The 3W Model and Algebra for Unified Data Mining

Theodore Johnson
AT&T Labs–Research
johnsont@research.att.com

Laks V. S. Lakshmanan
Concordia University and IIT–Bombay
laks@it.iitb.ernet.in

Raymond T. Ng
University of British Columbia
rng@cs.ubc.ca

Abstract

Real data mining/analysis applications call for a framework which adequately supports knowledge discovery as a multi-step process, where the input of one mining operation can be the output of another. Previous studies, primarily focusing on fast computation of *one* specific mining task at a time, ignore this vital issue.

Motivated by this observation, we develop a unified model supporting all major mining and analysis tasks. Our model consists of three distinct worlds, corresponding to intensional and extensional dimensions, and to data sets. The notion of dimension is a centerpiece of the model. Equipped with hierarchies, dimensions integrate the output of seemingly dissimilar mining and analysis operations in a clean manner.

We propose an algebra, called the *dimension algebra*, for manipulating (intensional) dimensions, as well as operators that serve as "bridges" between the worlds. We demonstrate by examples that several real data mining processes can be captured using our model and algebra. We demonstrate the naturality of the algebra by establishing several identities. Finally, we discuss efficient implementation of the proposed framework.

1 Introduction

Data mining studies can be classified broadly into two "generations". Studies in the first generation have focused primarily on which kinds of patterns to mine, and how fast they can be computed. Examples include associations and variants [3, 4, 20, 14, 8, 22, 29], clustering [24, 31, 2, 5], decision trees [27], and depth contours [18]. Recognizing that mining would be far more effective if not considered in isolation, studies in the second generation have focused on how mining can interact with other "components" in a more general framework of knowledge discovery (KDD). One component is the underlying DBMS. Studies such as [9, 28, 30] explore how association mining can handshake with the DBMS most effectively. Another component is the human analyst. Some studies such as [25, 10, 21], allow the user to express a focus for mining via constraints. Other studies such as [15, 1] provide the user with online feedback, and permit him to make dynamic changes to the parameters of computation.

All previous studies are almost always geared toward one mining task at a time. In real data mining applications, KDD is *rarely a one-shot activity*. Rather, it is a *multi-step process* involving different mining operations, data partitioning, aggregation, and data transformations. Thus, previous work fails to address the fundamental need of *supporting KDD as a multi-step process*. In the following examples, we illustrate several multi-step scenarios, extracted from real mining applications, which call for the ability to manipulate (e.g., analyze, query, transform) the results of mining tasks, making the output of one mining operation the input to another.

Example 1 (Associations and Decision Trees) Suppose an analyst analyzes the sales data of a chain store to determine which items were co-purchased with a certain promotional item p, generating a collection of frequent sets. As part of his exploration, he decides to roll up this collection of frequent sets from specific items (e.g., specific brands of meat products) to kinds of items (e.g., the general class of meat). He then wishes to determine the "circumstances" (e.g., `location`, `time`, etc.) under which the frequent co-

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

purchases were made. He does so by constructing a decision tree. The decision tree, when combined with frequent sets, might reveal interesting patterns such as "in northern New Jersey, meat products (not dairy products) are often bought together with p, whereas in southern New Jersey, dairy products (not meat products) are often bought together with p." This example illustrates interesting observations/patterns that can *only* be discovered by freely combining the outcomes of different mining tasks. ■

Example 2 (Stacking Decision Trees) Suppose T_1 is a decision tree that classifies customers in New Jersey into the categories of `highRisk` and `lowRisk` for credit rating. Let T_2 be a decision tree that predicts under what conditions people in New Jersey live in cities vs. the countryside. The analyst may want to combine the two decision trees so as to be able to predict under what conditions people have a certain credit rating and tend to live in a certain neighborhood.[1] One option is to take a cross product between T_1 and T_2. An alternative is to "stack" T_2 below T_1, i.e. each leaf of T_1 is further classified on the basis of T_2 (see Figure 1). Such a classification may be further analyzed, e.g., used as a basis of a group-by. ■

Example 3 (Computing Special Regions) Consider a sales data warehouse with measures like `revenue`, `profit`, and dimensions[2] such as `part`, `time`, `location`. The dimensions may have associated hierarchies. For example, a "region" such as `location` = 'quebec/montreal' may be a child of the region `location` = 'quebec'. Suppose the analyst wants to find the minimal regions which satisfy some aggregate property $\mathcal{P}$, e.g., $\mathcal{P}$ ≡ "the total sales exceeds $100,000", where minimality means children of the region do not satisfy $\mathcal{P}$. The analyst might similarly want to find regions whose sales are significantly different from their siblings. ■

A key aspect exhibited by the above examples is that the data set/space is split by data mining/analysis operations into (possibly overlapping) subsets or "regions". While the above examples, chosen for the familiarity of most readers, focus on associations, decision trees, and group-bys, similar examples can be drawn based on most known mining tasks like data cubes, data spheres, histograms, and clusters (Table 1 has more details). The main point of the above examples is that *interesting and powerful applications can be supported by allowing different mining, analysis, and aggregation tasks to be combined at will.* This is what a multi-step process for KDD is all about. And this is the kind of *unified* environment in which an analyst would like to operate for real mining applications.

However, such an environment does not exist (yet!). In fact, many of these tasks are supported in isolation by different software packages which may not even share the same data format (ascii, binary, etc.). More fundamentally, there is no *common foundation* for all these tasks to be interfaced with one another, not to mention the difficulty in feeding the output of one operation as input to another. Our contributions, set in this context, answer the following questions:

- **What are the "right" structures for unified mining?:** To ensure the compositionality and closure properties of mining operations, we model and manipulate *regions* and *their descriptions* as first-class objects. We call a set of related regions a *dimension*. Our 3W model consists of three different worlds: the *intensional dimension* world (I-World), the *extensional dimension* world (E-World), and the *data* world (D-World). (See Figure 2.) In the I-World, each region is represented in its intensional form, i.e., as a *description* of its members (Section 2). In the E-World, each region is represented in its extensional form, i.e., by an explicit *enumeration* of its member tuples w.r.t. a given data set (Section 4). The D-World consists of raw data, e.g., in the form of relations, from which regions and dimensions can be created as a result of mining.

- **How to manipulate the structures?:** In each of the three worlds, the structures can be manipulated with an algebra of choice. A significant contribution here is the *dimension algebra* that we develop for the I-World. It forms the key tool for linking various data mining tasks and for applying them in cascade (Section 3). We illustrate the expressive power of the algebra by showing how the key steps in Examples 2 and 3 can be captured using expressions in the dimension algebra.

- **How to move in and out of the worlds?:** Having created the three worlds, we establish the following *"bridges"* between them: the *mine* (μ), *populate* (α), *lookup* (λ) operators, and a "macro" called *refresh* (γ) (Section 5). We show how real multi-step data mining and analysis tasks outlined in earlier examples can be fully captured using expressions involving operators of the dimension algebra, and the various operators mentioned above.

- **How good/natural is the model?:** We establish numerous identities involving the dimension algebra and the other operators, thus establishing their naturality. Furthermore, we argue by examples that the model and the operators are expressive enough to support multi-step mining activ-

[1] The training data that led to the two trees may be presently unavailable to the analyst, or doing the combined classification from scratch may take too long for his purpose.

[2] As we will show later, the dimensions of warehouses can also be naturally modeled using the notion of dimensions proposed in this paper.

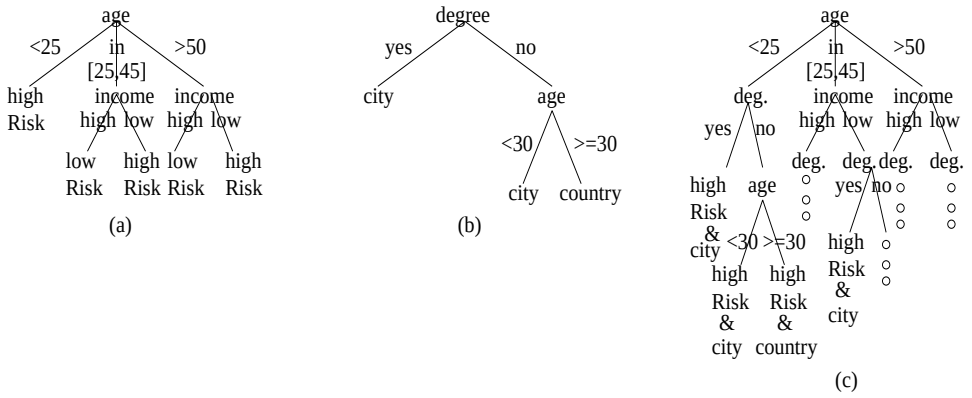

Figure 1: Stacking Decision Trees

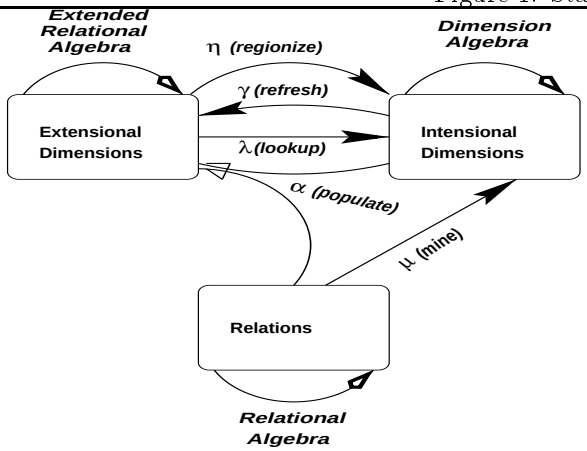

Figure 2: The Different Worlds in the 3W Model, and the Bridges

Task	Shape of Regions	Hierarchies
Dec. Trees	Isothetic	Tree
Freq. Sets	Isothetic	Powerset Lattice
Log-lin. Models	Isothetic	Range Hierarchy
Depth Contours	Convex	Range Hierarchy
Data Spheres	Convex	Range Hierarchy
Data Cubes	Isothetic	Categorical
Histograms	Isothetic	Range Hierarchy
Clusters	can be Non-convex	Adjacent Neighbors

Table 1: Common Data Mining Tasks and the Nature of Their Split Regions

ities that could not be done before (Sections 3 and 5).

- **How well can the model be implemented?:** Many of these identities enable optimization via query rewriting. In addition, we show that incremental computation is possible for certain operations. In many cases, existing spatial indexing and processing techniques can be applied. (See Section 6.)

2 Regions, Dimensions, and Hierarchical Domains: The I-World

We begin with the question of what the "right" structures should be in a unified mining/analysis model. Our proposed structures center around three key notions – regions, dimensions and hierarchical domains.

2.1 Intuition and Significance

As shown in Examples 1-3, decision trees, data cubes, associations, etc. all serve to split a given data set into a collection of (possibly overlapping) subsets of tuples, i.e., points in the space spanned by the data, which we call "regions". Indeed, most well-known mining and analysis operations effectively do a data space splitting, and create regions to achieve their ends (see Table 1). The regions created vary on their spatial shapes and on whether they overlap. For a majority of the tasks shown, the regions created are axis-parallel and hyper-rectangular. Such regions are called *isothetic*, and admit very efficient manipulation and processing.

The cube operator proposed in [11] at once splits up a data set according to all possible group-bys and computes the required aggregate measure for each group (in our terminology, region). However, for the purpose here, by data cube, we mean the operation of splitting the data set into regions. Computation of the aggregate can always be done by an explicit invocation of an aggregate operation.

Intuitively, a dimension is a set of related regions. With dimensions, an analyst can manipulate sets of regions, often creating new ones. In Example 1, the set of itemsets with support exceeding a given threshold forms a dimension. Furthermore, the set of itemsets containing the promotional item p forms another. The set of frequent sets sought in Example 1 is then the intersection of the two dimensions above.

Example 3 shows that dimensions come with interesting structure that relates their constituent regions in the form of a hierarchy. Examples include categorical hierarchies in data warehouses, set inclusion for frequent sets, and more generally, constraint implication for dimensions produced by most mining tasks.

Queries comparing a region to its children, ancestors, siblings, etc. can add significant value to the mining/analysis exercise. To support this, we include hierarchies as an integral part of our 3W model. Table 1 shows the hierarchies associated with dimensions produced by popular mining/analysis tasks.

2.2 Formal Definitions

We next formalize the above notions, first focusing on the I-World (intensional dimensions). Some of these notions assume a different form in the E-World (Section 4).

Definition 1 (Hierarchial Domains) A *hierarchical domain* is any non-empty set H on which the following predicates are defined: (i) equality, interpreted as syntactic identity as usual, and (ii) the predicates $<$ and $\ll$, interpreted as binary relations over H. We require that the graph $(H, <)$ form a DAG, and that $\ll$ be the transitive closure of $<$. ■

For $x, y \in H$, whenever $x < y$ (resp., $x \ll y$), we say y is a *child* (resp., *descendant*) of x. We abbreviate $x < y \ \vee \ x = y$ as $x <= y$ and $x \ll y \ \vee \ x = y$ as $x \ll= y$. We say x is an upper bound of y, z whenever $x \ll= y$ and $x \ll= z$. In this paper, we shall assume that hierarchies form either a tree or a lattice. The notion of a least upper bound is well defined on such hierarchies, in the sense that it exists and is unique. In particular, there is a greatest element, denoted *all*. Elements of a hierarchical domain are called hierarchical values. For an attribute A, $dom(A)$ denotes its domain. Attributes whose domains are hierarchical are called hierarchical attributes.

Definition 1 makes precise the notion of a hierarchy in an abstract sense. Here are some concrete and common examples of hierarchies. For the frequent set computation of Example 1, the lattice of all possible subsets of the set of items is a hierarchical domain. Similarly, given a range $[A, B]$ (for a numeric attribute), the set of all subranges of $[A, B]$ forms a hierarchy called the *range hierarchy*, with range containment acting as the $\ll=$ predicate. For decision trees, conjunctions of inequalities form a hierarchy (modulo equivalence) with (reverse) implication acting as $\ll=$. A hierarchical domain is thus an abstraction of such concrete hierarchies found in practice.

In the I-World, a region is represented in the form of descriptions of its members, i.e., region membership criteria. To capture this, we use constraints. The constraints could involve either the attributes of the data sets to be analyzed, or attributes which are computed from those of the given data sets. *In this first paper, we restrict attention to attributes of the base data set.* We sometimes refer to these attributes as coordinates, to emphasize their spatial nature.

Definition 2 (Constraints) Let $\vec{A} = \{A_1, ..., A_n\}$ be the attributes of a given data set (e.g., a relation). Then by $\text{LINEAR}(\vec{A})$, we denote the class of linear inequalities with real coefficients over the data set attributes $\vec{A}$. ■

Regions produced by most mining/analysis operations are convex, which can be modeled as a set of linear inequalities. Non-convex regions can be modeled as a "union" of multiple convex regions.

Definition 3 (Constraint Attributes) A *constraint attribute*, denoted as RDF_i, is a special hierarchical attribute, whose domain is a set of constraints drawn from conjunctions over $\text{LINEAR}(\vec{A})$, $\vec{A}$ being the data set attributes. Each constraint in $dom(\text{RDF}_i)$ is a conjunction of constraints of the form: $linear(A_1, \ldots, A_n) \geq c$, where c is a constant, and $A_1, \ldots, A_n$ are attributes of the given data set. For a tuple t from the given data set, we say that t *satisfies* $linear(A_1, \ldots, A_n) \geq c$ iff the inequality $linear(t[A_1], \ldots, t[A_n]) \geq c$ evaluates to true. Satisfaction generalizes to a set/conjunction of of constraints as usual.[3] A constraint $\mathcal{C}$ drawn from the domain of a constraint attribute is called a *region description formula* (RDF). ■

As an example, for the decision tree of Figure 1(a) and Example 2, the first `highRisk` region is described by the constraint `age < 25`, and the second `highRisk` region by the constraint `(25 ≤ age ≤ 45) & (income = low)`. Both of these regions are isothetic. As shown in Table 1, there are some commonly used mining tasks that produce regions that are more complex than isothetic, and require their membership criteria to be captured as general linear constraints (e.g. depth contours, clusters, etc.).

Definition 4 (Region Identifiers) Let RDF_i be a constraint attribute. We say that a hierarchical attribute RID_i is the *region identifier* (RID) attribute associated with RDF_i, provided the domains of RDF_i and RID_i (together with the hierarchy predicates) are isomorphic, i.e., there is a 1-1 onto function $desc : dom(\text{RID}_i) \rightarrow dom(\text{RDF}_i)$, such that $\forall h, h' \in dom(\text{RID}_i) : h \ll= h'$ iff $desc(h') \Rightarrow desc(h)$. Furthermore, for equivalent constraints $\mathcal{C}, \mathcal{C}'$, $h^{-1}(\mathcal{C}) = h^{-1}(\mathcal{C}')$. ■

Note that for every constraint attribute RDF, we can always postulate a corresponding RID attribute RID. RID attributes are similar in spirit to the notion of hierarchical attributes introduced by Jagadish et al. [17] in a different context. The intuition is that the RID_i attribute, associated with a given constraint attribute

[3] We denote the empty conjucntion as the formula *true*, while *false* is a representative member of the equivalence class of unsatisfiable constraints, e.g., $A < c \ \wedge \ A \geq c$.

RDF_i, allows us to encode the hierarchy relationships such as $<$, $\ll$. For instance, constraints for frequent sets could be of the form `beer` $= 1 \wedge$ `diaper` $= 1$, and they could be encoded using bit vectors as RIDs. The encoding via RIDs serves two purposes. First, in the E-world (Section 4), RIDs will be used as group identifiers to encode the membership of data tuples in specific regions. Second, as will be detailed in Section 6, RIDs enable efficient checking of hierarchy relationships, offering a fast alternative to the expensive implication checking of constraints. Thus, RID attributes help quickly detect hierarchy relatives of specified regions, similar to indices. Below, $\mathcal{V}$ denotes the set of all possible values – both hierarchical and non-hierarchical.

Definition 5 (Regions) A *dimension schema* is a set of attributes $\mathbf{A} = (\text{RID}_1, ..., \text{RID}_m, \text{RDF}_1, ..., \text{RDF}_m, P_1, ..., P_l)$, where RDF_i is a constraint attribute and RID_i is the corresponding RID attribute, $1 \leq i \leq m$. A *region* over $\mathbf{A}$ is a partial function $r : \mathbf{A} \rightarrow \mathcal{V}$ such that: (i) whenever $r(A)$ is defined, $A \in \mathbf{A}$, we have $r(A) \in dom(A)$, and (ii) $r(\text{RID}_i)$ is defined iff $r(\text{RDF}_i)$ is. ■

From now on, we represent the RID and RDF attributes of a dimension schema as the vectors $\vec{\text{RID}}$ and $\vec{\text{RDF}}$. For a region r, we call $r[\vec{\text{RID}}]$ as its RID-value, or simply the RID. A few important remarks are in order.

- The key reason why multiple RDF attributes are allowed is to simplify the process of factoring dimensions.
- There can be other attributes/properties associated with a region. We denote these by the P_i's in the above definition, and refer to them as the *property attributes*. For the decision tree example in Figure 1(a), P_1 may be the `decision-label` attribute indicating whether the region is `highRisk` or `lowRisk`.
- Whenever a region r is undefined over a constraint attribute RDF_i, the intended semantics is that $r(\text{RDF}_i) = true$. This corresponds to $r(\text{RID}_i) = all$.

Definition 6 (Intensional Dimension Instance) An *instance* $\mathcal{D}$ of a dimension schema $\mathbf{A}$ is a set of regions over $\mathbf{A}$. ■

For the example in Figure 1(a), $\mathcal{D}$ consists of eight regions, corresponding to the eight nodes of the tree, i.e., 5 leaf nodes and 3 non-leaf nodes. Notice that the above definition does not insist that all regions in an instance be defined over the same set of attributes. For instance, for the non-leaf regions, the `decision-label` could be undefined. Thus, a dimension instance can be a heterogeneous collection of regions.

3 The Dimension Algebra

Having defined regions and dimensions, we next address their manipulation in the I-World. Examples 1-3 demonstrate how manipulations of dimensions could add significant value to data mining and analysis. Thereto, we propose the *dimension algebra*. Some operators in the algebra are similar to those in the relational algebra, while others are very different. We start with the latter ones.

3.1 The Selection Operator

The basic idea is to allow selections that invoke various spatial predicates such as overlap, containment, etc. between regions. Any set of predicates that is closed under negation may be chosen for this purpose. The choice will impact the computations expressible in the algebra. For the sake of concreteness, we assume the spatial predicates are overlap ($\|$), containment ($\supset$), disjointness ($\nparallel$), and non-containment ($\not\supset$). In addition, for a property attribute P, we let $pred(P)$ be any predicate such as $P\ \theta\ v$, where θ is $=, \leq$, etc. (Allowable predicates on Ps could also include comparison of two property attributes. We suppress the obvious detail here.)

Definition 7 (Selection) Let $\mathcal{D}$ be a dimension instance over the schema $\mathbf{A}$, $\text{RDF}_a, \text{RDF}_b$ be constraint attributes, $\mathcal{C}$ be a constant constraint drawn from the domain of RDF_a, and $\circ$ be one of the spatial selection predicates mentioned above. Furthermore, let RID_a and RID_b be the corresponding RID attributes in $\mathbf{A}$, $h \in dom(\text{RID}_a)$, and $\triangle$ be any of the hierarchical predicates introduced in Definition 1. Finally, let P be a property attribute in $\mathbf{A}$, and $pred(P)$ be a predicate involving P. We define:

$\sigma_{\text{RDF}_a \circ \mathcal{C}}(\mathcal{D}) = \{r \mid r \in \mathcal{D}\ \&\ r[\text{RDF}_a] \circ \mathcal{C}$ is true $\}$.
$\sigma_{\text{RDF}_a \circ \text{RDF}_b}(\mathcal{D}) = \{r \mid r \in \mathcal{D}\ \&\ r[\text{RDF}_a] \circ r[\text{RDF}_b]$ is true $\}$.
$\sigma_{\text{RID}_a \triangle \text{RID}_b}(\mathcal{D}) = \{r \mid r \in \mathcal{D}\ \&\ r[\text{RID}_a] \triangle r[\text{RID}_b]$ is true $\}$.
$\sigma_{\text{RID}_a \triangle h}(\mathcal{D}) = \{r \mid r \in \mathcal{D}\ \&\ r[\text{RID}_a] \triangle h$ is true $\}$.
$\sigma_{pred(P)}(\mathcal{D}) = \{r \mid r \in \mathcal{D}\ \&\ pred(r[P])$ is true $\}$. ■

Let $\mathcal{D}$ be the dimension consisting of the eight regions/nodes of the decision tree shown in Figure 1(a), with one RDF attribute RDF_a. The expression $\sigma_{\text{RDF}_a \| (25 \leq age)}(\mathcal{D})$ finds all the regions that overlap with the constant region $25 \leq$ `age`. Similarly, $\sigma_{\texttt{decision-label}=highRisk}(\mathcal{D})$ identifies all `highRisk` regions.

Selection involving boolean combination of conditions should be obvious. A subtle point, however, is that the selection operator as defined above, constrains individual constraint attributes. In practice, we might be interested in constraining the region "bounded" by several constraint attributes. Here, note

$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \| \text{RDF}_c}(\mathcal{D}) = \sigma_{(\text{RDF}_a \| \text{RDF}_c) \wedge (\text{RDF}_b \| \text{RDF}_c)}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \| \mathcal{C}}(\mathcal{D}) = \sigma_{(\text{RDF}_a \| \mathcal{C}) \wedge (\text{RDF}_b \| \mathcal{C})}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \supset \text{RDF}_c}(\mathcal{D}) = \sigma_{(\text{RDF}_a \supset \text{RDF}_c) \wedge (\text{RDF}_b \supset \text{RDF}_c)}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \supset \mathcal{C}}(\mathcal{D}) = \sigma_{(\text{RDF}_a \supset \mathcal{C}) \wedge (\text{RDF}_b \supset \mathcal{C})}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \not\| \text{RDF}_c}(\mathcal{D}) = \sigma_{(\text{RDF}_a \not\| \text{RDF}_c) \vee (\text{RDF}_b \not\| \text{RDF}_c)}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \not\| \mathcal{C}}(\mathcal{D}) = \sigma_{(\text{RDF}_a \not\| \text{RDF}_c) \vee (\text{RDF}_b \not\| \mathcal{C})}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \not\supset \text{RDF}_c}(\mathcal{D}) = \sigma_{(\text{RDF}_a \not\supset \text{RDF}_c) \vee (\text{RDF}_b \not\supset \text{RDF}_c)}(\mathcal{D})$.
$\sigma_{(\text{RDF}_a \& \text{RDF}_b) \not\supset \mathcal{C}}(\mathcal{D}) = \sigma_{(\text{RDF}_a \not\supset \mathcal{C}) \vee (\text{RDF}_b \not\supset \mathcal{C})}(\mathcal{D})$.

Table 2: A Sample of Identities for the Selection Operator. See Theorem 1 for Precise Assumptions.

that for two constraints $\mathcal{C}_1, \mathcal{C}_2$, the region bounded by $\mathcal{C}_1$ and $\mathcal{C}_2$ is captured by their conjunction $\mathcal{C}_1 \& \mathcal{C}_2$. (We will denote the conjunction of selection conditions Cond_1 and Cond_2 as $\sigma_{\text{Cond}_1 \wedge \text{Cond}_2}$ to avoid confusion.) More precisely, let $\text{RDF}_a, \text{RDF}_b, \text{RDF}_c$ be any constraint attributes of a dimension schema $\mathbf{A}$ with instance $\mathcal{D}$. Then $\sigma_{(\text{RDF}_a \& \text{RDF}_b) \circ \mathcal{C}}(\mathcal{D})$ and $\sigma_{(\text{RDF}_a \& \text{RDF}_b) \circ \text{RDF}_c}(\mathcal{D})$ are defined in the obvious way. The following theorem shows under certain circumstances this complex selection reduces to the simpler one defined above. Similar identities, for boolean combination of selection conditions, are suppressed for brevity.

Theorem 1 Let $\mathcal{D}$ be an instance of the dimension schema $\mathbf{A}$. Let $\text{RDF}_a, \text{RDF}_b, \text{RDF}_c$ be the constraint attributes in $\mathbf{A}$, and $\mathcal{C}$ be a constant constraint drawn from some constraint attribute domain. Then identities (3)-(4) and (7)-(8) listed in Table 2 hold. Furthermore, whenever all constraints are isothetic and $\text{RDF}_a \| \text{RDF}_b$ is true, identities (1)-(2) and (5)-(6) hold. ■

The identities are non-trivial: for instance, without the isothetic requirement, identities (1)-(2) and (5)-(6) do not always hold. Suppose $\text{RDF}_a, \text{RDF}_b$ are any overlapping axis-parallel rectangles. Unless RDF_c is also isothetic, it could easily overlap the regions RDF_a & $\neg\text{RDF}_b$ and RDF_b & $\neg\text{RDF}_a$, without overlapping RDF_a & RDF_b. On the other hand, we can show that when all three are isothetic, this cannot happen.

3.2 The Projection Operator

Elimination of property attributes is classical and trivial. On the other hand, elimination of constraint attributes must ensure a consistency property in that the corresponding RID attributes should be automatically eliminated.

Definition 8 (Projection) Let $\mathbf{A}$ be a dimension schema,
$\mathcal{D}$ an instance of $\mathbf{A}$, and $\text{RDF}_i, ..., \text{RDF}_j$, $P_k, ..., P_\ell$ a subset of $\mathbf{A}$ consisting of constraint attributes and property attributes. Then $\pi_{\text{RDF}_i, ..., \text{RDF}_j, P_k, ..., P_\ell}(\mathcal{D}) = \{r[\text{RID}_i, ..., \text{RID}_j, \text{RDF}_i, ..., \text{RDF}_j, P_k, ..., P_\ell] \mid r \in \mathcal{D}\}$.

Intuitively, eliminating a constraint attribute RDF_i amounts to setting this value to *true* in every region. Similarly, the value of the eliminated RID attribute is essentially *all* in every region.

3.3 The Purge Operator: Generalized Duplicate Elimination

A region is inconsistent provided the conjunction of the RDF formulas defining it (i.e., its description) cannot be satisfied by any data point. For instance, a region r with $r[\text{RDF}_1]$ = "$A \leq 2$" and $r[\text{RDF}_2]$ = "$A \geq 3$" is clearly inconsistent Starting with only consistent regions, our operators can create an inconsistent region. Cartesian product (Section 10) is an example of such an operation, since each of two regions may be consistent by themselves, but not their conjunction (e.g., the third leaf from the left in Figure 1 c). The *purge* operator, defined next, removes inconsistent regions.

Definition 9 (Purge) Let $\mathcal{D}$ be any dimension over the schema $\mathbf{A} = (\text{RID}_1, ..., \text{RID}_m, \text{RDF}_1, ..., \text{RDF}_m, P_1, ..., P_\ell)$. Then the *purge* of $\mathcal{D}$ is defined by $\Upsilon(\mathcal{D}) = \{r \mid r \in \mathcal{D} \ \& \ \bigwedge_{1 \leq i \leq m} r[\text{RDF}_i] \text{ is satisfiable } \}$. ■

3.4 Other Operators

The operators introduced so far deal with the spatial aspect of regions and dimensions. Below we discuss the remaining operators, analogous to their relational counterparts, that deal with the set/relation aspect of dimensions.

Definition 10 (Cartesian product) Let $\mathcal{D}_1, \mathcal{D}_2$ be instances of two dimension schemas $\mathbf{A}_1, \mathbf{A}_2$ respectively. For simplicity, assume the attributes in the schemas $\mathbf{A}_1, \mathbf{A}_2$ are distinct. Then $\mathcal{D}_1 \times \mathcal{D}_2 = \{(r_1, r_2) \mid r_i \in \mathcal{D}_i, i = 1, 2\}$, with schema $\mathbf{A}_1 \cup \mathbf{A}_2$. ■

From a spatial perspective, cartesian product creates new regions which are obtained by taking the pairwise intersection of regions in the two dimension instances. The next operator is union. The main distinction with classical union is the absence of a union-compatibility requirement. This permits a fully heterogeneous union, and hence dimensions that are heterogeneous collections of regions. Recall, whenever a region is not defined on a certain RID attribute, we treat it as the top element *all* of $dom(\text{RID})$, whereas when it is not defined on a certain RDF attribute, we treat it as the *true* constraint in $dom(\text{RDF})$.

Definition 11 (Union) Let $\mathbf{A}_1, \mathbf{A}_2$ be two dimension schemas and $\mathcal{D}_1, \mathcal{D}_2$ corresponding instances. Then the union of these dimensions has the schema $\mathbf{A}_1 \cup \mathbf{A}_2$. The instance is defined by $\mathcal{D}_1 \cup \mathcal{D}_2 = \{r \mid r \in \mathcal{D}_1 \ \vee \ r \in \mathcal{D}_2\}$. ■

We say that two regions r_1, r_2 are equivalent provided the conjunctions of the constraint formulas of their RDF attributes are equivalent, i.e., $r_1[\vec{\text{RDF}}] \equiv r_s[\vec{\text{RDF}}]$.

$\sigma_{Cond}(\pi_{\text{RDF}_i,...,\text{RDF}_j,P_k,...,P_\ell}(\mathcal{D})) =$
$\pi_{\text{RDF}_i,...,\text{RDF}_j,P_k,...,P_\ell}(\sigma_{Cond}(\mathcal{D}))$, whenever Cond involves only the attributes $\text{RID}_i, ..., \text{RID}_j, \text{RDF}_i, ..., \text{RDF}_j, P_k, ..., P_\ell$.

$\sigma_{Cond}(\mathcal{D}_1 \times \mathcal{D}_2) = \sigma_{Cond}(\mathcal{D}_1) \times \mathcal{D}_2$, whenever Cond does not involve any of the attributes of $\mathcal{D}_2$.

$\pi_{\text{RDF}_p,...,\text{RDF}_q,\text{RDF}_i,...,\text{RDF}_j,P_a,...,P_b,P_c,...,P_d}(\mathcal{D}_1 \times \mathcal{D}_2) = \pi_{\text{RDF}_p,...,\text{RDF}_q,P_a,...,P_b}(\mathcal{D}_1) \times \pi_{\text{RDF}_i,...,\text{RDF}_j,P_c,...,P_d}(\mathcal{D}_2)$, where $\text{RDF}_p, ..., \text{RDF}_q, P_a, ..., P_b$ are in $\mathbf{A}_1$ but not in $\mathbf{A}_2$ and $\text{RDF}_i, ..., \text{RDF}_j, P_c, ..., P_d$ are in $\mathbf{A}_2$ but not in $\mathbf{A}_1$.

σ and π distribute over $\cup$.

σ distributes over $-$.

$\pi_{\text{RDF}_p,...,\text{RDF}_q,P_i,...,P_j}(\mathcal{D}_1) - \pi_{\text{RDF}_p,...,\text{RDF}_q,P_i,...,P_j}(\mathcal{D}_2) \subseteq \pi_{\text{RDF}_p,...,\text{RDF}_q,P_i,...,P_j}(\mathcal{D}_1 - \mathcal{D}_2)$.

$\sigma_{\text{Cond}_1 \& \text{Cond}_2}(\mathcal{D}) = \sigma_{\text{Cond}_1}(\mathcal{D}) \cap \sigma_{\text{Cond}_2}(\mathcal{D})$.

$\sigma_{\text{Cond}_1 \& \neg \text{Cond}_2}(\mathcal{D}) = \sigma_{\text{Cond}_1}(\mathcal{D}) - \sigma_{\text{Cond}_2}(\mathcal{D})$.

$\sigma_{\text{Cond}_1 \vee \text{Cond}_2}(\mathcal{D}) = \sigma_{\text{Cond}_1}(\mathcal{D}) \cup \sigma_{\text{Cond}_2}(\mathcal{D})$.

Table 3: A Sample of Identities Satisfied by the Operators

Definition 12 (Minus) Let $\mathbf{A}_1, \mathbf{A}_2$ be any dimension schemas and $\mathcal{D}_1, \mathcal{D}_2$ corresponding instances. Then $\mathcal{D}_1 - \mathcal{D}_2 = \{r \mid r \in \mathcal{D}_1 \ \& \ \nexists s \in \mathcal{D}_2 : s \text{ is equivalent to } r\}$.

Note that just like the union operator, minus does not require union-compatibility between its operands. The result of minus contains exactly those regions in $\mathcal{D}_1$ for which no equivalent region exists in $\mathcal{D}_2$. Thus, the schema of the result is $\mathbf{A}_1$. A natural question is why the minus operator is defined in terms of region equivalence, instead of, e.g., region containment. The next lemma below settles this question, justifying our choice of primitive operators.

Lemma 1 Let $\mathcal{D}_1, \mathcal{D}_2$ be dimension instances of schemas $\mathbf{A}_1, \mathbf{A}_2$, where $\mathbf{A}_i = (\vec{\text{RID}}_i, \vec{\text{RDF}}_i, \vec{P}_i)$. Let $\ominus$ be defined as $\mathcal{D}_1 \ominus \mathcal{D}_2 = \{r \mid r \in \mathcal{D}_1 \ \& \ \nexists s \in \mathcal{D}_2 : r \text{ is contained in } s\}$. Then $\ominus$ can be simulated using the minus operator. Specifically, $\mathcal{D}_1 \ominus \mathcal{D}_2 = \mathcal{D}_1 - \pi_{\vec{\text{RID}}_1, \vec{\text{RDF}}_1, \vec{P}_1}(\sigma_{\vec{\text{RDF}}_2 \supset \vec{\text{RDF}}_1}(\mathcal{D}_1 \times \mathcal{D}_2))$. But the minus operator cannot be simulated using $\ominus$ and the remaining operators. ■

Finally, there is the *rename* operator, denoted as $\rho_{B \leftarrow A}(\mathcal{D})$, that renames attribute A of dimension $\mathcal{D}$ to attribute B, for any attributes A, B. Its definition is similar to that in the relational algebra. We sometimes use $\rho_{\mathbf{A}' \leftarrow \mathbf{A}}(\mathcal{D})$ as an abbreviation of the cascade of rename operators which rename each attribute $A \in \mathbf{A}$ to its primed version A'.

3.5 Expressiveness and Naturality of the Operators

Thus far, we have completed the presentation of the operators in the dimension algebra. The theorem below establishes a number of identities analogous to those for the relational algebra. This theorem is significant in two ways. First, it shows that the definitions of the operators are natural. Second, these identities can be used for rewriting algebraic queries into equivalent and more efficient ones.

Theorem 2 The operators of the dimension algebra satisfy the identities listed in Table 3, where $\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2$ are dimension instances of schemas $\mathbf{A}, \mathbf{A}_1, \mathbf{A}_2$. ■

Example 4 (Revisiting Example 3) In Example 3, the analyst seeks to find a minimal region satisfying some property $\mathcal{P}$, where minimality is defined as a region satisfying $\mathcal{P}$ without any of its children satisfying $\mathcal{P}$. Let $\mathcal{D}$ be a dimension instance consisting of all the regions satisfying property $\mathcal{P}$. ($\mathcal{D}$ itself may be computed as an algebraic expression; but this is not the focus here.) Let $hasChild(\mathcal{D})$ denote those regions in $\mathcal{D}$ such that the region has at least one child also in $\mathcal{D}$. The expression $\bigcup_{1 \le i \le m} \pi_{\mathbf{A}}(\sigma_{\text{RID}_1=\text{RID}'_1 \wedge \cdots \wedge \text{RID}_{i-1}=\text{RID}'_{i-1} \wedge \text{RID}_i < \text{RID}'_i \wedge \text{RID}_{i+1}=\text{RID}'_{i+1} \wedge \ldots \wedge \text{RID}_m=\text{RID}'_m}(\mathcal{D} \times \rho_{\mathbf{A}' \leftarrow \mathbf{A}}(\mathcal{D})))$ precisely computes $hasChild(\mathcal{D})$, where the schema $\mathbf{A}$ contains m RID attributes. This is essentially a "self-join", relying on the hierarchical predicate $<$ defined in Definition 1 to select out those regions with a child in $\mathcal{D}$. The minimal regions in $\mathcal{D}$ that the analyst seeks are found by $\mathcal{D} - hasChild(\mathcal{D})$. It should be obvious that other similar hierarchical relationships can be captured as algebraic expressions, e.g., $hasNoChild(\mathcal{D})$, $hasNoPar(\mathcal{D})$, $hasAnc(\mathcal{D})$. ■

Example 5 (Revisiting Example 2) In Example 2, we motivate stacking decision tree T_2 below tree T_1, i.e., each leaf of T_1 is further classified based on T_2. We leave it to the reader to verify that the expression $hasNoChild(T_1) \times T_2 \ \cup \ hasChild(T_1) \times hasNoPar(T_2)$ indeed computes the stacking of T_1 on T_2. To take full advantage of the fact that there is no union-compatibility requirement, as shown in Definition 11, we can simplify the above expression to $hasNoChild(T_1) \times T_2 \ \cup \ hasChild(T_1)$. The first part deals with stacking T_2 under each leaf node of T_1, whereas the second part simply includes the non-leaf nodes of T_1 in the final result. ■

4 The Other Two Worlds

So far, we have introduced regions and dimensions in the I-World, and described how they can be manipulated. In this section, we turn our attention to the E-World and the D-World.

4.1 The E-World and Extensional Dimensions

In the I-world, regions in a dimension are represented by descriptions of their members. However, often times, an *extensional* representation of a dimension is also valuable. By "extensional", we mean that for a given data set E, the dimension is represented by an explicit enumeration of the tuples belonging to each region.

Definition 13 (Extensional Dimension) An *extensional dimension schema* is a set of attributes $\mathbf{A} = (\text{RID}_1, ..., \text{RID}_m, A_1, ..., A_n)$, where RIDs are hierarchical attributes and the As are normal attributes. An *extensional dimension instance* over $\mathbf{A}$ is a set of tuples over it. ■

A region $r \in \mathcal{E}$ is identified by its RID-value $r[\vec{\text{RID}}]$, and its members are precisely those tuples whose RID equals this value. Note that the As are classical attributes. Thus, an extensional dimension is a generalized partition of a data set (since regions can overlap). For the example illustrated in Figure 1(a), the intensional dimension $\mathcal{D}$ consists of eight regions, corresponding to the eight nodes. For a given (training or test) data set E, the analyst may want to "populate" $\mathcal{D}$ to find out exactly which tuples in E are in each region (based on which he may take further actions). While we will formalize the "populate" operation in Section 5, the result of this operation is an extensional dimension $\mathcal{E}$. Each tuple in E is assigned an appropriate RID in $\mathcal{E}$, which indicates which region in $\mathcal{D}$ (and hence in $\mathcal{E}$) it belongs to.

An intensional dimension schema and an extensional dimension schema are *compatible* provided they have the same set of RID (i.e. hierarchical) attributes. Let $\mathcal{D}$ be an intensional dimension and $\mathcal{E}$ be an extensional dimension such that their schemas are compatible. We say that a tuple $t_r \in \mathcal{E}$ *corresponds* to a region $r \in \mathcal{D}$ provided that both of them agree on their RIDs, i.e., $t_r[\vec{\text{RID}}] = r[\vec{\text{RID}}]$.

For the same reason that we allow intensional dimensions to be manipulated with the dimension algebra, in the E-world, we provide an algebra for the manipulation of extensional dimensions. Note that except for the $\vec{\text{RID}}$ attributes, an extensional dimension is really like an ordinary relation. Thus, an appropriate algebra for the E-World is the relational algebra extended with aggregation and with some modifications. For brevity, we only remark on the adaptations to the extended (relational) algebra here.

Projection might involve any subset of the RID attributes and/or the classical attributes. Intuitively, eliminating an RID attribute via projection amounts to setting the value of that attribute to the top element *all*. Finally, selection might involve any conventional selection predicate over the classical attributes. In addition, as in Definition 7, we also permit selection predicates of the form $\text{RID}_i \triangle \text{RID}_j$ over RID attributes, where $\triangle$ is one of the hierarchical predicates. Join can be simulated as usual via Cartesian product and selection.

4.2 The D-World and Why

Finally, there is the D-World, which can be viewed as consisting of a relational database (i.e., no RID or RDF attribtues). Thus, the natural algebra of choice for this world is the relational algebra (extended with aggregation). The key difference between extensional dimensions in the E-world and relations in the D-world is that the former has a strong region identity, whereas the latter is completely free of it. One may wonder why the latter is worth included in the model after all. From our empirical observation of how a real data mining and analysis process works, an analyst often spends considerable time operating in the D-World *during the entire process*. At the beginning, time may be spent on data integration and data preparation. But even afterwards, considerable time may again be spent in the D-World for data transformation and renewed preparation. By including the D-World in the 3W model, we can more faithfully model the interactions among the worlds, the topic of the next section.

5 Moving In and Out of the Worlds

As discussed above, each of the three worlds has its unique role to play. And the full power of the 3W model is only realized when an analyst is allowed to freely move between the worlds. In this section, we propose the "bridging" operators to facilitate this.

5.1 The Bridging Operators

The first operation amounts to "populating" the regions in an intensional dimension $\mathcal{D}$ with tuples from a given data set E, producing an extensional dimension $\mathcal{E}$. Intuitively, for each region in $\mathcal{D}$, there is a corresponding "extensional" region in $\mathcal{E}$ which contains just those tuples in E that satisfy the region description.

Definition 14 (Populate) Let $\mathcal{D}$ be an intensional dimension with dimension schema $(\text{RID}_1, \ldots, \text{RID}_m, \text{RDF}_1, \ldots, \text{RDF}_m, P_1, \ldots, P_l)$ and E be a relation with schema $(A_1, \ldots, A_n)$, such that the set of (coordinate) attributes included in the constraints in the domains of each RDF_i is a subset of $\{A_1, \ldots, A_n\}$. Then $\alpha(\mathcal{D}, E)$ produces an extensional dimension with schema $(\text{RID}_1, \ldots, \text{RID}_m, A_1, \ldots, A_n, P_1, \ldots, P_l)$, and instance $\alpha(\mathcal{D}, E) = \{t \mid \exists r \in \mathcal{D} : \exists t_r \in E : t[\vec{\text{RID}}] = r[\vec{\text{RID}}] \;\&\; t[\vec{A}] = t_r[\vec{A}] \;\&\; t[\vec{P}] = r[\vec{P}] \;\&\; t_r \text{ satisfies } r[\vec{\text{RDF}}]\}$, where satisfaction is in the sense formalized in Definition 3. ■

For instance, populating the decision tree of Figure 1(a) with a test relation would create an extensional dimension, which can be used to evaluate the accuracy of the decision tree. Similarly, populating a collection of frequent sets with a transaction database would give an extensional dimension with each region containing the set of supporting transactions.

Definition 15 (Mine) The *mine* operation, μ, given a parameter p, maps a relation E to an intensional dimension $\mathcal{D}$, i.e., $\mathcal{D} = \mu(E, p)$.

Typical data mining operations create intensional dimensions. We abstract this in the form of the *mine* operation. Examples of μ include decision tree, frequent sets, data cube, depth contour, etc. We may regard p as a number that specifies whether the desired intensional dimension is a decision tree, a depth contour, or any task mentioned in Table 1. For clarity, we use short strings instead of numbers in our examples. For example, $\mu(E, \mathrm{dc})$ corresponds to the data cube computation. Definition 15 above only defines the type of the mine operator at the level of the model. Its exact definition and computation depend entirely on the specific mining task invoked. In practice, invocation of μ would result in the running of a relevant (fast) mining algorithm. When a mining operation is invoked, we sometimes get both an intensional description of regions and an enumeration of the region members. It is convenient to separate these intensional and extensional aspects of a dimension for purposes of algebraic manipulation (as we have done). The extensional dimension $\mathcal{E}$ corresponding to an intensional dimension mined from a relation E can then be captured via the expression $\alpha(\mu(E, p), E)$.

Operation α allows us to relate a data set (D-world) to an intensional dimension (I-world) and, as a result, it lets us move to the E-world. The mine operator lets us move from the D-world to the I-world. (See Figure 2.) Next, we focus on operators bridging the I- and E-worlds. One important such situation is when the analyst has already obtained an extensional dimension $\mathcal{E} = \alpha(\mu(E, p), E)$, but may wish to "recall" or "look up" which intensional dimension gave rise to $\mathcal{E}$. To capture this, we propose a *lookup* operator λ.

Definition 16 (Lookup) Let $\mathcal{E}$ be any extensional dimension over the schema $(\mathrm{RID}_1, \ldots, \mathrm{RID}_m, A_1, \ldots, A_n)$. Then $\lambda(\mathcal{E}) = \{(t[\mathrm{RID}_1], \ldots, t[\mathrm{RID}_m], desc(t[\mathrm{RID}_1]), \ldots, desc(t[\mathrm{RID}_m])) \mid t \in \mathcal{E}\}$, where $desc(\mathrm{RID})$ is the function that represents the 1-1 correspondence between RIDs and constraints (RDFs), as defined in Definition 4. ■

Whenever the analyst wants to find out the intensional descriptions of regions in $\mathcal{E}$, as opposed to enumerations of their member tuples, λ is used to look up the appropriate hierarchical domain and return the corresponding region description formulas.

Finally, we define the *refresh* macro, below, as a matter of convenience:

Definition 17 (Refresh) Let E be a data set and $\mathcal{D}_1, \ldots, \mathcal{D}_n$ be intensional dimensions based on E. Let $Exp(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ be any expression in the dimension algebra. Then the *refresh* of Exp w.r.t. the data set E is defined as $\alpha(Exp(\mathcal{D}_1, \ldots, \mathcal{D}_n), E)$. ■

5.2 Completing the Examples

The focus of this section is to show that the machinery in this paper is rich enough to support multi-step mining activities in a clean algebraic framework. We do this by following through with the running examples developed in the earlier sections.

Example 6 (Example 3– The Full Story) Let us return to the task set out in Example 3. To begin, there is the data set E. By applying a data cube operator, the analyst gets $\mathcal{D} = \mu(E, \mathrm{dc})$, from which he also gets the corresponding extensional dimension as $\mathcal{E} = \alpha(\mu(E, \mathrm{dc}), E)$. Let $\mathrm{AGG}(\mathcal{E})$ be an expression in extended relational algebra that computes total sales grouped by region ids. (The detail of this standard expression is beside the point here.) Then $\mathcal{E}' = \sigma_{\mathrm{totSales} \geq 100000}(\mathrm{AGG}(\mathcal{E}))$ gives an extensional dimension, containing those regions in $\mathcal{E}$ that grossed a sale over $100,000. To obtain the intensional descriptions of the regions in $\mathcal{E}'$, the analyst performs a lookup with $\mathcal{D}' = \lambda(\mathcal{E}')$. $\mathcal{D}'$ contains all regions satisfying the predicate on sales. To obtain the minimal regions satisfying this property, all the analyst has to do is plug in $\mathcal{D}'$ in place of the term $\mathcal{D}$ in the dimension algebra expression for $hasNoChild(\mathcal{D})$ given in Example 4. This completes the task. ■

Example 7 (Example 1 – The Full Story) To perform the task set out in Example 1 with the transaction data set E, the analyst begins with the frequent set mining, with $\mathcal{D} = \mu(E, \mathrm{fs})$. To restrict the computation to those frequent sets containing the promotional item p, the analyst would instead use the expression $\mathcal{D}' = \sigma_{\mathrm{RDF} \supset \{p\}}(\mathcal{D})$, thus finding constrained frequent sets. He can do the rollup on frequent sets (from items to class of items) by computing the parents of the regions in $\mathcal{D}'$ using Cartesian product and selection based on hierarchy predicates. By populating the resulting dimension w.r.t. E, he obtains an extensional dimension, say $\mathcal{E}$, which shows the transaction sets corresponding to $\mathcal{D}'$. By applying a decision tree construction to $\mathcal{E}$, he then obtains another intensional dimension, say $\mathcal{D}$". This completes the task. ■

A generic remark about our examples is that in order to facilitate the development of the process within the algebra, we have explained it step by step. In practice, however, an analyst who wishes to conduct such a multi-step mining/analysis process, may wish to evaluate complex expressions in one shot. The system can then exploit the properties of the operators involved, in optimizing the computation effectively. A simple example is the computation of constrained frequent sets via $\sigma_{\mathrm{RDF} \supset \{p\}}(\mathcal{D})$ in Example 7. As pointed out in Section 1, many optimized algorithms have been developed for computing constrained frequent patterns in recent years, which can be leveraged. See Section 6 for more details on optimization.

5.3 Inter-World Interactions

Given the importance of the bridge operators in linking the worlds, a natural question is how do the various

$\alpha((\mathcal{D}_1 \cup \mathcal{D}_2), E) = \alpha(\mathcal{D}_1, E) \cup \alpha(\mathcal{D}_2, E)$.
$\alpha(\sigma_{\mathrm{Cond}_1 \wedge \mathrm{Cond}_2}(\mathcal{D}), E) =$
$\alpha(\sigma_{\mathrm{Cond}_1}(\mathcal{D}), E) \cap \alpha(\sigma_{\mathrm{Cond}_2}(\mathcal{D}), E)$.
$\alpha(\sigma_{\mathrm{Cond}_1 \wedge \neg\mathrm{Cond}_2}(\mathcal{D}), E) = \alpha(\sigma_{\mathrm{Cond}_1}(\mathcal{D}), E) -$
$\alpha(\sigma_{\mathrm{Cond}_2}(\mathcal{D}), E)$. Similar identities hold for others boolean combinations.
Let $\mathcal{C}$ be a constraint from the domain of RDF_A and $\mathcal{C}'$ from the domain ofRDF_B. Then:
(a) $\sigma_{\mathcal{C} \supset \mathrm{RDF}_A}(\mu(E, \mathrm{dc})) = \mu(\sigma_{\mathcal{C}}(E), \mathrm{dc})$. (E.g., all regions contained in `location` = montreal.) A similar identity holds for selection on RID attributes.
(b) $\sigma_{(\mathrm{RDF}_A \,\&\, \mathrm{RDF}_B) \supset (\mathcal{C} \& \mathcal{C}')}(\mu(E, \mathrm{dc})) = \mu(\pi_{A,B}(\sigma_{\mathcal{C} \& \mathcal{C}'}(E)), \mathrm{dc})$. (E.g., all regions containing `location` = montreal & `productClass` = meat.) A similar identity holds for selection on RID attributes.
$\pi_{\mathrm{RDF}_{A_i}, \ldots, \mathrm{RDF}_{A_j}}(\mu(E, \mathrm{dc})) = \mu(\pi_{A_i, \ldots, A_j}(E), \mathrm{dc})$.
$\mu(E_1 \cup E_2, \mathrm{dc}) = \mu(E_1, \mathrm{dc}) \cup \mu(E_2, \mathrm{dc})$, where E_1, E_2 are any union-compatible data sets.
$\mu(E_1 \times E_2, \mathrm{dc}) = \mu(E_1, \mathrm{dc}) \times \mu(E_2, \mathrm{dc})$.

Table 4: A Sample of Inter-World Identities

operators interact. The following theorem answers this question. Identity (1) in Table 4 says the populate operator respects the structure of the I- and E-worlds in that populating the union of two dimensions coincides with taking the union of the two populations. A similar remark holds for product. Similarly, as shown in identities (2) and (3), boolean combinations of selection conditions in the intensional world are faithfully mapped by α to corresponding set-theoretic operations in the extensional world.

Theorem 3 Let $\mathcal{D}_1, \mathcal{D}_2$ be any intensional dimensions and E be a data set. Then identities (1)-(3) in Table 4 hold. ■

On the one hand, the kinds of identities above attest to the naturality of our definitions. On the other, they are useful for query optimization via query rewriting. Indeed, for some special cases, we can even go beyond the above identities. For instance, consider the data cube operator. Recall that in our framework, the cube operator merely splits up a data set into regions corresponding to all possible group-bys. For cube, we can consider that the attributes mentioned in the constraints (RDFs), as also those used to construct RIDs, are a subset of the attributes of the given relation. To emphasize this, let us use RID_A (resp., RDF_A) to denote the RID (resp., RDF) attribute associated with attribute A of the relation. The following theorem deals with the data cube operator. Similar, but somewhat different, identities hold for the frequent sets operator, but are omitted for lack of space.

Theorem 4 Let E, E_1, E_2 be data sets. Then for data cube computation, identities (4)-(7) in Table 4 hold. ■

6 Implementation and Optimization Issues

In previous sections, we showed that the framework proposed here can effectively support multi-step mining/analysis activities, that could not be done before within one clean algebraic setting. The question we address in this section is how efficiently we can implement the proposed framework. Some evidence for the possibility for efficient implementation was provided in earlier sections through identities. For space limitations, our discussion here must remain at a somewhat high level. As we argue below, the main reasons behind efficient implementability are: (1) certain key operations admit efficient incremental processing; and (2) many other key operations can benefit from spatial indexing and processing, a strength of the database community.

1. $\alpha((\mathcal{D}_1 \cup \mathcal{D}_2), E) = \mathcal{E}_1 \cup \mathcal{E}_2$.
2. $\alpha(\sigma_{\mathrm{Cond}}(\mathcal{D}_1), E) = \{t \in \mathcal{E}_1 \mid desc(t[\vec{\mathrm{RID}}])$ satisfies Cond $\}$.
3. $\alpha(\pi_{\mathrm{RDF}_i, \ldots, \mathrm{RDF}_j, P_k, \ldots, P_\ell}(\mathcal{D}_1), E) = \pi_{\mathrm{RID}_i, \ldots, \mathrm{RID}_j, \vec{A}_1}(\mathcal{E}_1)$.
4. $\alpha((\mathcal{D}_1 - \mathcal{D}_2), E) = \mathcal{E}_1 - \mathcal{E}_2$.
5. $\alpha(\Upsilon(\mathcal{D}_1), E) = \{t \mid t \in \mathcal{E}_1, desc(t[\vec{\mathrm{RID}}])$ is consistent $\}$.

Table 5: Efficient Computation of the Refresh Operation

1. $\lambda(\sigma_{\mathrm{Cond}}(\mathcal{E}_1)) = \{r \in \mathcal{D}_1 \mid \exists t : t \in \sigma_{\mathrm{Cond}}(\mathcal{E}_1), desc(t[\vec{\mathrm{RID}}]) = r[\vec{\mathrm{RD}}\mathrm{F}], t[\vec{\mathrm{RID}}] = r[\vec{\mathrm{RID}}]\}$.
2. $\lambda(\pi_{\vec{X}}(\mathcal{E}_1)) = \pi_{\mathrm{RID}_i, \ldots, \mathrm{RID}_j, \mathrm{RDF}_i, \ldots, \mathrm{RDF}_j}(\mathcal{D}_1)$, where $\vec{X} \cap \mathbf{A}_1 = \{\mathrm{RID}_i, \ldots, \mathrm{RID}_j\}$.
3. $\lambda(\mathcal{E}_1 \cup \mathcal{E}_2) = \mathcal{D}_1 \cup \mathcal{D}_2$.
4. $\lambda(\mathcal{E}_1 - \mathcal{E}_2)) = \mathcal{D}_1 - \mathcal{D}_2$.
5. $\lambda(\mathcal{E}_1 \times \mathcal{E}_2) = \mathcal{D}_1 \times \mathcal{D}_2$.

Table 6: Efficient Computation of the Lookup Operation

6.1 Incremental Computation

Two main operations linking different worlds are refresh and lookup, since they help maintain extensional (resp., intensional) dimensions in sync with manipulations being done on the intensional (resp., extensional) dimensions.

Theorem 5 Let $\mathcal{D}_1, \mathcal{D}_2$ be intensional dimensions, E be any data set, and $\mathcal{E}_1, \mathcal{E}_2$ be the corresponding extensional dimensions based on E, i.e., $\mathcal{E}_i = \alpha(\mathcal{D}_i, E)$. Suppose $\vec{A}_1$ is the set of non-RID attributes of $\mathcal{E}_1$. Then the identities listed in Table 5 hold. ■

The above theorem shows that the refresh macro, as defined in Definition 17, admits efficient incremental processing. Given that the extensional dimension $\mathcal{E}_i$ has been created based on the intensional counterpart $\mathcal{D}_i$, refreshing $\mathcal{E}_i$ w.r.t. an additional dimension algebra operation (and hence a sequence of operations) on $\mathcal{D}_i$, can be done *solely by examining the content of the existing* $\mathcal{E}_i$.

The following result shows that lookup – another key bridging operation – admits incremental processing as well.

Theorem 6 Let $\mathcal{E}_1, \mathcal{E}_2$ be two extensional dimensions and $\mathcal{D}_1, \mathcal{D}_2$ be their intensional counterparts, i.e., $\mathcal{D}_i = \lambda(\mathcal{E}_i)$. Suppose also that the schema of $\mathcal{D}_i$ is $\mathbf{A}_i$. Then the identities listed in Table 6 hold. ■

6.2 Application of Spatial Techniques

So far, we have considered the optimization of the bridging operations: refresh (which is essentially an incremental version of populate) and lookup. In the previous sections, we established similar identities for other operators including (some special cases of) the mine operator. Next, we turn our attention to efficient implementation of the dimension algebra.

The main problems involving constraints which directly impact the efficiency of dimension algebra at a logical level, are testing constraint implication (and equivalence), consistency checking, and constraint simplification. It is important to note that, as shown in Table 1, numerous existing data mining tasks produce isothetic regions, for which all three problems can be solved efficiently. Specifically, in this case, the constraints are of the form $A_i \theta c$, θ being $\leq$ or $\geq$, and the problems can be solved in linear time.

At a physical level, known spatial indexing techniques can be effectively leveraged to help quickly locate points in isothetic regions. Standard multi-dimensional indexing techniques, such as R-trees and its variants [12, 6], are directly applicable to the checking of such spatial predicates as overlap, containment, etc. Note that in traditional spatial processing, indexing structures like R-trees have proved useful for polygons that are not even isothetic (e.g., convex), with the addition of a refinement phase. This suggests that such index structures should prove equally effective in the implementation of key dimension algebra operators, whether the regions are isothetic or not.

Finally, the RID attributes can often be encoded so that the checking hierarchical predicates can be performed efficiently. For example, when the hierarchy in question is a tree or can be factored into a product of tree hierarchies, all hierarchy predicates can be checked in linear time, using the ideas developed in [17]. Examples include categorical hierarchies, e.g., `location`, `product`, as well as decision trees. A hierarchy defined by a tuple of hierarchical attributes, each of which has a tree hierarchy, is a natural example of a hierarchy that can be factored into trees. For range hierarchies (over total orders or partial orders), merely encoding the endpoints of the ranges gives an efficient encoding, from which parents, children, ancestors, etc. can be easily enumerated. For instance, the RID [2, 10] has [1,10] and [2,11] as parents, and [2,9] and [3,10] as children. Finally, for lattices corresponding to powersets (e.g., frequent sets), a number of techniques exist. For instance, we could use a bit vector (made suitably compact to save space) for efficient checking of hierarchy predicates.

7 Related Work

Earlier, we have mentioned numerous data mining studies and how they can be classified into two generations. Below we discuss a few studies that are also very related to the subject matter of this paper – namely, the development of a model and algebra for data mining. In [16], Mannila and Imielinski discussed their vision of manipulating association rules algebraically. The 3W model developed here is a concrete proposal, and is more general, as it models not only association rules, but also many well-known data mining and analysis operations, such as decision trees, data spheres, etc. In [23], Meo et al. proposed adding an association rule operator to SQL. Again, our proposed 3W model is far more general and fundamental.

The dimension algebra developed here is related to constraint query languages in general [19, 7] and geometric query languages in particular [26, 13]. Paredaens et al. [26], and Gyssens et al. [13] consider constraint languages that are in the FO[R] class, i.e., first-order logic augmented with polynomial inequalities over reals, and relation variables with fixed arities. This class of constraints are shown to have nice properties, such as decidability for equivalence checking. Our dimension algebra uses constraints restricted to LINEAR[R], and therefore enjoys at least the properties of FO[R].

The notion of hierarchical domains we use in this paper was first proposed by Jagadish et al. [17]. However, they confine attention to categorical tree hierarchies in data warehouses. By contrast, we use hierarchies (via the notion of dimensions) as a central unifying concept for disparate mining tasks and for warehousing, and permit general lattice-based hiearrchies. Finally, the notion of constraint domains was not considered by them, while this notion plays a pivotal role in the I-world. Besides, we propose an algebra for manipulating dimensions (among other things).

8 Summary and Future Work

We have presented the 3W Model and an algebraic framework for unified data mining and analysis. It allows the input of one operation to be the output of another. We have shown via examples and numerous operator identities that the proposed framework is natural and is expressive enough to support multi-step mining processes within one clean algebraic setting, to our knowledge, *for the first time in the literature*. Furthermore, we have also demonstrated and argued that the proposed framework can be efficiently implemented.

This being the first paper on unified mining and analysis within a formal framework, it opens up several important questions for future research. Mining-query optimization needs to be thoroughly investigated – both at the logical level of query rewriting

using identities and at the physical level of utilizing spatial database techniques. It is very important to develop a prototype system of the proposed framework for extensive empirical evaluation, which may lead to many interesting algorithmic and optimization problems. A third issue is tight integration of data warehousing and mining, for which we have taken a first step by using dimensions as a unifying concept for warehouses and the result of mining tasks. But much more work remains to be done. Our ongoing work addresses these questions.

References

[1] C. Aggarwal and P. Yu. Online Generation of Association Rules. In *Proc. 1998 ICDE*, pp 402–411.

[2] R. Agrawal, J. Gehrke, D. Gunopolos and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proc. 1998 SIGMOD*, pp. 94–105.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 SIGMOD*, pp 207–216.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 VLDB*, pp 487–499.

[5] M. Ankerst, M. Breunig, H.P. Kriegel and J. Sander. Optics: Ordering Points to Identify the Clustering Structure. In *Proc. 1999 SIGMOD*, pp. 49–60.

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. newblock The R*-Tree: an Efficient and Robust Access Method for Points and Rectangles. In *Proc. 1990 SIGMOD*, pp. 322–331.

[7] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational Expressive Power of Constraint Query Languages. Journal of the ACM, 45:1, 1998, pp. 1–34.

[8] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 SIGMOD*, pp 265–276.

[9] S. Chaudhuri. Data mining and database systems: Where is the intersection? *Bulletin of the Technical Committee on Data Engineering*, 21:4–8, March 1998.

[10] M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential Pattern Mining with Regular Expression Constraints, In *Proc. 1999 VLDB*, pp 223–234.

[11] J. Gray et al. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Proc. 12th ICDE*, 1996, pp. 152–159.

[12] R. Guttmann. A Dynamic Index Structure for Spatial Searching. In *Proc. 1984 SIGMOD*, pp. 47–57.

[13] M. Gyssens, J. Van den Bussche, and D. Van Gucht. Complete Geometric Query Languages. *J. of Comput. & Syst. Sciences* 58:3(483-511) 1999.

[14] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 VLDB*, pp 420–431.

[15] C. Hidber. Online Association Rule Mining. In *Proc. 1999 SIGMOD*, pp 145–156.

[16] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of ACM*, 39:58–64, 1996.

[17] H. Jagadish, L. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses? In *Proc. 1999 VLDB*, pp. 530–541.

[18] T. Johnson, I. Kwok, and R. Ng. Fast Computation of 2-Dimensional Depth Contours. In *Proc. 1998 KDD*, pp. 224–228.

[19] P. Kannellakis, G. Kuper, and P. Revesz. Constraint Query Languages. Journal of Computer and System Sciences, 51:1, 1995, pp. 26–52.

[20] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 1994 CIKM*, pp 401–408.

[21] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. 1999 SIGMOD*, pp. 157–168.

[22] H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1, 1997, pp. 259-289.

[23] R. Meo, G. Pasila, and S. Ceri. A New SQL-like Operator for Mining Association Rules. In *Proc. 1996 VLDB*, pp. 122–133.

[24] R. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. 1994 VLDB*, pp. 144-155.

[25] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 SIGMOD*, pp. 13–24.

[26] J. Paradaens, J. Van dn Bussche, and D. Van Gucht. Towards a Theory of Spatial Database Queries. In *Proc. 1994 PODS*, pp. 279–288.

[27] J. Quinlan. Induction of Decision Trees. Machine Learning, 1, 1986, pp. 81–106.

[28] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proc. 1998 SIGMOD*, pp 343–354.

[29] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 VLDB*, pp 594–605.

[30] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, and S. Nestorov. Query flocks: A generalization of association-rule mining. In *Proc. 1998 SIGMOD*, pp 1–12.

[31] T. Zhang, R. Ramakrishnan and M. Livny. BIRCH: an Efficient Data Clustering Method for Very Large Databases. In *Proc. 1996 SIGMOD*, pp. 103–114.