PARAS: A Parameter Space Framework for Online Association Mining*

Xika Lin†, Abhishek Mukherji†, Elke A. Rundensteiner , Carolina Ruiz , Matthew O. Ward
Computer Science Department, Worcester Polytechnic Institute
100 Institute Road, Worcester MA, USA.
xika|mukherab|rundenst|ruiz|matt@wpi.edu

ABSTRACT

Association rule mining is known to be computationally intensive, yet real-time decision-making applications are increasingly intolerant to delays. In this paper, we introduce the *parameter space model*, called **PARAS**. **PARAS** enables efficient rule mining by compactly maintaining the final rulesets. The **PARAS** model is based on the notion of *stable region abstractions* that form the *coarse granularity ruleset space*. Based on new insights on the redundancy relationships among rules, **PARAS** establishes a surprisingly compact representation of complex *redundancy relationships* while enabling efficient redundancy resolution at query-time. Besides the classical rule mining requests, the **PARAS** model supports three novel classes of exploratory queries. Using the proposed **PSpace index**, these exploratory query classes can all be answered with near real-time responsiveness. Our experimental evaluation using several *benchmark* datasets demonstrates that **PARAS** achieves 2 to 5 orders of magnitude improvement over state-of-the-art approaches in online association rule mining.

1. INTRODUCTION

1.1 Motivation

Data mining is extensively used by analysts in applications from market basket analysis [2, 9], web usage mining [14], census analysis [5], intrusion detection to bioinformatics [18]. It is well-known that data mining algorithms are compute-intensive and parameterized. Analysts from diverse domains ranging retail, stock trading to biology utilize state-of-the-art interactive data visualization and exploratory systems such as [20] for analyzing data sets and exploring interesting patterns such as association rules. For example, by analyzing online transaction logs a retail analyst at amazon may identify products that are frequently purchased together. Such products can be placed together on amazon.com, made into bundled offers or used for recommendations when users search for one of the products. Another example is a trader who wants to trace the live stock market trends to find associations between the ticker symbols (e.g., GOOG) and volumes of stocks traded in transactions for the past hour. Below, we discuss three critical properties of an *interactive* mining system.

Motivation 1: Fast turnaround times of mining queries. An analyst submits an association rule mining request with parameters such as *minsupp*, *minconf* or other interestingness measures. Existing online mining techniques prestore frequent itemsets [1] in an itemset-based index such as the *adjacency lattice* (Figure 1). At query-time, the lattice is leveraged to iteratively generate the association rules. At each iteration, the support and confidence of the newly generated rule must be compared with the query parameters *minsupp* and *minconf*. Despite utilizing the precomputed *intermediate* itemsets, response times for mining rules for even moderately sized datasets ($\sim$ 200 MB) tend to take hundreds of seconds or more [15, 16]. This is not within the response time range expected for interactive systems - thus risks losing a user's attention during the process. In mission critical applications, this delay in decision making may be detrimental.

Motivation 2: Recommendations for parameter settings. While mining queries are parameterized, an analyst may not know the precise parameter settings that would result in the desired number and types of interesting rules in a single trial. Numerous successive trial-and-error interactions using different query parameter values may be required to get satisfactory results. Unfortunately, existing mining models tend to be blackboxes that provide no knowledge of precise parameter settings to match the analyst's interest. A mining system capable of making recommendations for parameter tuning and helping the analysts extract desired associations using significantly fewer trial-and-error cycles would offer a competitive advantage to the analysts.

Motivation 3: Support for parameter space exploration. Existing mining systems simply output the rules satisfying the user's requests. However, beyond receiving such output, analysts may benefit from understanding the relationships among the different rules (such as rules sharing common items) or the distribution of these rules in the space of query parameters. Further, if some parameter settings were to produce a huge number of associations, this may overwhelm the analyst. Thus knowledge of redundancies among associations [1] would be useful for reducing the size of the output. Unfortunately, most existing systems do not provide such useful insights about a target dataset. Instead, the analysts have to perform manual inspections to make sense of the results.

1.2 The State-of-the-Art

Online rule mining techniques prestore frequent itemsets [1] to overcome the prohibitive costs of running the mining algorithm at query-time. Aggarwal et al. [1] and follow-on work [11, 12] use the principle of *preprocess once−query many (POQM)* only partially

†Authors contributed equally to this work.

*This work was partly supported by the National Science Foundation under grants IS-0812027 and CCF-0811510.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy.
Proceedings of the VLDB Endowment, Vol. 6, No. 3
Copyright 2013 VLDB Endowment 2150-8097/13/01... $10.00.

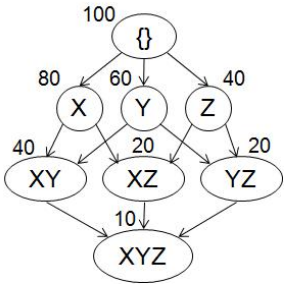


Figure 1: Adjacency Lattice.

Rule	Support	Confidence
X⇒YZ	S(X ∪ Y ∪ Z) = 0.1	S(X ∪ Y ∪ Z) / S(X) = 0.125
XY⇒Z	S(X ∪ Y ∪ Z) = 0.1	S(X ∪ Y ∪ Z) / S(X ∪ Y) = 0.25
XZ⇒Y	S(X ∪ Y ∪ Z) = 0.1	S(X ∪ Y ∪ Z) / S(X ∪ Z) = 0.5
X⇒Y	S(X ∪ Y) = 0.4	S(X ∪ Y) / S(X) = 0.5
X⇒Z	S(X ∪ Z) = 0.2	S(X ∪ Z) / S(X) = 0.25

Figure 2: Redundancy in Rules.

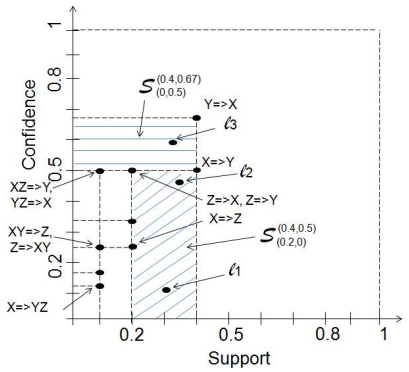


Figure 3: Parameter Space.

as they precompute the *intermediate itemsets* in an itemset-based index (adjacency lattice in Fig. 1). However, unfortunately *rule generation* remains a query-time task. Worst yet, at query-time, such systems must not only first generate the rules but also filter out rules that are redundant to produce succinct query results [1]. This *combined generate and filter* online step tends to render the turnaround times unacceptable, particularly, for a dataset with millions of items. While Sahar [17] and Han et al. [19] identify the importance of analyzing the interestingness measures of rules, they do not tackle online mining through precomputation. In contrast, we take the notion of POQM with respect to online rule mining to the next level by achieving effective materialization offering truly online performance.

1.3 Research Challenges

For developing an interactive rule mining system, the following research challenges must be tackled:

Managing large number of rules. In general, for k items in a dataset $\mathcal{D}$, a total of 2^k-1 itemsets can be composed [8]. Clearly, for a dataset with several hundreds or thousands of items, it is prohibitively expensive to store all rules individually. This raises two critical requirements for rendering the precomputation of final rulesets practical. First, we must design a data structure to compactly prestore rulesets. Next, we must develop an efficient strategy that uses this compact storage at query-time for processing online mining requests.

Managing rules for all parameter values. To facilitate a direct lookup of rulesets given input parameter values, the rules must be indexed on the two-dimensional space of *support* and *confidence* parameters. Yet *minsupp* and *minconf* values input by users are from a continuous domain [0,1]. For a large dataset, prestoring rules for all possible parameter settings will be extremely challenging. On the other hand, some *regions* of the parameter space may be *stable*, such that despite changes in the input parameter settings, the output ruleset remains unchanged. This raises the need for effective *stable region*-aware indexing of rules within the parameter space.

Resolving rule redundancies at query-time. If too many rules are returned to the user, including redundant ones (Section 2.2), this clutter prevents the user's understanding. Hidden in the clutter, interesting patterns may go unnoticed. While query-time redundancy resolution [1] must be supported for interactive mining, its known prohibitive expense must be avoided at run-time. To further complicate matters, *redundancy is a query-time phenomenon* (see Section 5), that is, rules cannot be deemed redundant and thus eliminated in an offline step. Instead, redundancies can only be resolved at query-time based on the context of the complete output ruleset. Such online redundancy resolution requires $\mathcal{O}(2^n)$ time where n is the number of prestored rules.

Avoiding repeated storage of rules during preprocessing. For prestoring rulesets, we utilize a multi-dimensional parameter space with support and confidence as its dimensions. If a rule $\mathcal{R}_i$ maps to a location ($\mathcal{R}_i.supp$, $\mathcal{R}_i.conf$) of this space, then $\mathcal{R}_i$ belongs in the output for every user input (*minsupp*, *minconf*) such that *minsupp* $\leq \mathcal{R}_i.supp$ and *minconf* $\leq \mathcal{R}_i.conf$. This leads to conflicting demands that the compact structure must not store rules repeatedly, yet at query-time non-redundant rules may be efficiently assembled for any setting by collecting all such rules.

1.4 Running Example of Parameter Space

Let us examine the adjacency lattice for a dataset $\mathcal{D}$ depicting 100 records and three items X, Y and Z (Fig. 1). Each itemset node $\mathcal{N}_i$ has a support count (e.g., S(X) = 80). To motivate the use of a two-dimensional parameter space with *support* on x-axis and *confidence* on y-axis for organizing rulesets, Fig. 3 plots the rules generated from the adjacency lattice (Figure 1). For example, rule (Y ⇒ X) maps to (0.4,0.67) and rules (XZ ⇒ Y) and (YZ ⇒ X) both map to location (0.1,0.5). Many rules map to the same (support, confidence) location, leading to a more compact structure than simply indexing the independent rules.

Figure 3 contains the shaded regions $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$ and $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$. No new rules would be produced even if parameter values were to be adjusted by the users within eac of these regions. We call them *stable regions*. An example of *repeating rules* is (Y ⇒ X) that appears for the first time in stable region $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$ but gets repeatedly output for all requests with *minsupp* $\leq$ 0.4 and *minconf* $\leq$ 0.67 as it is valid between (0,0) and (0.4,0.67).

1.5 The PARAS Approach

Based on the above observations, we introduce a *parameter space model*, consisting of *support* and *confidence* axes[1], for support of online association mining. The intuition is that a parameterized index can be directly searched for association rules by matching the query parameters with the index key. Instead of prestoring only the itemsets as commonly done by [1, 11, 12], we prestore the actual *rules*. Thus, the online rule generation step is altogether avoided. In addition to achieving speedup in online mining, our **PARAmeter Space Model for Association Mining** (**PARAS**)[2] framework provides support for novel exploratory queries (Sec. 3.1). For interactive analytics, such as recommendations for query parameter selection. **PARAS** also gives rich insights into the redundancy relationships of association rules within the space. Our **PARAS** framework

[1]While support and confidence are popular measures, the parameter space can also use other measures, such as *lift* and *conviction*.
[2]Hindi name for a legendary philosopher's stone said to be capable of turning base metals (lead, for example) into gold.

is complemented by efficient algorithms for both offline preprocessing and online query processing.

1.6 Contributions

The contributions of this work are as follows:

• We propose the parameter space model, called **PARAS**, that organizes association rules in a space of query parameters. Instead of maintaining the huge number of individual rules in an infinite parameter space, our compact space representation, called **PSpace**, abstracts rulesets at a coarse granularity of stable regions (Sec. 2).

• We observe that redundancy is a query-time phenomenon (Section 2). To overcome this, we establish a theoretical foundation of redundancy relationships among associations that allows us to precompute compact abstractions of redundancy. This abstracted redundancy meta-knowledge enables effective *redundancy resolution* at query-time. For N rules in the output ruleset, we reduce the complexity of a naive online redundancy resolution [1] from $\mathcal{O}(2^N)$ to online redundancy resolution that takes $\mathcal{O}$(N) time by performing a $\mathcal{O}(N^2)$ time offline redundancy abstraction step (Sec. 5).

• Our **PARAS** framework (Section 3) offers efficient algorithms for offline **PSpace** index construction. **PARAS** provides *stable region*-aware indexing where each rule is stored exactly once and stable regions are compactly stored in a region neighborship graph (Sec. 4).

• **PARAS** supports a rich set of novel *exploratory mining queries* beyond traditional rule mining. Effective strategies for online processing of these novel query types using the **PSpace** index are also developed (Sec. 6).

• Our extensive experiments using IBM Quest [2], webdocs [14] and other benchmark datasets demonstrate that **PARAS** achieves 2 to 5 orders of magnitude improvement over commonly used techniques in online rule mining (Sec. 7).

2. FOUNDATION OF THE PARAS MODEL

2.1 Parameter Space Model

We use the *parameter space* as a model for managing, retrieving, and exploring the *association rules* of a dataset. For a dataset $\mathcal{D}$, this space contains the rules of $\mathcal{D}$, denoted by $\{\mathcal{R}\}^{\mathcal{D}}$. Each dimension p_k of the space represents an interestingness measure, such as support, confidence and lift [19].

Definition 1. ***Parameter Space:*** *Given a dataset $\mathcal{D}$ and the user-chosen d interestingness parameters each denoted as p_i, we use a d-dimensional parameter space, denoted by $\mathcal{P} = \{p_1, \ldots, p_k, \ldots, p_d\}$ for organizing the rules $\{\mathcal{R}\}^{\mathcal{D}}$ for $\mathcal{D}$. Each rule $\mathcal{R}_j$ is represented by its* ***parametric location*** *($\mathcal{R}_j$.value(p_1), …, $\mathcal{R}_j$.value(p_k), …, $\mathcal{R}_j$.value(p_d)) where $\mathcal{R}_j$.value(p_i) denotes the value of the i^{th} parameter for rule $\mathcal{R}_j$.*

For simplicity, we henceforth work with a two-dimensional parameter space using *support* and *confidence* as dimensions. A *parametric location* ℓ_1 is defined by a combination of support and confidence values, denoted by (ℓ_1.supp, ℓ_1.conf) (Fig. 3). Many association rules may map to the same *parametric location*, e.g., (XZ ⇒Y) and (YZ ⇒X) both map to (0.1,0.5). Therefore, all rules mapping to the same parametric location can be compactly indexed in our parameter space model.

The user specified *minsupp* and *minconf* values may range between 0 and 1, making the number of locations infinitely large. Yet, the number of distinct *parametric locations* is typically much smaller than the actual number of association rules. Thus, if the association rules are grouped by their *parametric locations*, there is a modest number of such locations compared to the number of actual rules (Section 7.3).

In Figure 3, we observe that certain regions of the parameter space often either contain no rules or contain the same set across a large range of parameter settings (e.g., the shaded regions marked $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$ and $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$). This leads us to the notion of stable regions, which forms our *coarse granularity abstractions* for rules.

Definition 2. ***Stable Region:*** *Given a parameter space $\mathcal{P}$ of d interestingness dimensions $\{p_1, \ldots, p_d\}$ for dataset $\mathcal{D}$, a* ***stable region*** *$\mathcal{S}^{(upper(p_1),\ldots,upper(p_d))}_{(lower(p_1),\ldots,lower(p_d))}$ is a d-dimensional rectangular hyperbox with extreme points ($\mathcal{S}$.lower(p_1),…, $\mathcal{S}$.lower(p_d)) and ($\mathcal{S}$.upper(p_1), …, $\mathcal{S}$.upper(p_d)) within which no matter how the parameter values are adjusted, the set of rules generated from $\mathcal{D}$ remains unchanged.*

In Figure 3, the shaded region $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$ is bounded by the parametric locations (0.2,0) and (0.4,0.5) as its extreme points. Let us suppose that a user inputs two separate queries Q_1 and Q_2 with the parametric locations ℓ_1 and ℓ_2, respectively. Then both parametric locations lie within $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$. Thus, we can infer that the outputs for both Q_1 and Q_2 will be the same, i.e., $\{R\}^{Q_1} = \{R\}^{Q_2}$ = {(X ⇒ Y), (Y ⇒ X)}. However, if the user inputs another query Q_3 with parametric location ℓ_3, that lies within region $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$), the output would only contain rule (Y ⇒ X). Thus, the output remains unchanged as long as parameter values are chosen within the bounds of a stable region, whereas the output changes when crossing between stable regions. Next, Lemma 2.1 establishes a neighborhood relationship among adjacent stable regions.

Lemma 2.1. *Consider two stable regions $S^{(usupp1,uconf1)}_{(lsup1,lconf1)}$ and $S^{(usupp2,uconf2)}_{(lsup2,lconf2)}$ such that usupp2 ≥ usupp1 and uconf2 ≥ uconf1, then all association rules valid in stable region $S^{(usupp2,uconf2)}_{(lsup2,lconf2)}$ are also valid in region $S^{(usupp1,uconf1)}_{(lsup1,lconf1)}$. The reverse is not true.*

PROOF. A rule $\mathcal{R}_i$ at a parametric location ($\mathcal{R}_i$.*supp*,$\mathcal{R}_i$.*conf*) qualifies to be output for any mining request with *minsupp* ≤ $\mathcal{R}$.*supp* and *minconf* ≤ $\mathcal{R}_i$.*conf*. Assuming rule $\mathcal{R}_i$ belongs to stable region $S^{(usupp2,uconf2)}_{(lsup2,lconf2)}$, $\mathcal{R}_i$.*supp* = *usupp2* and confidence $\mathcal{R}_i$.*conf* = *uconf2*. Therefore, $\mathcal{R}_i$ will also be valid in region $S^{(usupp1,uconf1)}_{(lsup1,lconf1)}$ as *usupp1* ≤ $\mathcal{R}_i$.*supp* and *uconf1* ≤ $\mathcal{R}_i$.*conf*.

Definition 3. ***Lending Neighbor Stable Region:*** *Consider two neighbor stable regions $S^{(usupp1,uconf1)}_{(lsup1,lconf1)}$ and $S^{(usupp2,uconf2)}_{(lsup2,lconf2)}$, related by Lemma 2.1. We then say that $S^{(usupp2,uconf2)}_{(lsup2,lconf2)}$ is the* ***lending neighbor stable region*** *for $S^{(usupp1,uconf1)}_{(lsup1,lconf1)}$, in short, l-neighbor.*

By Def. 3, in Figure 3 $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$ is the *lending neighbor stable region* for $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$. $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$ lends rule (Y ⇒ X) to $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$, as (Y ⇒ X) first appears in $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$ and is also valid for $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$. We refer to the *lending neighbor stable region(s)* as *l-neighbors* in short.

We partition the *parameter space* $\mathcal{P}$ into a finite number of *non-overlapping stable regions*, denoted by $\{\mathcal{S}\}$. The non-overlapping property among stable regions can be guaranteed due to our approach of modeling the regions (Section 5.3). Utilizing the concept of *neighbor stable regions* (Def. 3), for each such *stable region*, we maintain (a.) the rules that are valid within that region and (b.) the links to its *l-neighbors*. As we demonstrate in Section 6, this partitioning of the parameter space into stable regions enables us to process novel exploratory online queries as well as to recommend query parameter settings based on user interest (Section 6).

2.2 Redundancy in Association Rules

Aggarwal et al. [1] define redundancy relationships among rules, such that redundant rules may be filtered out for presenting succinct query results to the user. The redundant rules could always be derived on demand, if so desired. We examine how these redundancy relationships can be identified in the parameter space model. In particular, redundancy can be of two types [1], as defined below.

Definition 4. ***Simple Redundancy:*** *Let $A \Rightarrow B$ and $C \Rightarrow D$ be two rules such that the itemsets A, B, C and D satisfy the condition $A \cup B = C \cup D$. The rule $C \Rightarrow D$ is* ***simply redundant*** *with respect to the rule $A \Rightarrow B$, if $C \supset A$.*

Definition 5. ***Strict Redundancy:*** *Let $A \Rightarrow B$ and $C \Rightarrow D$ be two rules generated from itemsets X_i and X_j such that $X_i \supset X_j$, $A \cup B = X_i$, $C \cup D = X_j$, and $C \supseteq A$. Then the rule $C \Rightarrow D$ is* ***strictly redundant*** *with respect to the rule $A \Rightarrow B$.*

The concept of redundancy can be illustrated using the rules generated from the lattice (Figure 1) as listed in Figure 2. Based on Definitions 4 and 5, if a rule $\mathcal{R}_1$ is *simple* or *strict redundant* with respect to another rule $\mathcal{R}_2$, then $\mathcal{R}_2$ is said to *simple* or *strict dominate* $\mathcal{R}_1$, respectively. In Figure 2, the rule (X $\Rightarrow$ YZ) *simple dominates* the rules (XY $\Rightarrow$ Z) and (XZ $\Rightarrow$ Y) (Def. 4). In Figure 2, the rule (X $\Rightarrow$ YZ) *strict dominates* rules (X $\Rightarrow$ Y) and (X $\Rightarrow$ Z) (Def. 5). In general, a rule may be dominated by several *dominating rules* and may in turn dominate several other *dominated rules*.

3. THE PARAS SYSTEM OVERVIEW

3.1 Supported Queries

Our **PARAS** framework supports a rich variety of classes of analytical queries using the **PSpace index** $\mathcal{P}$.

Rule Mining (RM) Query: Q1 is a rule mining query that, given dataset $\mathcal{D}$, finds a set of rules satisfying query parameters (*minsupp*,*minconf*). The Using clause with a Boolean input *REFlag* gives users the option to output only non-redundant rules, while the default value is *FALSE*.

```
Q1:  OUTPUT RuleSet {R}^(minsupp,minconf)
FROM D
WITH minsupport=minsupp AND minconfidence=minconf
USING REFlag = T/F;
```

Stable Region (SR) Queries: This query provides meta information about the *stable regions* of $\mathcal{P}$. Query Q2 identifies the stable region containing the user-chosen (*minsupp*,*minconf*) input. The returned region $\mathcal{S}_{(lsupp,lconf)}^{(usupp,uconf)}$ is such that (*usupp* $\geq$ *minsupp* $\geq$ *lsupp*) and (*uconf* $\geq$ *minconf* $\geq$ *lconf*). By Definition 2, no change in the generated ruleset will be achieved by the user as long as different parameter settings are chosen from within this stable region. For a change in the output ruleset, the user will thus be instructed to select parameters with either *minsupp*, *minconf*, or both outside the bounds of region $\mathcal{S}_{(lsupp,lconf)}^{(usupp,uconf)}$, for further exploration.

Example 1: Caught in a large stable region: *For a sparse dataset such as T100k [2], often despite submitting several successive mining requests with distinct (minsupp,minconf) input parameter values, the system repeatedly returns the same set of rules $\{\mathcal{R}\}^{(minsupp,minconf)}$ due to the sparse population of rules. When using an existing mining system, the user must progress through frustrating trial-and-error to finally get a new set of rules. In such situations, query Q2 saves time and effort by recommending which next parameter settings will cause a difference in the output.*

```
Q2:  OUTPUT Stable Region S^(usupp,uconf)_(lsupp,lconf)
FROM P
WITH minsupport=minsupp AND minconfidence=minconf;
```

Example 2: Unnoticeable change: *For a dense dataset such as Chess [3], each parameter setting produces a large number of associations. Suppose that using query Q2, a user changes the query input from ($minsupp^{old}$,$minconf^{old}$) to (lsupp,lconf) such that $minsupp^{old} \geq$ lsupp and $minconf^{old} \geq$ lconf. Then the ruleset $\{\mathcal{R}\}^{(lsupp,lconf)}$ would also contain the rules in the original ruleset $\{\mathcal{R}\}^{(minsupp^{old},minconf^{old})}$ satisfying ($minsupp^{old}$,$minconf^{old}$). The change in the ruleset may be difficult to discern, especially if the user has to manually inspect the ruleset $\{\mathcal{R}\}^{(lsupp,lconf)}$ to identify the newly added rules as well as some dominated ones. Here, a delta output of associations is desirable, as accomplished by Query Q3.*

```
Q3:  OUTPUT RuleSet S^(usupp,uconf)_(lsupp,lconf).{R}
FROM P
WITH minsupport=minsupp AND minconfidence=minconf;
```

Query Q3 returns the associations that belong only to the stable region containing the user-chosen (*minsupp*,*minconf*).

Example 3: Exploring multiple regions: *Analysts often compare two or more rules, with possibly different support and/or confidence values. In the mushroom dataset [3], analysts may be interested in comparing different types of mushrooms based on how the values of their attributes such as odor, cap-color and cap-shape co-occur. The* ***l-neighborhood query*** *Q4 allows users to retrieve l-neighbor stable regions such that the corresponding rules may be compared.*

```
Q4:  OUTPUT L-Neighbor Regions S^(usupp,uconf)_(lsupp,lconf).{S}
FROM P
WITH minsupport=minsupp AND minconfidence=minconf;
```

Query Q4 finds not only the stable region $\mathcal{S}_{(lsupp,lconf)}^{(usupp,uconf)}$ in which query parameter (*minsupp*,*minconf*) lies but also all its *lending* neighboring stable regions. The output is a set of stable regions with their respective bounds. The analyst could then further explore the neighbor regions for rules that they lend to $\mathcal{S}_{(lsupp,lconf)}^{(usupp,uconf)}$.

Redundancy Relationship (RR) Queries: This new query class explores redundancy relationships in the context of stable regions. The user inputs (*minsupp*,*minconf*), with little knowledge about the stable regions for the dataset. For every rule $\mathcal{R}_i$ in the region containing (*minsupp*,*minconf*), query Q5 returns the simple and strict dominating locations (see Lemmas 5.2 and 5.4 for details).

```
Q5:  OUTPUT Dominating Regions S^(usupp,uconf)_(lsupp,lconf).{S≫}
FROM P
WITH minsupport=minsupp AND minconfidence=minconf;
```

Query Q5 finds the stable region where (*minsupp*,*minconf*) lies. Then for the rules valid within the stable region, it finds all stable regions of their dominating rules. Query Q5 is helpful in determining for which parameter ranges, the rules within the stable region $\mathcal{S}_{(lsupp,lconf)}^{(usupp,uconf)}$, where (*minsupp*,*minconf*) lies, will be dominated by other rules.

3.2 The PARAS Framework

For efficiently processing the different classes of mining queries described above, we designed the **PARAmeter Space Framework**

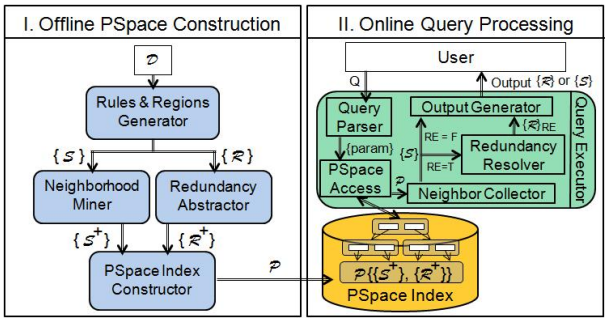


Figure 4: PARAS Framework.

for association mining, in short **PARAS**, as illustrated in Figure 4. **PARAS** consists of two phases (a) offline **PSpace** construction and (b) online query processing using the **PSpace** index. In the offline phase, to construct the **PSpace** index, we must tackle the following two challenges:

Challenge 1: constructing and compactly indexing the stable regions along with their association rules, and

Challenge 2: abstracting the redundancy information for each rule in some compact model.

The **Rules&Regions Generator** module performs the following two tasks, (a) generating all association rules and (b) constructing and populating stable regions. The generated ruleset $\{\mathcal{R}\}$ and the set of stable regions $\{\mathcal{S}\}$ are produced and passed on for further processing.

The **Neighborhood Miner** tackles the problem of repeated association rules (Section 1.3). It captures the neighborhood relationships among stable regions avoiding the need for maintaining associations repeatedly. The final set of stable regions together with their l-neighborhood information is called the *enriched set of stable regions*, denoted by $\{\mathcal{S}^+\}$. Our approach for stable region construction and neighborhood miner is explained in Section 4.

The **Redundancy Abstractor** captures the redundancy relationships among rules at the offline phase such that, if desired by the user, non-redundant association rules can be efficiently generated upon demand. The set of rules together with redundancy relationships is called the *enriched ruleset*, denoted as $\{\mathcal{R}^+\}$ (Section 5).

The **PSpace Index Constructor** uses the *enriched ruleset* $\{\mathcal{S}^+\}$ to create our proposed **PSpace** index, denoted by $\{\mathcal{R}^+\}$ and the *enriched set of stable regions* $\mathcal{P}$ (Section 5.3).

The online query processing phase is performed by the **Query Executor**. The **Query Parser** interprets the query to identify the *query class* and *type* as well as capture the *query parameter values*. The **PSpace Access** offers the API for accessing the index $\mathcal{P}$. A direct search on the **PSpace** index can be performed using the (*minsupp*,*minconf*) parameters to quickly retrieve the desired stable regions. **Neighbor Collector** uses the *enriched set of stable regions* $\{\mathcal{S}^+\}$ for a given region to *iteratively* find all *lending* neighboring stable regions. If the user desires *non-redundant associations*, the **Redundancy Resolver** module is employed. It uses the redundancy information prestored in $\{\mathcal{R}^+\}$. The **Output Generator** module presents the final output to the user. The description of *online query processing* is in Section 6.

4. OFFLINE PSPACE CONSTRUCTION

Our proposed **offline PSpace construction** (Algorithm 1) is composed of three tasks, task 1: generate all associations; task 2: compute stable regions; and task 3: determine neighborhoods among regions. To perform the above tasks we adapt the algorithms *ConstructLattice* and *GenerateRules* from [1]. For a dataset $\mathcal{D}$, an adjacency lattice $\mathcal{L}$ (Figure 1) is constructed using the *ConstructLattice* algorithm (explained in [1]). This lattice $\mathcal{L}$ is then utilized to perform the first two tasks below:

Task 1: Generate all associations. The original *GenerateRules* algorithm [1] utilizes a lattice $\mathcal{L}$ to generate non-redundant rules. This is achieved by a subroutine *FindBoundary* that, for a given itemset node $\mathcal{N}_i$ in lattice $\mathcal{L}$, returns only the *boundaryList*$^{\mathcal{N}_i}$[1] of parent nodes. However, **PARAS** needs to pre-store all associations for dataset $\mathcal{D}$. By replacing *FindBoundary* with *FindFullParentList*, the list of all parent itemsets is produced, denoted by *parentList*$^{\mathcal{N}_i}$. This modified *GenerateRules* generates all rules.

Task 2: Compute stable regions. We further modify the *GenerateRules* algorithm such that the stable regions are constructed in parallel with rule generation, as described in subroutine *Rules & Regions Generator* (Algo. 1.A). To partition the parameter space into stable regions, we first compute the *cut locations*. A *cut location* is identified by the *upper* location of a stable region. For $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$ the *cut location* is (0.4,0.5). Therefore, while generating rules using lattice $\mathcal{L}$, for itemset nodes $\{\mathcal{N}_i\}$ with identical support count (denoted by S($\mathcal{N}_i$)) in $\mathcal{L}$, the *support* for the *cut location* for $\mathcal{N}_i = \frac{S(\mathcal{N}_i)}{|\mathcal{D}|}$, where $|\mathcal{D}|$ (of $\mathcal{L}$ in Figure 1 = 100) is the total number of records. Similarly, for $\mathcal{N}_i$ and a parent node $\mathcal{N}^P_j \epsilon$ *parentList*$^{\mathcal{N}_i}$, the *confidence* value for the corresponding *cut location* is given by $\frac{S(\mathcal{N}_i)}{S(\mathcal{N}^P_j)}$. In other words, the *cut location* for $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$ is the location of association (X $\Rightarrow$ Y) in the parameter space. Therefore, each stable region is constructed in parallel during rule generation process as described in Algo. 1.A.

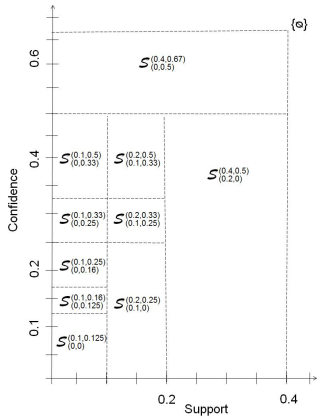


Figure 5: Stable Regions.

Task 3: Determine neighborhood relationships among regions. For each stable region $\mathcal{S}$, the *Neighborhood Miner* subroutine (Algorithm 1.B) adds the minimum parameter values. It also determines the list of *lending* neighbor stable regions. The final parameter space partitioned into its stable regions is depicted in Figure 5. Figure 6 lists all stable regions along with their associations and their *l-neighbor* stable regions.

In our example space, region $\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$ contains rule {(X$\Rightarrow$Y)} and has the region $\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$ as its *lending* neighbor. The complete list of associations for a particular stable region is given by the associations within the region plus the associations *recursively* collected from its *l-neighbors*. This way, a compact representation of stable regions along with the associations valid within them is achieved such that no association rule is stored repeatedly. The collection of regions enriched with neighbors is denoted by $\{\mathcal{S}^+\}$.

S. Regions	Neighbors	Associations
$\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$	$\varnothing$	{(Y⇒X)}
$\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$	$\mathcal{S}^{(0.4,0.67)}_{(0,0.5)}$	{(X⇒Y)}
$\mathcal{S}^{(0.2,0.5)}_{(0.1,0.33)}$	$\mathcal{S}^{(0.4,0.5)}_{(0.2,0)}$	{(Z⇒X),(Z⇒Y)}
$\mathcal{S}^{(0.2,0.33)}_{(0.1,0.25)}$	$\mathcal{S}^{(0.2,0.5)}_{(0.1,0.33)}$	{(Y⇒Z)}
$\mathcal{S}^{(0.2,0.25)}_{(0.1,0)}$	$\mathcal{S}^{(0.2,0.33)}_{(0.1,0.25)}$	{(X⇒Z)}
$\mathcal{S}^{(0.1,0.5)}_{(0,0.33)}$	$\mathcal{S}^{(0.2,0.5)}_{(0.1,0.33)}$	{(XZ⇒Y),(YZ⇒X)}
$\mathcal{S}^{(0.1,0.33)}_{(0,0.25)}$	$\mathcal{S}^{(0.1,0.5)}_{(0,0.33)}$+$\mathcal{S}^{(0.2,0.33)}_{(0.1,0.25)}$	{∅}
$\mathcal{S}^{(0.1,0.25)}_{(0,0.16)}$	$\mathcal{S}^{(0.1,0.33)}_{(0,0.25)}$+$\mathcal{S}^{(0.2,0.25)}_{(0.1,0)}$	{(XY⇒Z),(Z⇒XY)}
$\mathcal{S}^{(0.1,0.16)}_{(0,0.125)}$	$\mathcal{S}^{(0.1,0.25)}_{(0,0.16)}$	{(Y⇒XZ)}
$\mathcal{S}^{(0.1,0.125)}_{(0,0)}$	$\mathcal{S}^{(0.1,0.16)}_{(0,0.125)}$	{(X⇒YZ)}

Figure 6: Enriched Stable Regions.

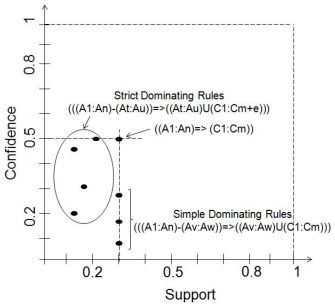


Figure 7: Dominating Rules.

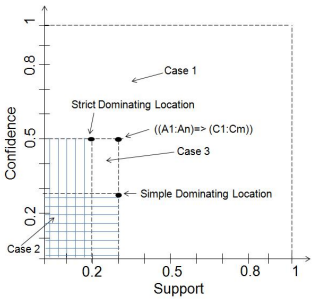


Figure 8: Dominating Locations.

5. REDUNDANCY RELATIONSHIPS

The rules produced using the stable regions constructed above may contain redundancies. In the parameter space model, our analysis derives certain properties of redundancy relationships that enable us to abstract redundancy information compactly as an offline step. When the user desires to retrieve non-redundant associations, **PARAS** can thus generate them efficiently at query-time.

5.1 Abstracting Redundancy Relationships

As a key observation we note that the **redundancy relationship is a query-time phenomenon**. In other words, the redundancy among rules depends on query-time input parameters rendering it impossible to eliminate a rule as redundant at an offline step. Instead, the redundancy relationships can be established only at query-time. For the parameter space in Figure 3, suppose that the user inputs (0.1,0.1) as query parameters. Here, the *dominating rule* $\mathcal{R}^{\gg}$ = (X ⇒ YZ) located at (0.1,0.125) qualifies as output. Then, to produce only non-redundant rules, the *dominated rules* $\{\mathcal{R}^{\ll}\}$ = { (XY ⇒ Z), (XZ ⇒ Y), (X ⇒ Y) and (X ⇒ Z) } must be eliminated. However, if the user inputs (0.1,0.2) instead, then (X ⇒ YZ) would not qualify for output and the rules previously deemed redundant are no longer redundant for this query. Thus, as the query parameters are supplied by the user at query-time, the decision about rule elimination can only be determined at query-time.

While elimination of the *dominated rules* can only be performed at query-time, our goal is to isolate as much as possible the redundancy relationships among rules inside the parameter space model in the preprocessing phase. This leads to the challenge that we must design a corresponding query-time strategy to produce non-redundant associations by utilizing this predetermined redundancy model. A straightforward yet expensive approach may proceed as follows. In the offline phase, for each rule $\mathcal{R}_j$, store the set of rules that dominate $\mathcal{R}_j$, denoted by $\{\mathcal{R}^{\gg}\}_j$. At query-time, if the rule $\mathcal{R}_j$ is included in the set of rules for the stable region containing query parameters (*minsupp*,*minconf*), then test if any of the rules dominating $\mathcal{R}_j$, denoted by $\mathcal{R}_i^{\gg} \in \{\mathcal{R}^{\gg}\}_j$, qualify the query parameters. If yes, then rule $\mathcal{R}_j$ is eliminated, else $\mathcal{R}_j$ is output.

Simple Dominating Rules and Location. Unfortunately, a rule $\mathcal{R}_j^{\ll_{sim}}$ may be *simple dominated* by multiple rules (Def. 4). For example, (XY ⇒ Z) is *simple dominated* by two rules, namely, (X ⇒ YZ) and (Y ⇒ XZ). Therefore, (XY ⇒ Z) can only qualify for output if neither of its *simple dominating rules* qualify. We call them the *set of simple dominating rules of rule* $\mathcal{R}_j^{\ll_{sim}}$ as we described in Def. 6. Here, the *simple dominating ruleset* is denoted by $\{\mathcal{R}^{\gg_{sim}}\}_j$ = {(X ⇒ YZ),(Y ⇒ XZ)}.

Definition 6. ***Simple Dominating Rule Set:*** *For a rule* $\mathcal{R}_j^{\ll_{sim}}$*, its simple dominating rule set, denoted by* $\{\mathcal{R}^{\gg_{sim}}\}_j$*, is the set of all the rules that* ***simple dominate*** $\mathcal{R}_j^{\ll_{sim}}$.

Algorithm 1 Offline PSpace Construction

Input: Dataset $\mathcal{D}$
Output: PSpace Index $\mathcal{P}$
begin
  $\mathcal{L}$ ⟵ ConstructLattice($\mathcal{D}$);
  $\{\mathcal{S}\}, \{\mathcal{R}\}$ ⟵ Rules&RegionsGenerator($\mathcal{L}$);
  $\{\mathcal{S}^+\}$ ⟵ NeighborhoodMiner($\{\mathcal{S}\}$);
  $\{\mathcal{R}^+\}$ ⟵ RedundancyAbstractor($\{\mathcal{R}\}$);
  $\mathcal{P}$ ⟵ PSpaceIndexConstructor($\{\mathcal{S}^+\}$,$\{\mathcal{R}^+\}$);
  return $\mathcal{P}$;

1.A: Rules&RegionsGenerator($\mathcal{L}$)
begin
  $\{\mathcal{S}\}$ ⟵ ∅;
  /* Get Parent List all k-itemsets in $\mathcal{L}$, k>1. */
  for *each* $\mathcal{N}_i \in \mathcal{L}$ **do**
    parentList$^{\mathcal{N}_i}$ ⟵ FindFullParentList($\mathcal{N}_i$);
  /* Generate rules and regions. */
  for *each* $\mathcal{N}_i \in \mathcal{L}$ **do**
    $\{\mathcal{R}\}$ ⟵ ∅;
    for *each* $\mathcal{N}_j^P \in parentList^{\mathcal{N}_i}$ **do**
      $\mathcal{R}^{i,j}$ ⟵ $\{I(\mathcal{N}_j^P) \Rightarrow I(\mathcal{N}_i) - I(\mathcal{N}_j^P)\}$;
      /* I($\mathcal{N}_i$) = itemset for $\mathcal{N}_i$ */
      $\{\mathcal{R}\}$ ⟵ $\{\mathcal{R}\} \cup \mathcal{R}^{i,j}$;
      $supp$ ⟵ $\frac{S(\mathcal{N}_i)}{|\mathcal{D}|}$; $conf$ ⟵ $\frac{S(\mathcal{N}_i)}{S(\mathcal{N}_j^P)}$;
      if ($\mathcal{S}^{old}$ ⟵ *getRegion*($\{\mathcal{S}\}$,$(supp, conf)$) ≠ ∅) **then**
        $\mathcal{S}^{old}$.addToRuleList($\mathcal{R}^{i,j}$);
      else
        Create New Region $\mathcal{S}^{new}$;
        $\mathcal{S}^{new}$.addUpperParameters($supp, conf$);
        $\mathcal{S}^{new}$.addToRuleList($\mathcal{R}^{i,j}$);
        $\{\mathcal{S}\}$ ⟵ $\{\mathcal{S}\} \cup \mathcal{S}^{new}$;
  return $\{\mathcal{S}\}$, $\{\mathcal{R}\}$;

1.B: NeighborhoodMiner($\{\mathcal{S}\}$)
begin
  for *each* $\mathcal{S}_i \in \{\mathcal{S}\}$ **do**
    $\mathcal{S}_i$.findLowerParameters();
    $\mathcal{S}_i$.findClosestHigherNeighbors();
  return $\{\mathcal{S}^+\}$;

1.C: RedundancyAbstractor($\{\mathcal{R}\}$)
begin
  for *each* $\mathcal{R}_j \in \{\mathcal{R}\}$ **do**
    $\{\mathcal{R}^{\gg_{sim}}\}_j$ ⟵ CollectTopSimpleDomRules($\mathcal{R}_j$);
    $\ell_j^{\gg_{sim}}$ ⟵ FindMaxDomLocation($\{\mathcal{R}^{\gg_{sim}}\}_j$);
    $\{\mathcal{R}^{\gg_{str}}\}_j$ ⟵ CollectTopStrictDomRules($\mathcal{R}_j$);
    $\ell_j^{\gg_{str}}$ ⟵ FindMaxDomLocation($\{\mathcal{R}^{\gg_{str}}\}_j$);
    $\mathcal{R}_j$.AddDominatingLocations($\ell_j^{\gg_{sim}}$,$\ell_j^{\gg_{str}}$);
  return $\{\mathcal{R}^+\}$;

For each *simple dominated rule* $\mathcal{R}_j^{\ll sim} \equiv ((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$, all rules that potentially *simple dominate* it conform to the template $(((A_1{:}A_n)\text{-}(A_v{:}A_w)) \Rightarrow ((A_v{:}A_w) \cup (C_1{:}C_m)))$. For the rule (XY $\Rightarrow$ Z) having two items X and Y in the antecedent, there are two *simple dominating rules*, considering all subsets of XY in the antecedent, namely, X and Y. Our observation is further generalized as in Lemma 5.1 below.

Lemma 5.1. *For a simple dominated rule $\mathcal{R}_j^{\ll sim} \equiv ((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$ with n antecedent items, the number of **simple dominating rules**, denoted by $|\{\mathcal{R}^{\gg sim}\}_j|$ is 2^n-2.* [3]

We further observe in Figure 7, that all rules in the set of simple dominating rules $\{\mathcal{R}^{\gg sim}\}_j$ have the same support value as rule $\mathcal{R}_j^{\ll sim}$. Thus it is possible to uniquely identify one single location containing one or more rules that is closest to $\mathcal{R}_j^{\ll sim}$ as described in Lemma 5.2.

Lemma 5.2. ***Simple Dominating Location:*** *For each simple dominated rule $\mathcal{R}_j^{\ll sim}$, the set of simple dominating rules $\{\mathcal{R}^{\gg sim}\}_j$ contains a rule $\mathcal{R}_i^{\gg sim}$ closest to the dominated rule $\mathcal{R}_j^{\ll sim}$, such that $\forall \mathcal{R}_k^{\gg sim} \epsilon \{\mathcal{R}^{\gg sim}\}_j$ and $(k \neq i)$, $\mathcal{R}_i^{\gg sim}.conf \geq \mathcal{R}_k^{\gg sim}.conf$. The location of rule $\mathcal{R}_i^{\gg sim}$ is called the **simple dominating location**[4] of $\mathcal{R}_j^{\ll sim}$, denoted by $\ell_j^{\gg sim}$.*

Strict Dominating Rules and Location. Similar to the above case of *simple dominating rules*, a *strict dominated rule* $\mathcal{R}_j^{\ll str}$ can be dominated by several *strict dominating rules* $\{\mathcal{R}^{\gg str}\}_j$. We call them the *set of strict dominating rules of rule* $\mathcal{R}_j^{\ll str}$ as defined below. In the example in Figure 1, for a rule $\mathcal{R}_j^{\ll str}$ = (X $\Rightarrow$ Y), the *strict dominating rule set* $\{\mathcal{R}^{\gg str}\}_j = \{(X \Rightarrow YZ)\}$.

Definition 7. *For a rule $\mathcal{R}_j^{\ll str}$, its **strict dominating rule set**, denoted by $\{\mathcal{R}^{\gg str}\}_j$, is the set of all the rules that **strict dominate** $\mathcal{R}_j^{\ll str}$.*

For a rule $\mathcal{R}_j^{\ll str}$, the number of such strict dominating rules can also be estimated as in Lemma 5.3.

Lemma 5.3. *For a strict dominated rule $\mathcal{R}_j^{\ll str}$ having n antecedent items $((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$, the strict dominating rules $\{\mathcal{R}^{\gg str}\}_j$ conform to the template $(((A_1{:}A_n)\text{-}(A_t{:}A_u)) \Rightarrow ((A_t{:}A_u) \cup (C_1{:}C_{m+e})))$. The cardinality of **strict dominating rules**, denoted by $|\{\mathcal{R}^{\gg str}\}_j|$, is 2^{n+e}, where e is the number of additional consequents in the dominating rules within the set $\{\mathcal{R}^{\gg str}\}_j$.*

Using Lemma 5.3, for a dominated rule $\mathcal{R}_j^{\ll str}$, the number of strict dominating rules can be determined. We further observe in Figure 7, that all rules in the set of strict dominating rules $\{\mathcal{R}^{\gg str}\}_j$ must have both their support and confidence values less than or equal to those of the rule $\mathcal{R}_j^{\ll str}$. Thus it is possible to uniquely identify one single rule location closest to $\mathcal{R}_j^{\ll str}$ as described in Lemma 5.4.

Lemma 5.4. ***Strict Dominating Location:*** *For each strict dominated rule $\mathcal{R}_j^{\ll str}$, the set of strict dominating rules $\{\mathcal{R}^{\gg str}\}_j$ contains one or more rules $\mathcal{R}_i^{\gg str}$ (in the same location) closest to the dominated rule $\mathcal{R}_j^{\ll str}$, such that $\forall \mathcal{R}_k^{\gg str} \epsilon \{\mathcal{R}^{\gg str}\}_j$ where $(i \neq k)$, $\mathcal{R}_i^{\gg str}.supp \geq \mathcal{R}_k^{\gg str}.supp$ AND $\mathcal{R}_i^{\gg str}.conf \geq \mathcal{R}_k^{\gg str}.conf$. The location of rule $\mathcal{R}_i^{\gg str}$ is called the **strict dominating location**[4], denoted by $\ell_j^{\gg str}$.*

[3] Proofs of the lemmas and the theorems are omitted due to space restriction and can be found in [13].

[4] Multiple rules may map to the simple / strict dominating location that collectively represents them.

Lemmas 5.2 and 5.4 now lead us to a **key insight**, namely, they allow us to compactly store the association rules along with respective redundancy relationships. For each rule $\mathcal{R}_j$, only its two locations, namely, the *simple dominating location* $\ell_j^{\gg sim}$ and the *strict dominating location* $\ell_j^{\gg str}$ must be captured by the offline step. At the online query processing phase, these two locations are sufficient to determine the redundancy relationship for rule $\mathcal{R}_j$, as described in Theorem 1.

Theorem 1. ***Constant time criteria for online redundancy determination:*** *Given an online mining query with parameter values (minsup, minconf), for each rule $\mathcal{R}_j$ in the result set $\{\mathcal{R}_j\}$, only **two parameter space locations**, namely, the **simple dominating location** $\ell_j^{\gg sim}$ and the **strict dominating location** $\ell_j^{\gg str}$ are sufficient for determining whether the rule $\mathcal{R}_j$ is redundant. In the case of ($minsupp \leq \ell_j^{\gg sim}.supp$ AND $minconf \leq \ell_j^{\gg sim}.conf$) OR ($minsupp \leq \ell_j^{\gg str}.supp$ AND $minconf \leq \ell_j^{\gg str}.conf$) is true, the rule $\mathcal{R}_j$ is redundant. Otherwise, the rule $\mathcal{R}_j$ is not redundant.*

As illustrated in Figure 8, the online algorithm for *redundancy resolution* requires checking the three cases of where the user input (*minsupp*,*minconf*) lies with respect to each rule in the result set and their dual dominating locations.

5.2 Optimized Location Computation

The *Redundancy Abstractor* subroutine (Algorithm 1.C) captures redundancies for each rule $\mathcal{R}_j$. Suppose $\mathcal{R}_j$ is of the form $((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$ as in Figure 7. One straightforward approach for computing the simple dominating location for rule $\mathcal{R}_j$ is as follows: (a.) collect all simple dominating associations and (b.) find the simple dominating rule with the maximum (*supp*,*conf*) value pairs. The location of that rule is the simple dominating location $\ell_j^{\gg sim}$ for rule $\mathcal{R}_j$. Clearly, the same process could be employed to find the strict dominating location $\ell_j^{\gg str}$.

However, this process of finding the dominating location by searching through the set $\{\mathcal{R}^{\gg sim}\}_j$ of k rules is equivalent to *finding the largest of k numbers*. It requires $\mathcal{O}$(k) time. From Lemma 5.1, the total number of simple dominating rules for a rule $((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$ having n antecedents is 2^n-2. Therefore, finding the simple and strict dominating locations using the complete set of dominating rules (here k = 2^n-2) is very computationally intensive. Below, we instead demonstrate that a much smaller subset of dominating association rules is sufficient for computing the dominating locations, as stated in Theorem 2.

Theorem 2. ***Top Simple Dominating Rules:*** *For a rule $((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$ having n antecedents, to find the simple dominating location, it is necessary and sufficient to search only the **(n-1)-antecedent** simple dominating rules with format $(((A_1{:}A_n)\text{-}A_i) \Rightarrow (A_i \cup (C_1{:}C_m)))$, called the **top simple dominating rules**.*

Lemma 5.5. *The total number of **top simple dominating rules** is given by $\binom{n}{n-1} = n$.*

Theorem 3. ***Top Strict Dominating Rules:*** *For a rule $((A_1{:}A_n) \Rightarrow (C_1{:}C_m))$ to find the strict dominating location, it is necessary and sufficient to search only the e dominating rules with format $(((A_1{:}A_n)) \Rightarrow ((C_1{:}C_m) \cup C_h))$, where e is the number of consequent items in the dominating rules but not in $(C_1{:}C_m)$ and C_h is a single item out of the set $(C_{m+1}{:}C_{m+e})$. We call them the **top strict dominating rules**.*

Using Theorem 2, only the *top n simple dominating rules* must be collected using *CollectTopSimpleDomRules* method, instead of

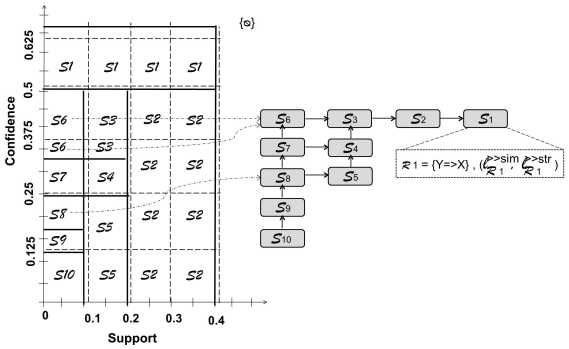


Figure 9: The PSpace Index.

all the 2^n-2 simple dominating rules. Similarly, using Theorem 3, only the *top e strict dominating rules* must be collected using the *CollectTopStrictDomRules* method, where e is the number of additional consequent items in the dominating rules but not in (C_1:C_m). The overall approach is given in Algorithm 1.C. The optimizations achieved by Theorems 2 and 3 result in significant improvements as the offline redundancy abstraction now requires $\mathcal{O}(n^2)$ time opposed to the approach utilizing all dominating rules that would require $\mathcal{O}(2^n)$. The collection of rules enriched with redundancy information is denoted by $\{\mathcal{R}^+\}$.

5.3 The Compact PSpace Index

Using the *enriched stable regions* $\{\mathcal{S}^+\}$ and the enriched ruleset $\{\mathcal{R}^+\}$, the **PSpace** is created using the subroutine *ConstructIndex* (Algorithm 1) and indexed by a two-layered index, called the **PSpace Index**. The top level of the PSpace index facilitates the search to locate a particular stable region given input parameters. As such, any spatial indexing method could be utilized. In our implementation, grid-based spatial indexing is utilized to partition the PSpace into equal sized grid cells and allocate the stable regions to their respective positions in the grid. A stable region may span over one or more grid cells, while a grid cell is allocated at least one stable region. The stable regions in each cell point to the corresponding nodes in the next level of the PSpace index. In Figure 9, we partition PSpace into 0.1 × 0.125 sized grid cells. *S8* and *S9* are in the same grid cell, while *S6* spans over two grid cells. For optimization, we reduce the number of grid cells by marking the upperbounds for support and confidence and indicating that no stable region exist beyond the upperbound in *PSpace*. For online query processing using the grid see Section 6. Using the proposed grid structure, the online search for a stable region can be performed in near constant time as shown in Section 6.

The next level of the PSpace index, namely, the *region neighborship graph* (R.H.S. of Figure 9), expedites the collection of rules from neighbor regions. Each stable region forms a node in the graph and each node is linked to its lending neighbors. As each region may have at most two such neighbors, these links result in a *sparse directed graph* of stable regions. The *region neighborship graph* enables us to not only locate lending neighbor regions in near constant time, but also to produce complete rule sets in time linear to the number of rules involved. Inside the *region neighborship graph*, each region node consists of the enriched ruleset unique to the region (Figure 6) along with their respective simple and strict dominating locations. For example, node $\mathcal{S}_1$ in Figure 9 stores rule $\mathcal{R}_1 \equiv (Y \Rightarrow X)$ as well as its dominating locations $\ell^{\gg sim}_{\mathcal{R}_1}$ and $\ell^{\gg str}_{\mathcal{R}_1}$. The **PSpace index** is used to efficiently process a rich variety of online exploratory queries as shown in Section 6.

6. ONLINE QUERY PROCESSING

We now explain how the different classes of online mining queries (Section 3.1) are processed by **PARAS** (Algorithm 2). The *PSpace Access* module is employed to load the *PSpace index* $\mathcal{P}$ (see Section 5.3 for index details). Depending on the *query class*, interpreted by the *Query Parser*, the apropriate subroutine for the query class is invoked. If the *query class* is RM (or SR or RR), then the subroutine *RuleMiningQ* (or *StableRegionQ* or *RedundancyQ*) is invoked with appropriate parameter values.

Algorithm 2 Online Query Processing

Input: Query $\mathcal{Q}$
Output: RuleSet $\{\mathcal{R}\}$ or RegionSet$\{\mathcal{S}\}$
begin
  $\mathcal{P} \longleftarrow$ PAccess.GetIndex();
  if *QParser.GetQClass($\mathcal{Q}$) == RM* **then**
    RuleMiningQ(QParser.GetQRE($\mathcal{Q}$),$(minsupp, minconf)$,$\mathcal{P}$);
  else if *QParser.GetQClass($\mathcal{Q}$) == SR* **then**
    StableRegionQ(qType,$(minsupp, minconf)$,$\mathcal{P}$);
  else if *QParser.GetQClass($\mathcal{Q}$) == RR* **then**
    RedundancyQ(qType,$(minsupp, minconf)$,$\mathcal{P}$);

2.A: RuleMiningQ(REFlag,$(minsupp, minconf)$,$\mathcal{P}$)
begin
  $\{\mathcal{R}\} \longleftarrow \emptyset$; $\mathcal{S} \longleftarrow \mathcal{P}$.LocSearchRegion$(minsupp, minconf)$;
  $\{\mathcal{R}\} \longleftarrow \{\mathcal{R}\} \cup \mathcal{S}$.GetRuleSet();
  neighborList $\longleftarrow$ NeighborCollector($\mathcal{S}$); /* Get all neighbors. */
  for *each* $\mathcal{S}_i \in$ *neighborList* **do**
    $\{\mathcal{R}\} \longleftarrow \{\mathcal{R}\} \cup \mathcal{S}_i$.GetRuleSet(); /* Collect RuleSets. */
  if *REFlag.IsTrue()* **then**
    $\{\mathcal{R}\} \longleftarrow$ RedundancyResolver($\{\mathcal{R}\}$,$(minsupp, minconf)$);
  return $\{\mathcal{R}\}$;

2.B: StableRegionQ(qType,$(minsupp, minconf)$,$\mathcal{P}$)
begin
  if *qType == Q2* **then**
    return $\mathcal{P}$.LocSearchRegion$(minsupp, minconf)$;
  else if *qType == Q3* **then**
    $\mathcal{S} \longleftarrow \mathcal{P}$.LocSearchRegion$(minsupp, minconf)$;
    return $\mathcal{S}$.GetRuleSet();
  else if *qType == Q4* **then**
    $\mathcal{S} \longleftarrow \mathcal{P}$.LocSearchRegion$(minsupp, minconf)$;
    return $\mathcal{S}$.GetNeighborList();
  return false;

2.C: RedundancyQ(qType,$(minsupp, minconf)$,$\mathcal{P}$)
begin
  $\{\mathcal{L}^{\gg}\} \longleftarrow \emptyset$; $\{\mathcal{R}\} \longleftarrow \emptyset$;
  $\{\mathcal{R}\} \longleftarrow StableRegionQ(Q3, (minsupp, minconf), \mathcal{P})$;
  /* Iterate over rules. */
  for *each* $\mathcal{R}_j \in \{\mathcal{R}\}$ **do**
    /* Collect Simple Dom Loc. */
    $\mathcal{L}^{\gg sim} \longleftarrow copyLocation(\mathrm{R}_j.\ell^{\gg sim}.supp, \mathcal{R}_j.\ell^{\gg sim}.conf)$;
    $\{\mathcal{L}^{\gg}\} \longleftarrow \{\mathcal{L}^{\gg}\} \cup \mathcal{L}^{\gg sim}$;
    /* Collect Strict Dom Loc. */
    $\mathcal{L}^{\gg str} \longleftarrow copyLocation(\mathrm{R}_j.\ell^{\gg str}.supp, \mathcal{R}_j.\ell^{\gg str}.conf)$;
    $\{\mathcal{L}^{\gg}\} \longleftarrow \{\mathcal{L}^{\gg}\} \cup \mathcal{L}^{\gg str}$;
  return $\{\mathcal{L}^{\gg}\}$;

RuleMiningQ. Algorithm 2.A describes how the rule mining query is processed. The *LocSearchRegion* method performs a location search on the **PSpace index** using (*minsupp*,*minconf*) as input to retrieve the stable region $\mathcal{S}$ that contains (*minsupp*,*minconf*). $\mathcal{S}$ is *enriched* with its ruleset and neighbors. The *NeighborCollector* module recursively collects all the neighbor stable regions. The

output ruleset $\{\mathcal{R}\}$ consists of the ruleset of region $\mathcal{S}$ and the rulesets of the *lending neighbors*. If *REFlag* is set to *TRUE*, the *RedundancyResolver* (Algorithm 3) performs the inexpensive checks, as illustrated in Figure 8, to reduce the output ruleset.

Algorithm 3 Redundancy Resolver

Input: RuleSet $\{\mathcal{R}\}$, $(minsupp, minconf)$
Output: Redundancy Eliminated RuleSet $\{\mathcal{R}\}^{RE}$
begin
  $\{\mathcal{R}\}^{RE} \longleftarrow emptyset$;
  for *each* $\mathcal{R}_i \in \{\mathcal{R}\}$ **do**
    $simDL \longleftarrow \mathcal{R}_i$.GetSimDomLoc();
    $strDL \longleftarrow \mathcal{R}_i$.GetStrDomLoc();
    /* $\mathcal{R}_i$ qualifies by failing Case 1. Case 2: */
    if $(((minsupp \leq simDL.supp)$ *AND* $(minconf \leq simDL.conf))$ *OR* $((minsupp \leq strDL.supp)$ *AND* $(minconf \leq strDL.conf)))$ **then**
      PRINT ($\mathcal{R}_i$ is dominated.);
    else
      /* Case 3 */
      $\{\mathcal{R}\}^{RE} \longleftarrow \{\mathcal{R}\}^{RE} \cup \mathcal{R}_i$;
  return $\{\mathcal{R}\}^{RE}$;

The **response time** for the rule mining query consists of three components, namely, Cost(LocSearchRegion) + Cost(Neighbor Collection) + Cost(Redundancy Resolution). Our **PSpace index** is an in-memory structure that compactly represents all the stable regions *enriched* with rulesets and neighbors. In case the data exceeds the memory, disk space is utilized for extra storage. The cost for a location search on the grid structure of the **PSpace index** is given as Cost(LocSearchRegion) = $\mathcal{O}(1)$.As illustrated in Figure 9, by converting input parameters (0.15,0.2) into offsets, the appropriate cell can be found in constant time and the stable region (here, S_5) is retrieved. For *neighbor collection* on the *sparse directed graph* of stable regions a *depth first search* (DFS) is required starting at the node containing (*minsupp,minconf*). The time complexity of DFS is $\mathcal{O}(|V| + |E|)$. In our case, $E \leq (2 \times V)$ as each vertex has a fanout of at most two edges, thus, Cost(Neighbor Collection) = $\mathcal{O}(|V|)$. Further, assuming uniform distribution of regions in the 2-D space, the number of stable regions that lie above (*minsupp,minconf*) is denoted by $V = \{N_S \times$ (1 - *minsupp*) $\times$ (1 - *minconf*)$\}$, where N_S is the number of stable regions. For very low (*minsupp,minconf*) input, all N_S stable regions must be collected. If the **I**ntermediate set of **R**ules $\{\mathcal{R}\}$ input to the *redundancy resolution* method contains a total of N_{IR} rules, the time required for *redundancy resolution* is $[N_{IR} \times C_{RR}]$, where C_{RR} is the constant cost of redundancy checking over a single candidate rule (Algo. 3). If the **PSpace index** requires secondary storage such that both grid cells and stable regions are stored on disk, additional costs for disk access are added into the costs of location search and neighbor collection. Redundancy resolution can still be performed in-memory over the retrieved stable regions.

StableRegionQ. Algorithm 2.B describes how the three stable region queries are processed. All three stable region query types invoke the *LocSearchRegion* method to retrieve the stable region containing (*minsupp,minconf*). *Query type* Q2 simply returns stable region $\mathcal{S}$ as output. As each stable region $\mathcal{S}$ is *enriched* with its ruleset, the *GetRuleSet* method in Q3 returns the ruleset of $\mathcal{S}$. Similarly, for Q4, *GetNeighborList* returns the neighbor list for $\mathcal{S}$.

Similar to query Q1, each of the *stable region queries* incurs the location search cost (Cost(LocSearchRegion)). The ruleset for Q3 and the neighbor list for Q4 can be retrieved in near constant time as each stable region is *enriched* with that information. Thus, Cost(Q2)=Cost(Q3)=Cost(Q4)= Cost(LocSearchRegion) + C_{SRQ}, where C_{SRQ} is the constant cost of accessing the rules and/or neighbors of a stable region.

RedundancyQ. Algorithm 2.C describes how the redundancy query is processed. The *dominating stable regions* are desired. For this query the *StableRegionQ* subroutine is invoked with *query type* Q3, (*minsupp,minconf*) and $\mathcal{P}$ as input. For each rule $\mathcal{R}_j$ in the returned ruleset $\{\mathcal{R}\}$, the *simple* and *strict dominating* locations of $\mathcal{R}_j$ are retrieved to collect the output stable regions $\{\mathcal{S}^{\gg}\}$.

The **response time** of *redundancy query* Q5 is given by Cost(Q5) = [Cost(LocSearchRegion) + N_R] = $\mathcal{O}(N_R)$. Here, N_R denotes the number of rules in the stable region containing (*minsupp,minconf*). For each rule $\mathcal{R}_j$, its *simple* and *strict* dominating locations are retrieved in near constant time.

7. EXPERIMENTAL EVALUATION

Experimental Setup. We conducted experiments on a Windows 7 machine with Intel(R) Xeon(R) CPU X3440@2.53 GHz processor and 8 GB of RAM. All algorithms were coded in C++ using Visual Studio 2010.

Experimental Datasets. We evaluated the performance of the **PARAS** system and its competitors using synthetic and real dataset benchmarks. We used two synthetic datasets generated by the *IBM Quest data generator* [2] modeling transactions in a retail store, *T10I4D100k* (T100k) and *T10I4D5000k*(T5000k). T5000k has 5 million transactions with 1000 items. On average, each transaction has 10 items. The data file size is about 200 MB. We also tested the *Webdocs* dataset from FIMI Repository [14]. The *webdocs* dataset captures real data of spidered web html documents. Webdocs has 1.7 million transactions with 5,267,656 distinct items. The maximal length of a transaction is 71472. The data file size is about 1.5 GB. The results for two additional real datasets from the UC Irvine Machine Learning Repository [3], namely, *chess* and *mushroom* are available in the technical report [13] due to space constraints in this paper. Thus, these diverse datasets are suitable for evaluating the scalability of **PARAS** and its competitors.

Alternate State-of-the-art Techniques. While we had difficulty in executing the original rule mining algorithms in Apriori [2], Eclat [21] and *FP-growth* [10] on the large data sets, improved C++ implementations of these algorithms available in [4] run successfully. We evaluated the performance of online mining queries with and without redundancy resolution. For mining requests without redundancy resolution, the performance of **PARAS** is compared against the original Apriori, Eclat, FP-growth from [4]. For requests with *redundancy resolution*, we compared **PARAS** (which produces non-redundant rules) against the above three online mining algorithms by adding online redundancy resolution code such that the results produced by all algorithms are comparable (identical). The algorithms enhanced with redundancy resolution (RR) are henceforth referred to as AprioriRR, EclatRR and FPgrowthRR. Next, we also compared **PARAS** against the POQM solution [1]. As it involves an offline step to generate and pre-store frequent

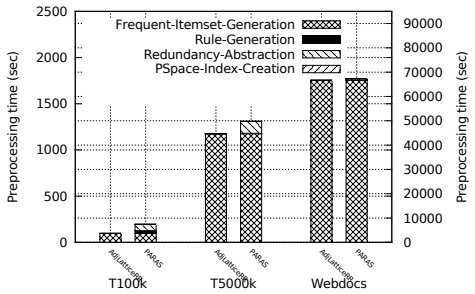
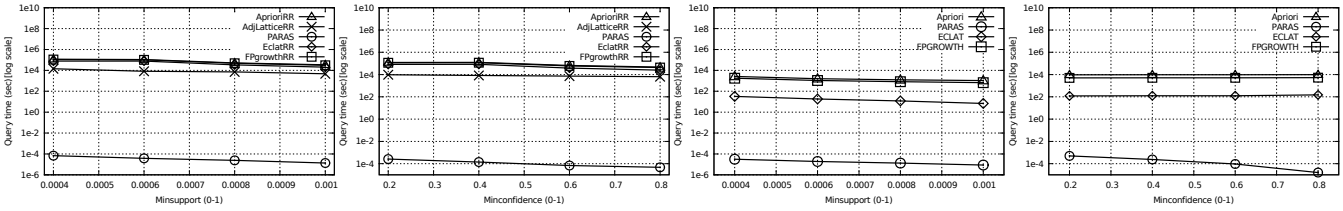


Figure 10: Preprocessing Times for All Three Datasets.



(a) minconf = 0.60 (b) minsupp = 0.0004 (c) minconf = 0.60 (d) minsupp = 0.0004

Figure 11: Avg. Online Query Processing Times (T100k) [Rules w Redundancy Resolution in (a),(b) and w/o in (c),(d)].

Dataset	AdjLatticeRR *(supp)*	PARAS *(supp, conf)*
T100k	(0.0001)	(0.0001, 0.1)
T5000k	(0.0003)	(0.0003, 0.15)
Webdocs	(0.08)	(0.08, 0.30)

Table 1: Thresholds for Indexes

Dataset	Varying *minsupp* ({*minsupp*},*minconf*)	Varying *minconf* (*minsupp*,{*minconf*})
T100k	({0.0004, 0.0006, 0.0008, 0.0010}, 0.60)	(0.0004. {0.20, 0.40, 0.60, 0.80})
T5000k	({0.0005, 0.0010, 0.0015, 0.0020}, 0.60)	(0.0010, {0.20, 0.40, 0.60, 0.80})
Webdocs	({0.10, 0.15, 0.20, 0.25}, 0.60)	(0.15, {0.45, 0.60, 0.75, 0.90})

Table 2: Online Query Settings

itemsets within an *adjacency lattice* (Sec. 1.2), we call it AdjLatticeRR. The online step generates the rules with redundancy resolution (pseudocode in [1]). As **PARAS** and the other mining techniques adopt the redundancy definitions in AdjLatticeRR [1], for each mining request, all five approaches produce identical results.

Experimental Methodologies. Performance measures are:

• **Offline Preprocessing Times.** We measure the total offline preprocessing times for AdjLatticeRR and **PARAS**. As AprioriRR, EclatRR and FPgrowthRR do not involve any preprocessing, they are excluded.

• **Mean Online Processing Times.** We measure the online processing time for a query averaged over several runs, for all five methods. We varied the *minsupp* and *minconf* query input parameters in the range [0,1].

• **Index Sizes.** We compare the sizes of the preprocessed information. AprioriRR, EclatRR, and FPgrowthRR are fully online techniques without any preprocessing involved. Thus, we compared the size of the adjacency lattice in AdjLatticeRR (i.e., # of frequent itemsets) and the *PSpace index* size in **PARAS** (i.e., # of stable regions) against the # of associations. We studied the impact of varying the primary support threshold on these index sizes.

7.1 Evaluation of Preprocessing Times

We first compare the preprocessing times for **PARAS** and AdjLatticeRR. AdjLatticeRR generates the frequent itemsets offline, whereas offline preprocessing in **PARAS** involves the four steps of *frequent itemset generation*, *rule&region generation*, *redundancy abstraction* and *PSpace index creation*. Among these, for each dataset, frequent itemset generation takes the longest preprocessing time for both **PARAS** and AdjLatticeRR (Fig. 10). This confirms prior works [1, 10, 15] that rule generation is more efficient compared to frequent itemset generation. However, we now show that if redundancy resolution is required, the overall online processing time becomes significantly higher. In **PARAS**, while redundancy abstraction at the offline step adds offline overhead, it significantly reduces the online redundancy resolution costs (as we will see in Sec. 7.2). In Fig. 10, *T100k* and *T5000k* datasets (left y axis), redundancy abstraction has higher overhead than rule&region generation and PSpace index creation. However, for *Webdocs* (right y axis), compared with the cost of frequent itemset generation (60k+ seconds), the costs of the other steps, namely, rule&region generation, redundancy abstraction and PSpace index creation are negligible. Overall the three additional preprocessing steps in **PARAS** require no more than 10% extra time than *AdjLatticeRR*. Since they are done only once offline, acceptable in practice.

7.2 Evaluation of Online Processing Time

Next, we varied parameters *minsupp* or *minconf* (x-axis) and compared the online processing times (y-axis in log scale) of the alternative techniques. Table 1 (column two) lists for each tested dataset, the primary support threshold used to prestore frequent itemsets in the *adjacency lattice*. Column three lists the primary support and confidence thresholds used for populating the **PSpace index** of **PARAS**. We performed two sets of experiments.

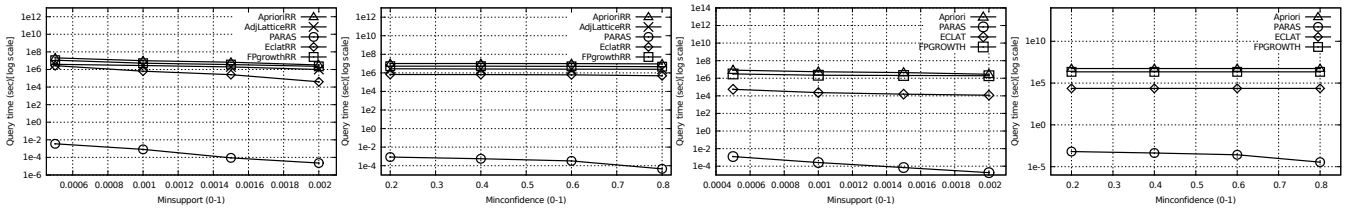
7.2.1. Evaluation Involving Redundancy Resolution. First, we compare AprioriRR, AdjLatticeRR, EclatRR, FPgrowthRR and **PARAS** for user queries involving *redundancy resolution*. For query Q1 in **PARAS**, we set *REFlag* = TRUE. The query processing times are averaged over several runs of each query. To determine the effect of varying *minsupp*, we conducted several experiments by fixing *minconf* to a constant value and varying just the *minsupp* value.

7.2.1.A. Impact of Varying *minsupp*. Table 2 (column one) lists the fixed *minconf* and different *minsupp* values used for the three datasets. Figs. 11(a), 12(a) and 13(a) illustrate the query processing times for T100k, T5000k and Webdocs datasets, respectively. For all five techniques, the query processing time decreased with increase in the *minsupp*. As *minsupp* increases more rules get filtered - producing fewer rules as output. For AprioriRR, EclatRR, FPgrowthRR and AdjLatticeRR, a smaller number of frequent itemsets are processed for rule generation. For **PARAS** fewer stable regions are considered for composing the output ruleset and fewer rules require redundancy resolution.

Overall, **PARAS** consistently performed *several orders of magnitude* better than the four competitors. In particular, **PARAS** outperformed AprioriRR by 4, 5 and 5 orders, AdjLatticeRR by 4, 4 and 5 orders, EclatRR by 4, 4 and 4 orders and FPgrowthRR by 4, 5 and 5 orders for *T100k*, *T5000k* and *Webdocs* datasets, respectively.

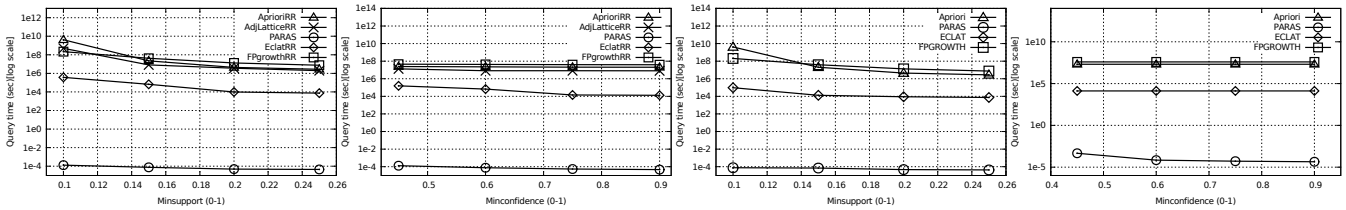
7.2.1.B Impact of Varying *minconf*. Next, we fixed the *minsupp* to a constant value and measured query processing times by varying *minconf* values (Table 2, column two). Figs. 11(b), 12(b) and 13(b) depict the processing times for T100k, T5000k and Webdocs datasets, respectively. The trend of the five alternate algorithms is similar as before. **PARAS** outperformes the four competitor approaches by several orders of magnitude.

Overall, **PARAS** consistently outperformed AprioriRR by 3, 4 and 5 orders, AdjLatticeRR by 3, 4 and 5 orders, EclatRR by 4, 3 and 4 orders and FPgrowthRR by 4, 4 and 5 orders for *T100k*, *T5000k* and *Webdocs* datasets, respectively. We note that the rate of decrease (slope) of the query processing times with the increase in *minconf* is not as steep as the slope with increase in *minsupp*.



(a) minconf = 0.60 (b) minsupp = 0.0010 (c) minconf = 0.60 (d) minsupp = 0.0010

Figure 12: Avg. Online Query Processing Times (T5000k) [Rules w Redundancy Resolution in (a),(b) and w/o in (c),(d)].



(a) minconf = 0.60 (b) minsupp = 0.0004 (c) minconf = 0.60 (d) minsupp = 0.0004

Figure 13: Avg. Online Query Processing Times (Webdocs) [Rules w Redundancy Resolution in (a),(b) and w/o in (c),(d)].

7.2.1.C Evaluation of Handling Multiple Queries. We now compare the processing times for multiple successive queries. Fig. 14 depicts the chart for the *Webdocs* dataset with a number of successive 2, 10, 20 and 50 queries (x-axis). The total processing time is show on y-axis. A diversity of queries are generated by randomly selecting *minsupp* values between 0.10 and 0.30 and *minconf* values between 0.10 and 0.90, respectively. AprioriRR performs all steps at query-time. AdjLatticeRR only performs rule generation with redundancy resolution at query-time. Thus it outperforms AprioriRR. **PARAS** only requires a look-up in the **PSpace index** and performs the inexpensive *redundancy resolution* using the dominating locations prestored within the **PSpace index**. Thus, **PARAS** delivers instantaneous responses even for large workloads with 50 queries or more. In Figure 14, for the case of *two* successive queries, all five approaches performed reasonably well. As the number of queries increases, the gains of using **PARAS** became more apparent. For 50 successive queries, **PARAS** took less than 1 second whereas AprioriRR, AdjLatticeRR, EclatRR and FPgrowthRR used approximately 38, 21, 2 and 26 hours, respectively.

7.2.2. Evaluation of Queries without Redundancy Resolution. Next, we considered user requests without redundancy resolution. We compared Apriori, Eclat and FPgrowth against **PARAS** by setting the Boolean *REFlag* to FALSE. AdjLatticeRR cannot be compared as it only produces non-redundant association rules. Similar as above, we conducted separate experiments by fixing one of the query parameters and varying the other as discussed below.

7.2.2.A. Impact of Varying *minsupp*. Figures 11(c), 12(c) and 13(c) depict charts for the three tested datasets. **PARAS** outperformed Apriori by 4, 5 and 5 orders, Eclat by 3, 4 and 3 orders, FPgrowth by 4, 4 and 5 orders for *T100k*, *T5000k* and *Webdocs* datasets, respectively.

7.2.2.B. Impact of Varying *minconf*. Figures 11(d), 12(d) and 13(d) depict charts for the three tested datasets. **PARAS** outperformed Apriori by 3, 4 and 5 orders, Eclat by 2, 3 and 3 orders, FPgrowth by 3, 4 and 5 orders for *T100k*, *T5000k* and *Webdocs* datasets, respectively.

7.3 Evaluation of Index Sizes

We compare the sizes of the prestored index structures used in AdjLatticeRR and **PARAS**. AprioriRR, EclatRR and FPgrowthRR are skipped as they are entirely online. For AdjLatticeRR, the adjacency lattice size is determined by the number of frequent itemsets, while *PSpace index* size by the number of stable regions. The actual index sizes (in say, MB) can be estimated by multiplying the number of instances (itemsets or stable regions) with the average space required per instance. The lower the primary support threshold, the larger the number of frequent itemsets stored in the *adjacency lattice*. Similarly, the choices of primary support and confidence thresholds determine the number of stable regions and rules stored within the *PSpace index*.

In Figs. 15.(a),(b),(c), for the three datasets, we examine how the numbers (y-axis) of frequent itemsets, stable regions and association rules change with respect to changes in the *primary support* [1] threshold values (x-axis). The primary support thresholds are in reverse order to show how the index sizes increase as the primary support threshold is relaxed to lower values.

For *T100k* (Fig. 15(a)), as the primary support changes from 0.0010 to 0.0004, the numbers of stable regions remain unchanged or at best increase slightly whereas the numbers of frequent itemsets and rules increase gradually. For *T5000k* (Figure 15(b)), the numbers of frequent itemsets, stable regions and rules increase when primary support changes from 0.0020 to 0.0005. For *Webdocs* (Figure 15(c)), the primary confidence is fixed at 0.30. the numbers of frequent itemsets, stable regions and rules increase gradually with the relaxation in primary support from 0.25 to 0.15, whereas the change is rapid for primary support 0.15 and 0.10. Overall, our PSpace index is slightly larger than the lattice of AdjLatticeRR.

Experimental Conclusions. The main findings are:

- **PARAS** requires about 10% extra offline preprocessing time compared with AdjLatticeRR, which is acceptable.
- For a large diversity of online queries, **PARAS** consistently outperforms the state-of-the-art competitors from the literature by 2 to 5 orders of magnitude over the tested datasets.
- The benefits of **PARAS** are more apparent when multiple successive queries are processed. As **PARAS** processes several queries within a second, thus staying within the needs of human attention span for interactive exploration. On the other hand, the competitors take several hours for the same.
- The *PSpace index* size of **PARAS** is on average 3.3× the *adjacency lattice* of AdjLatticeRR. The modern costs of memory makes this tradeoff practical given the huge CPU savings.
- Overall, the gains of several orders of magnitude when using **PARAS** for online processing outweigh the one-time minimal offline preprocessing time and storage requirements.

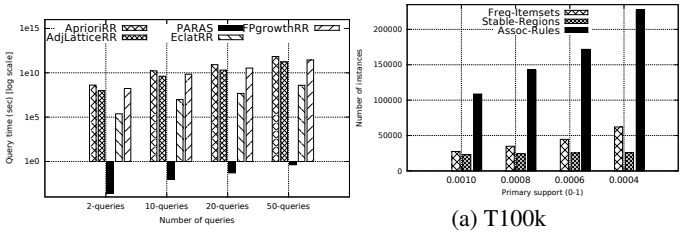
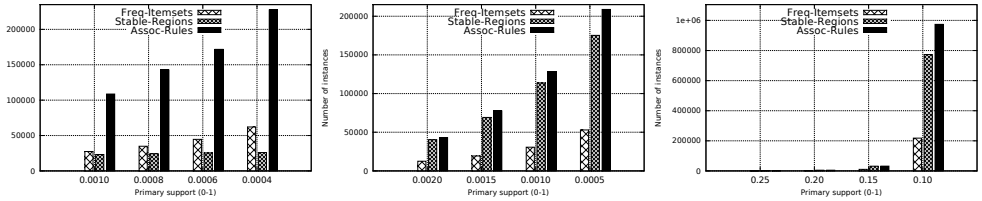


Figure 14: Multi-Query Costs.



(a) T100k (b) T5000k (c) Webdocs

Figure 15: Size of the PSpace Index.

8. RELATED WORK

Association Rule Mining. Online mining techniques, [1, 11, 12] apply the principles of POQM only partially such that they prestore only intermediate values, namely, the frequent itemsets, and not final results, namely, the rules. Adjacency lattice [1] and FP-tree [10] are two commonly used structures for prestoring itemsets. Aggarwal et al. [1] also introduce the concept of redundancy among rules to reduce the number of rules in the query output. They avoid the prohibitive costs of frequent itemset generation by prestoring itemsets. At query-time, they perform rule generation with redundancy resolution. This combined online step can significantly increase the query response time. In contrast, we explore the space of the query parameters to capture the distribution of rules within the space, instead of prestoring the itemsets.

Interestingness Measures as Parameters. [17, 19] identify the importance of analyzing the interestingness measures of rules. Han et al. [19] compare different null-invariant measures and provide insights into similarities and differences among them. The interestingness preprocessing step [17] reduces the number of potentially interesting rules that must undergo interestingness checks. Neither of these works tackle online mining through precomputation. In contrast, we explore the space of interestingness parameters for prestoring data mining results to facilitate fast online mining.

Parameter Space Exploration. Prior works have explored the space of parameters for handling parameterized database queries [6] and tuning database configuration parameters [7]. Most data mining queries are compute-intensive and parameterized. Thus, we explore the parameter space for data mining, i.e., in particular, for online association mining.

9. CONCLUSION

We present our **PARAS** framework for fast online association mining. We propose a novel parameter space model for prestoring rules such that a near real-time performance is guaranteed for online mining queries. In the context of the parameter space, we achieve surprisingly compact redundancy abstraction at preprocessing time, such that both space complexity of our proposed PSpace index and the online query processing cost are greatly reduced. In a variety of tested cases, **PARAS** outperforms the four competitor techniques, each by several orders of magnitude.

10. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. A new approach to online generation of association rules. *IEEE Trans. Knowl. Data Eng.*, 13(4):527–540, 2001.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[3] A. Asuncion and D. Newman. UCI ML repository. http://www.ics.uci.edu/ mlearn/MLRepository.html, 2007.

[4] C. Borgelt. Efficient apriori, eclat & fp-growth. http://www.borgelt.net.

[5] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *SIGMOD Rec.*, 26(2):265–276, 1997.

[6] S. Chaudhuri, H. Lee, and V. R. Narasayya. Variance aware optimization of parameterized queries. In *SIGMOD*, pages 531–542, 2010.

[7] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *PVLDB*, 2(1):1246–1257, 2009.

[8] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

[9] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. VLDB '95, pages 420–431, 1995.

[10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[11] M. Kaya and R. Alhajj. Online mining of fuzzy multidimensional weighted association rules. *Applied Intelligence*, 29:13–34, 2008.

[12] M. Kubat, A. Hafez, V. V. Raghavan, J. R. Lekkala, and W. K. Chen. Itemset trees for targeted association querying. *IEEE Trans. Knowl. Data Eng.*, 15(6):1522–1534, 2003.

[13] X. Lin, A. Mukherji, E. A. Rundensteiner, C. Ruiz, and M. O. Ward. Paras. http://users.wpi.edu/ mukherab/Dissertation/WPI-TR.pdf, 2012.

[14] C. Lucchese, S. Orlando, R. Perego, and F. Silvestri. Webdocs: a real-life huge transactional dataset. In *FIMI'04*, 2004.

[15] B. Nag, P. M. Deshpande, and D. J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In *KDD*, pages 244–253, 1999.

[16] R. Ng, L. V. S. Lakshmanan, J. Han, and T. Mah. Exploratory mining via constrained frequent set queries. *SIGMOD Rec.*, 1999.

[17] S. Sahar. Interestingness preprocessing. In *ICDM*, pages 489–496, 2001.

[18] Wikipedia. Association rule mining wikipage. http://en.wikipedia.org/wiki/Association_rule_learning.

[19] T. Wu, Y. Chen, and J. Han. Association mining in large databases: A re-examination of its measures. In *PKDD*, pages 621–628, 2007.

[20] Xmdvtool home page. http://davis.wpi.edu/ xmdv/, July 2012.

[21] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD*, 1997.