-POS dataset against varying supports on a fixed window size of 100. BL\C is not included in the plot due to its extremely slow responses. The trend against varying support is seen to be similar for both TOARM and RMW. In absolute terms, however, RMW is seen to respond to queries 7 times faster than TOARM. A similar observation holds for $2-d$ windows too (Figure 16). Observations on the IBM datasets also showed similar trends against varying supports; we omit the charts for brevity.

8.3 Varying Heterogeneity Heterogeneity in transactions affects various techniques in different

ways. Homogeneity is seen to favor BL\C whereas TOARM and RMW can exploit heterogeneity well. Such expected effects are pronounced well in the behavior of TOARM and BL\C in the case of the IBM heterogeneous dataset (against varying levels of heterogeneity) as shown in Figure 17. RMW, once again, consistently outperforms the other techniques across varying heterogeneity due to the effectiveness of pruning conditions at pre-processing time.

8.4 Storage Analysis TOARM stores just the counts of frequent itemsets for each cell whereas BL\C stores the counts of each itemset that is frequent in at least one cell, across all cells. RMW selectively stores additional counts apart from those of TOARM, thus incurring storage costs higher than TOARM, but lesser than BL\C. The storage costs for the BMS-POS dataset across varying support specifications are plotted in Figure 18. BL\C is seen to incur very high storage costs whereas RMW requires only 40% and 65% more storage than TOARM in 1- d and 2- d respectively. In absolute terms, the storage requirement for RMW is moderate (between 65MB and 200MB) across varying supports.

Memory Usage: *As we consider one frequent itemset at a time during RMW pre-processing (regardless of number of dimensions), the memory usage for pre-computation is bounded by the grid size (i.e., number of cells in the space) when an external algorithm is used to compute the union of frequent itemsets.*

8.5 Pre-processing Times The relative pre-processing time trends are not identical to their storage requirement since RMW omits storing the supports of those itemsets that are pruned despite having counted their supports (i.e., despite incurring computational costs for them). From Figure 19, BL\C is seen to take upto 100 minutes whereas TOARM takes less than 10 minutes. RMW takes upto twice as much time as TOARM in the 1- d case and thrice as much in the 2- d case. This is not surprising since the increased flexibility in choosing 2- d regions leads to lesser pruning in 2- d . Observed RMW pre-processing times are easily tolerable for typical applications.

8.6 Scalability with Number of Dimensions In the multi-dimensional RMW approach as outlined in Section 5.4, we would need to compute 2^m counts for each combination of itemset and cell. Thus, the pre-computation is exponential in the number of dimensions, thus limiting scalability. Further, the sub-optimality of pruning illustrated in Figure 4 is aggravated with increasing number of dimensions, due to more windows that each cell can be part of; at extremely

high number of dimensions, this could cause it to degrade to BL\C. Analyses of the kind RMW seeks to enable in real-time are often useful only over a subset of metadata attributes (e.g., time, place of transaction). In particular, such analyses may not be useful over attributes like customer id and mode of payment. In such cases, RMW could be enabled only over attributes like the former, making scalability less of a concern. In cases where such analyses need to be performed over many attributes, a better approach would have to be devised.

9 Conclusions

Interactive association rule mining over multi-dimensional query windows is a challenging problem. Existing methods such as TOARM require support counting for some itemsets at query time, leading to higher response time. The extreme approach of pre-computing supports for all itemsets across all cells (BL\C) avoids going back to the transaction database for counting at run time, at the cost of increased storage and computation cost. To address these issues, we propose the *RMW* method that determines the minimal set of itemsets to compute and store for each cell, so that mining over any user query may be performed without revisiting the transaction database. We prove that the set of itemsets chosen by RMW is sufficient to answer any query as well as the optimality of the set of itemsets stored for the 1- d case. We illustrate the effectiveness of RMW over TOARM and BL\C methods through experiments on real and synthetic data. Our results show that RMW outperforms both TOARM and BL\C by several factors across varying data distributions, query windows and supports, keeping the pre-computation cost and extra storage moderate. The pre-computation cost for RMW increases with the number of dimensions. Typically, only a subset of dimensions are useful for such ad-hoc window based rule mining. RMW pre-computation can be done only on this subset rather than all the dimensions, to reduce pre-computation costs.

References

- [1] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD*, 1993.
- [2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994.
- [3] J. Han and Y. Fu, "Discovery of multiple-level association rules from large databases," in *VLDB*, 1995.
- [4] R. Srikant and R. Agrawal, "Mining generalized association rules," in *VLDB*, 1995, pp. 407–419.
- [5] J. Han, L. V. S. Lakshmanan, and R. T. Ng,

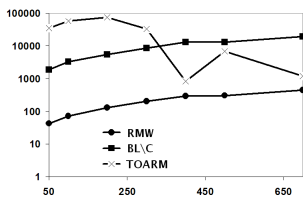


Figure 13: Response Time (ms) vs. Window Size (1-d, $D=10$) (IBM Data)

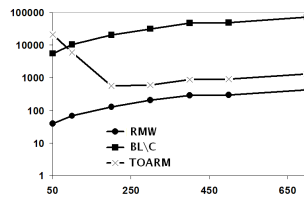


Figure 14: Response Time (ms) vs. Window Size (1-d, $D=50$) (IBM Data)

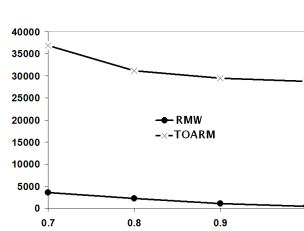


Figure 15: Response Time (ms) vs. Support% (1-d, Window Size = 100)

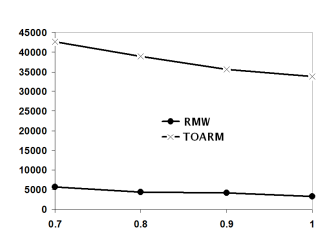


Figure 16: Response Time (ms) vs. Support% (2-d, Window Size = 100)

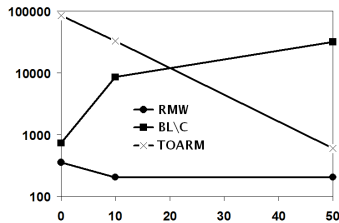


Figure 17: Response Time (ms) vs. D (1d, Window = 300) (IBM Data)

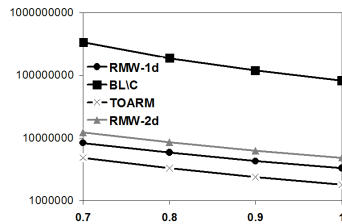


Figure 18: Storage vs. Support

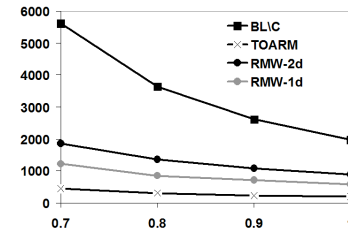


Figure 19: Pre-processing Times (in secs)

- “Constraint-based multidimensional data mining,” *IEEE Computer*, 1999.
- [6] C.-Y. Wang, S.-S. Tseng, and T.-P. Hong, “Flexible online association rule mining based on multidimensional pattern relations,” *Inf. Sci.*, vol. 176, no. 12, pp. 1752–1780, 2006.
 - [7] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining frequent patterns without candidate generation: A frequent-pattern tree approach,” *Data Min. Knowl. Discov.*, vol. 8, no. 1, pp. 53–87, 2004.
 - [8] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, “Dynamic itemset counting and implication rules for market basket data,” in *SIGMOD*, 1997.
 - [9] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in *SIGMOD*, 2000.
 - [10] R. J. B. Jr., R. Agrawal, and D. Gunopulos, “Constraint-based rule mining in large, dense databases,” in *ICDE*, 1999.
 - [11] M.-S. Chen, J. Han, and P. S. Yu, “Data mining: An overview from a database perspective,” *IEEE TKDE*, 1996.
 - [12] D. W.-L. Cheung, J. Han, V. T. Y. Ng, and C. Y. Wong, “Maintenance of discovered association rules in large databases: An incremental updating technique,” in *ICDE*, 1996.
 - [13] D. W.-L. Cheung, S. D. Lee, and B. Kao, “A general incremental technique for maintaining discovered association rules,” in *DASFAA*, 1997, pp. 185–194.
 - [14] P. S. M. Tsai, C.-C. Lee, and A. L. P. Chen, “An efficient approach for incremental association rule mining,” in *PAKDD*, 1999.
 - [15] G. S. Manku and R. Motwani, “Approximate frequency counts over data streams,” in *In VLDB*, 2002.
 - [16] C. Hidber, “Online association rule mining,” in *SIGMOD*, 1999.
 - [17] C. C. Aggarwal and P. S. Yu, “A new approach to online generation of association rules,” *IEEE TKDE*, 2001.
 - [18] B. Nag, P. Deshpande, and D. J. DeWitt, “Using a knowledge cache for interactive discovery of association rules,” in *KDD*, 1999.
 - [19] M. Kamber, J. Han, and J. Chiang, “Metarule-guided mining of multi-dimensional association rules using data cubes,” in *KDD*, 1997, pp. 207–210.
 - [20] H. Zhu, “On-line analytical mining of association rules,” Master’s Thesis, Simon Fraser University, 1998.
 - [21] T. Imielinski, L. Khachiyan, and A. Abdulghani, “Cubegrades: Generalizing association rules,” *DMKD*, vol. 6, no. 3, pp. 219–257, 2002.
 - [22] B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan, “Prediction cubes,” in *VLDB*, 2005, pp. 982–993.
 - [23] R. B. Messaoud, S. L. Rabaséda, O. Boussaid, and R. Missaoui, “Enhanced mining of association rules from data cubes,” in *DOLAP*, 2006, pp. 11–18.
 - [24] V. Harinarayan, A. Rajaraman, and J. D. Ullman, “Implementing data cubes efficiently,” in *SIGMOD*, 1996.
 - [25] A. Shukla, P. Deshpande, and J. F. Naughton, “Materialized view selection for multidimensional datasets,” in *VLDB*, 1998.
 - [26] C. C. Aggarwal and P. S. Yu, “Online generation of association rules,” in *ICDE*, 1998, pp. 402–411.