DADA: A Data Cube for Dominant Relationship Analysis

Cuiping Li[1]* Beng Chin Ooi[2] Anthony K.H. Tung[2] Shan Wang[1]

[1]Dept. of Computer Science,Renmin University of China, Beijing 100872, China.
cuiping_li@263.net,swang@mail.ruc.edu.cn

[2] Dept. of Computer Science, Natl University of Singapore, S'pore 117543, Singapore.{ooibc,atung}@comp.nus.edu.sg

ABSTRACT

The concept of dominance has recently attracted much interest in the context of skyline computation. Given an N-dimensional data set S, a point p is said to dominate q if p is better than q in at least one dimension and equal to or better than it in the remaining dimensions. In this paper, we propose to extend the concept of dominance for business analysis from a microeconomic perspective. More specifically, we propose a new form of analysis, called **Dominant Relationship Analysis (DRA)**, which aims to provide insight into the dominant relationships between products and potential buyers. By analyzing such relationships, companies can position their products more effectively while remaining profitable.

To support DRA, we propose a novel data cube called DADA (Data Cube for Dominant Relationship Analysis), which captures the dominant relationships between products and customers. Three types of queries called **Dominant Relationship Queries (DRQs)** are consequently proposed for analysis purposes: 1)Linear Optimization Queries (LOQ), 2)Subspace Analysis Queries (SAQ), and 3)Comparative Dominant Queries (CDQ). Algorithms are designed for efficient computation of DADA and answering the DRQs using DADA. Results of our comprehensive experiments show the effectiveness and efficiency of DADA and its associated query processing strategies.

1. INTRODUCTION

The concept of dominance has recently attracted much interest in the skyline context in relation to answering preference queries. In this paper, we propose extending the concept for business analysis on a data cube.

Given an N-dimensional dataset S, let $D = \{D_1, ..., D_N\}$ be the set of dimensions. Let p and q be two data points in S. We then denote the values of p and q on dimension D_i as p_i and q_i.

*Part of work done while author visited National University of Singapore and partly supported by NSFC(60473069, 60496325, 60273017)

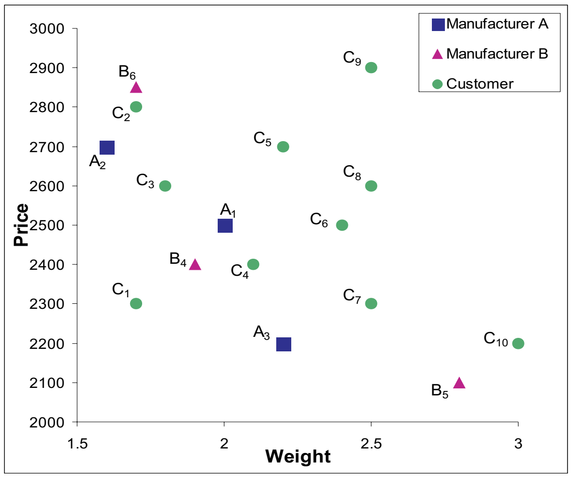


Figure 1: Notebooks and Customer Preferences

DEFINITION 1.1 (DOMINATE, $p < q$). *For each of the dimension D_i, we define an order $\prec_{D_i}$. We say that p is better than q in dimension D_i (denoted as $p_i < q_i$) if p_i comes before q_i based on $\prec_{D_i}$ or conversely, q_i is worse than p_i (also denoted as $q_i < p_i$). If p_i and q_i are equals, we denote them as $p_i = q_i$.*

Under this setting, a point p is said to dominate q if p is better or equal to q in all dimensions, and is better than q in at least one of the dimensions. □

Model	CPU	Memory	Harddisk	Weight	Price
A_1	2.0Mhz	1024Mb	40Gb	2kg	$2500
A_2	1.9Mhz	256Mb	60Gb	1.6kg	$2700
A_3	1.9Mhz	512Mb	60Gb	2.2kg	$2200
B_4	1.8Mhz	512Mb	40Gb	1.9kg	$2400
B_5	1.9Mhz	1024Mb	40Gb	2.8kg	$2100
B_6	1.8Mhz	768Mb	50Gb	1.7kg	$2850

Table 1: Notebook Configuration

Given the concept of dominance, the skyline points in the dataset S are defined as those points which are not dominated by any other point in S. Skyline points are useful in answering preference queries [5] since the best answer given any weight assignment for a monotonic preference function is always guaranteed to come from skyline points. As an example, we consider a set of six notebook models as shown in Table 1, where the first three are produced by manufacturer A and the next three by manufacturer B. If we consider only their weight and price attributes, which are better if minimized (we call these min attributes for ease of reference), then the skyline as shown in Figure 1 will be A_2, A_3, B_4 and B_5 with the two notebooks A_1 and B_6 being dominated by the competitor's B_4 and A_2 respectively. In this case, regardless of the weight being assigned by the customers, the highest scoring notebook will only come from A_2, A_3, B_4 and B_5. The same concept can easily be extended to more attributes such as CPU speed, memory size, etc., where a higher value is better (i.e., what we call maximum attributes).

While the concept of dominance is very useful from the perspective of customers selecting the products they like, what is interesting to manufacturers is whether their products are popular with customers compared to their competitors' products. Referring again to Figure 1, let C_1,...,C_{10} indicate the preference of 10 customers in a survey in which they are asked the weight of the notebook they are comfortable with, and the price they expect to pay for it. Relative to each notebook, there are three types of customers:

- **Dominated Customers:** As the name implies, these are customers who are dominated by the notebook, i.e., the notebook definitely satisfies their requirements. For example, the dominated customers of notebook A_1 are C_5, C_6, C_8 and C_9.
- **Dominating Customers:** These are customers who dominate the notebook, i.e., the notebook definitely does not satisfy their requirements. For example, the dominating customer of notebook A_1 is C_1.
- **Incomparable Customers:** These are customers who neither dominate nor are dominated by the notebook. For example, C_3 and C_4 are incomparable customers of notebook A_1.

Given any notebook, the numbers of dominated and dominating customers can be used as measurements to gauge how good the positioning of the product is in the market. Obviously, it is best to dominate as many customers as possible while keeping the number of dominating customers minimal (or equivalently, maximizing the number of incomparable customers among those who are not dominated). The tradeoff between these two measures is not always straightforward. For example, if the price of notebook A_2 is decreased to \$2400 with the use of cheaper components that increase the weight to 1.75kg, its dominated customers will increase from three to six, but the number of customers dominating it will increase from zero to one.

One obvious way to avoid all these concerns is to position the notebook in a market which is not dominated by any customers or any other notebooks. However, this is not always the best alternative because of the following two reasons:
1. The notebook might become non-profitable, needing a large amount of resources to prevent it from being dominated.
2. Even if a notebook is dominated by another notebook, hidden factors such as brand loyalty and marketing strategies could still mean that customers would buy the notebook. Furthermore, the manufacturing capacity for the dominating notebook might not be enough to cater to all the customers it dominates. As such, it might make sense to allow a notebook to be dominated in exchange for higher profit.

From the above discussion, the usefulness of analyzing the dominant relationships between products and customers is clear. From here on, we will refer to such form of analysis as **Dominant Relationship Analysis (DRA)**. In this paper, we focus on DRA, and contribute to its advancements with the following:

- We present three types of queries as representative queries for DRA: i) Linear Optimization Queries (LOQs), ii) Subspace Analysis Queries (SAQs), and iii) Comparative Dominant Queries (CDQs). Each type of queries introduces a different aspect of DRA that we hope to illustrate. Collectively, we call these queries **Dominant Relationship Queries (DRQs)**.
- We present DADA [1], a data cube organization that is designed for DRA. We construct DADA by converting the dominant relationship between spatial objects into a lattice, and then making use of the convexity of our measure for effective compression. We present efficient query processing strategies on top of DADA for the three DRQs.
- We present comprehensive experiments to demonstrate the efficiency of our algorithms for constructing DADA and answering DRQs.

The paper is organized as follows: Section 2 defines the three types of DRQs. Section 3 discusses related work. Section 4 presents the computation of DADA, and Section 5 presents query proecssing strategies for the three DRQs using DADA. We present the experimental evaluation in Section 6, and conclude in Section 7.

2. PRELIMINARIES

In this section, we first set the context and state the assumptions that are adopted in this paper. We then introduce the three representative DRQs that we will consider in this paper.

2.1 Context and Assumptions

We assume we have two manufacturers A and B, each producing a set of products $P_A = \{A_1, ..., A_s\}$ and $P_B = \{B_1, ..., B_t\}$, respectively. We also have the preference of a set of customers $C = \{C_1, ..., C_n\}$.

Each of the products or customer preferences can be represented as a point in an N-dimensional space, D, with dimensions D_1,...,D_N being the attributes of the products and customer preferences. For simplicity, we use the general term "object" to refer to a product or a customer's preference if the need to distinguish them is not needed.

We assume that the domain values of each dimension D_i, $DV(D_i)$, are **discretized, fully ordered, numerical** values which can be mapped into positive integers $\{1,...,|DV(D_i)|\}$. Thus, the whole of the N-dimensional space can be divided

[1]DADA stands for <u>D</u>ata Cube for <u>D</u>omin<u>a</u>nt Relationship Analysis

conceptually into $|DV(D_1)| \times |DV(D_2)| \times ...|DV(D_N)|$ cells, and each of the objects lies in one of the cells. Given any object p, we use the notation $p[D_i]$ to refer to the value of p in dimension D_i. We highlight the following:

1. Our assumption is similar to most if not all data cube techniques which discretize numeric dimensions to an acceptable resolution level.

2. Mapping domain values into positive integers does not affect the dominant relationship between the objects of analysis. This is only done to ensure easier discussion later on in the paper.

We also assume, without loss of generality, that the attributes of a product or customer preference are minimum attributes [5], i.e., smaller values are preferred [2]. With this final assumption, we can now adopt the definition of **dominate** that we have provided in Section 1 for the following definitions:

Definition 2.1. *dominating*(p, C, D')
Given an object p, a set of objects C and a set of dimensions $D' \subseteq D$, we define dominating(p, C, D') as the set of objects from C which are dominated by p in the subspace D' of D. □

Definition 2.2. *dominated*(C, p, D')
Given an object p, a set of objects C and a set of dimensions $D' \subseteq D$, we define dominated(C, p, D') as the set of objects from C which dominate p in the subspace D' of D. □

2.2 Three Representative DRQs

In this section, we look at three types of DRQs that are representatives of the DRA we seek to examine.

2.2.1 Linear Optimization Query

We first look at **LOQs**. These queries are motivated by the observation that manufacturers do not have infinite resources, and must consider various trade-offs and constraints when they position their products. For example, making the notebook lighter requires better components, which in turn pushes up the notebook price. Here, we model such constraints as a linear plane L that is anti-correlated with regard to all the dimensions. Our assumption here is that all attributes are minimum attributes, and making a product better in one attribute requires sacrificing other aspects in order to stay profitable.

Definition 2.3. *Linear Optimization Query (LOQ(L, C, D))*
Given a plane, L, and a set of objects, C, in an N-dimensional space of D, we define LOQ(L, C, D) as the aggregate max$(|dominating(p, C, D)|)$, where p is any point in the plane L. □

Note that for brevity, we take $|dominating(p, C, D)|$ as the measure for optimization. In fact, $|dominated(C, p, D)|$ can be used as the measure for optimization as well (to minimize in this case).

Obviously, the actual location of the point p in the definition of LOQ is as important as $LOQ(L, C, D)$ itself. However, since p might not be unique, finding $LOQ(L, C, D)$ can be more efficient and would make our definition tidier. Later in the paper, our algorithm for handling LOQ will identify both $LOQ(L, C, D)$ and p. We now illustrate the concept of LOQ using the following example:

[2]Maximum attributes in which larger values are preferred can be converted into minimum ones by taking negations.

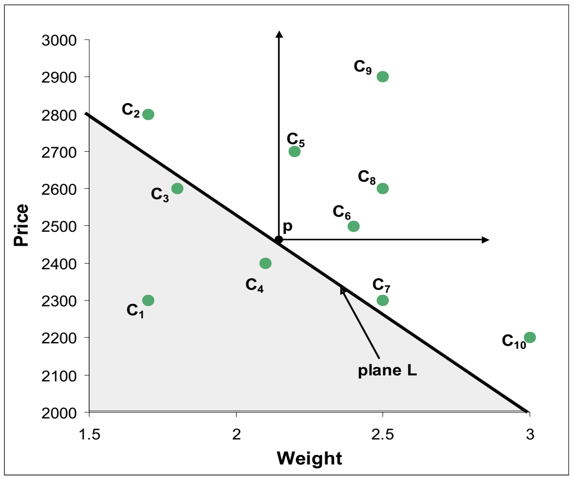


Figure 2: Linear Optimization Query: Which portion of plane L dominates the most number of points?

Example 1. *Consider Figure 2, where customer preferences for the price and weight of the notebook are depicted together with a plane L. The shaded region at the bottom-left of the plane represents configurations which are not profitable for the manufacturer, and the region at the top-right of the plane are configurations which are profitable.*

In this example, LOQ(L,C,D)=4 at location p indicated in the diagram. □

2.2.2 Subspace Analysis Queries

Next, we consider **SAQs**. These queries are motivated by our observation that manufacturers could be interested to analyze the dominant relationship in the subspace of D. For example, they might find that a certain combination of attribute settings in a product are important to many customers. This could help identify a niche market for the manufacturer, who could then design a product targeting such a customer segment. More formally:

Definition 2.4. *SAQ(p,C,D')*
Given a set of points C and a point p in the N-dimensional space of D, find:
1. $|dominating(p, C, D')|$ *and*
2. $|dominated(C, p, D')|$

where $D' \subseteq D$. □

Definition 2.4 is the most basic SAQ on which more complex SAQs can be built. For example, we can choose to compute SAQ(p,C,D')s for all subsets D' of D that satisfy a certain interest measure threshold such as:

$$|dominating(p, C, D')| - |dominated(C, p, D')|$$

Our purpose here is to illustrate the usefulness of SAQs with a simple example rather than exhaustively define all possible types of SAQs.

Example 2. *Returning to Figure 2 where p is dominating four points and dominated by two points, we compute SAQ(p,C,D'), where C represents all customer preferences and $D' = \{Weight\}$, and we find p dominating six points and dominated by four points in the subspace. If we use the measure that we gave earlier, we can conclude that the dominating power of p is not much stronger if we only consider the attribute "Weight" instead of "Weight" and "Price".* □

Note that in general, both $|dominating(p, C, D')|$ and $|dominated(C, p, D')|$ increase when we move to a subset of D. However, the difference between them does not follow such a property.

2.2.3 Comparative Dominant Query

As the name implies, **CDQs** are queries that aim to compare the set of dominated objects between competitive products. We first introduce the concept of **group dominant**:

Definition 2.5. *Group Dominant, $gdominating(A, C, D)$ Given two sets of objects A and C in an N-dimensional space of D, we define $gdominating(A, C, D)$ as the set of objects in C which are dominated by some object from A.* □

We can now define two sub-classes of CDQs using the concept of group dominant.

Definition 2.6. $CDQ^-(A, B, C, D)$
Given three sets of objects in the N-dimensional space of D, we define $CDQ^-(A, B, C, D)$ as:

$$|gdominating(A, C, D) - gdominating(B, C, D)| \qquad \square$$

Definition 2.7. $CDQ^\cap(A, B, C, D)$
Given three sets of objects in the N-dimensional space of D, we define $CDQ^\cap(A, B, C, D)$ as:

$$|gdominating(A, C, D) \cap gdominating(B, C, D)| \qquad \square$$

Intuitively, $CDQ^-(A, B, C, D)$ computes the number of objects in C that are dominated by some objects in A and not by any object in B. This is useful for a manufacturer who wants to identify the number of customers who are solely dominated by his/her products. Likewise, the query $CDQ^\cap(A, B, C, D)$ is useful for manufacturers who want to know the number of customers who are dominated by both their products and those of their competitors. Note that while our definition of CDQ is general enough for finding, say, the total number of customers dominated by a set of products (set B to $\emptyset$ with CDQ^-), or comparing a single product against a set of them (set A to a single item), there could be other interesting CDQs. For example, we have not tried to account for customers who dominate some set of products. Again, we emphasize that we want to illustrate the spirit of CDQ rather than enumerate CDQs exhaustively.

3. RELATED WORK

3.1 Microeconomic View of Data Mining

Our work is mainly inspired by the work in [19] which proposes to view data mining from a microeconomic perspective i.e., the authors argue that the interestingness of knowledge being discovered should be measured by their utility to the organization. Various examples are given in [19] to illustrate utility oriented mining, including profit oriented association discovery, market segmentation, data mining as sensitivity analysis, and segmentation in a model of competition. Individual efforts towards this direction include [27, 28, 6] for profit oriented association rules discovery, [18, 11] for customer oriented catalog segmentation, and [29] for data mining as sensitivity analysis. Our work approaches this direction from a new perspective by providing a platform on which microeconomic based data mining can be performed. Indeed, construction of a data cube has been noted as a good means to facilitate more advanced data mining [15, 9], and several major database products have such a feature.

By computing DADA, we are using the relationships of dominated/dominating customers and products as a basis for decision making. For examples, LOQs allow us to find an interesting market position in the product attribute space which can dominate more customers while remaining profitable. SAQs illustrate a different aspect in that they seek to find attribute combinations that ensure more customers would be dominated. This is similar in spirit to profit oriented association rule mining. Finally, CDQs allow organizations to compare targeted customers both within their products and against their competitors' products. In the first case, the comparison would provide organizations with additional information for segmenting their own products. In the second case, the additional information could be useful for segmentation in a model of competition – an area which has so far been left largely untouched by the data mining community. We envision that dominant relationship analysis is set to become an important tool in data mining just like OLAP, association rule mining, classification/regression modelling and cluster analysis [2].

3.2 Data Cube

The data cube operator was proposed in [13, 14]. Much research has gone into efficiently computing data cubes [1, 16, 23, 3] and organizing them for query answering [16, 24, 17, 25]. The relationship between cells in a data cube is often seen as a lattice structure [4], where a parent/child nodes pair represents the subset/superset relationship of the dimensions being summarized. Interestingly, as we will show later, the dominant relationship between cells in our attribute space can be organized as a lattice structure as well. This means that the two concepts can be elegantly represented in a composite lattice structure just as hierarchies are introduced into data cubes [16]. Furthermore, we are able to reuse some of the concepts in cube computation [3] and compression using concepts from Galois lattice [10, 12, 21] in order to compute and organize DADA for query answering efficiently. To our knowledge, DRQs are different from conventional data cube queries and designed for different purposes.

3.3 Skyline Queries

Our work can be viewed as a generalization of skyline queries [5, 26, 20, 22, 30] in the sense that skyline cells are a subset of the cells that we are interested in. Having computed DADA, it is easy to identify cells that contain skyline points since these are the cells where $dominated(C, p, D)$ equals 0. On the other hand, we have developed DADA bearing in mind that not every product can afford to compete with other products in the skyline. Interestingly, we observe that a product that is in the skyline does not necessarily dominate more customers than products that are not in the skyline. DRQs are thus more important than skyline queries for positioning products in the market.

The are also emerging work on finding interesting skyline points in high dimensional space [8, 7, 31]. These work proposed new notion of dominance in high dimensional space to overcome the problem of the current definition which can result in too many skyline points for high dimensional data. Adopting these new notion of dominance in DADA will be an interesting subject for future studies.

4. DEFINING AND COMPUTING DADA

In this section, we first define DADA, and then present efficient algorithms for computing and compressing DADA.

4.1 Defining DADA

A lattice as defined in [4] refers to a partially ordered set $(\mathcal{L}, \preceq)$ such that every pair p,q in $\mathcal{L}$ has a least upper bound, $lup(p,q)$ and a greatest lower bound $glb(p,q)$. If $\mathcal{L}$ is finite, then we refer to the lattice as a finite lattice. A finite lattice can be represented as a directed graph in which the lattice elements in $\mathcal{L}$ are the nodes, and there exists a directed edge from a node e to e' if and only if: 1) $e \preceq e'$, and 2) $\nexists e'', e \preceq e'' \preceq e'$. In this case, we say that e is the parent of e' (correspondingly, e' is the child of e). If there exists a path from a node e to e', then e is called the ancestor of e (correspondingly, e' is the descendant of e).

THEOREM 4.1. *Let $\mathcal{L}$ be the set of cells in the N-dimensional space formed by D. Let $\preceq$ be either the dominating or dominated relation between the cells. Then $(\mathcal{L}, \preceq)$ is a finite lattice.*

PROOF. Let $p = \langle p_1, ..., p_N\rangle$ and $q = \langle q_1, ..., q_N\rangle$ be any two cells where $\preceq$ is the dominating relation. Then $lup(p,q) = \langle min(p_1,q_1), ..., min(p_N,q_N)\rangle$ and $glb(p,q)$=$\langle max(p_1,q_1)$, ..., $max(p_N,q_N)$ $\rangle$. That is, if a cell is dominated by $lup(p,q)$, then it must be larger than $lup(p,q)$ in all dimensions and thus could not dominate either p or q. The same reasoning applies for glb and for the case where $\preceq$ is the dominated relation. □

Given the above theorem, we can now define the dominating and dominated lattices.

DEFINITION 4.1. *Dominating/Dominated Lattice*
Let $\mathcal{L}$ be the set of cells in the N-dimensional space formed by D. If $\preceq$ is the dominating relationship, $(\mathcal{L}, \preceq)$ is called the **dominating lattice**. *If $\preceq$ is the dominated relationship, then $(\mathcal{L}, \preceq)$ is called the* **dominated lattice**. □

DEFINITION 4.2. *DADA*
*Given the set of cells in the N-dimensional space formed by D and a set of points C that are located in the same space, a Data Cube for Dominant Relationship Analysis (***DADA***) refers to a data cube formed from* EITHER *of the following:*

1) The dominating lattice of the cells with the aggregate at each cell/node p being dominating(p, C, D).

2) The dominated lattice of the cells with the aggregate at each cell/node p being dominated(C, p, D). □

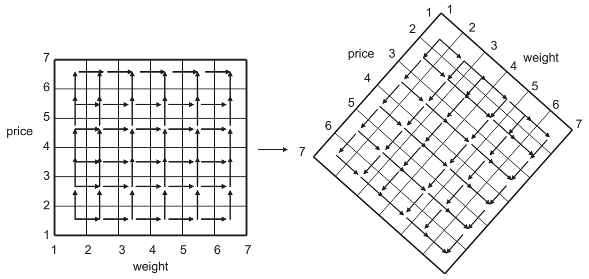
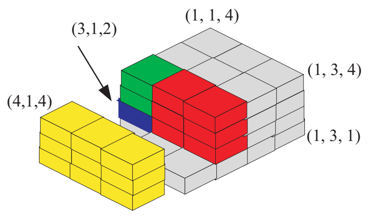


Figure 3: Domination Relationship to Lattice Structure

Points	X	Y	Z
P_1	2	1	2
P_2	3	2	1
P_3	3	2	2

(a)



(b)

Figure 4: (a) Set of Points; (b) Multiarray

Figure 3 illustrates how the dominant relationship of our notebook example is converted into a lattice structure on the right hand side of the figure. As can be seen, ignoring the boundary condition, each node $p = \langle p_1, ..., p_N\rangle$ in the lattice has N children nodes which are of the form $p' = \langle p_1, ...p_i + 1, .., p_N\rangle$ for some i , $1 \leq i \leq N$.

4.2 Computing DADA

Next, we describe our techniques for computing DADA. We mainly illustrate how to compute the cube for a dominating lattice since computation of a dominated lattice can be done in a similar fashion.

4.2.1 Basic Algorithm

To illustrate our algorithm, we will use the following running example in this section.

EXAMPLE 3. *Assume that we have a three-dimensional space D, and the cardinality of each dimension is $4, 3$ and 4 respectively. Figure 4(a) shows a set C which includes three points.* □

DEFINITION 4.3. **Lexicographical Order**
We impose an arbitrary order for the dimensions as D_1,...D_N. We also impose a lexicographical order on the cells such that p is ordered before q if and only if there exists an i such that $p_i < q_i$ and for all $j < i$, $p_j = q_j$. □

DEFINITION 4.4. **Cell Enumeration Tree**
Given a dominating lattice, we derive a cell enumeration tree by removing all edges from a parent to a child if the parent is not immediately before the child in the lexicographical order. □

Figure 5 shows the cell enumerating tree of our running example. Assuming the dimension order is $D_1 \prec D_2 \prec D_3$, the order of node in Figure 5 will be ⟨1,1,1⟩, ⟨2,1,1⟩, ⟨3,1,1⟩,... ⟨4,3,4⟩. Hereafter, unless otherwise mentioned, we will use the same assumption for dimension order.

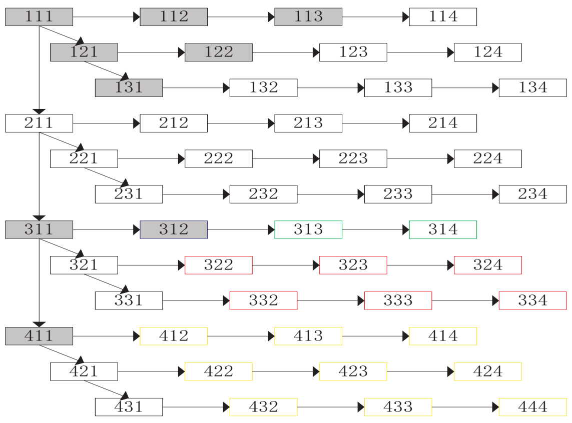


Figure 5: Cell Enumeration Tree

We next look at an example to show the relationship between the number of points dominated at a cell p and the number of points dominated by its children. Consider the cell $\langle 1, 1, 1 \ \rangle$ in Figure 5. It is obvious that the number of points p dominates is the sum of all the points that are in the cells below it in the tree. This corresponds to the total number of points being dominated by its children in the lattice in different subspaces.

$$\begin{aligned}&dominating(\langle 1,1,1\rangle, C, \{X,Y,Z\}) = \\ &numcell\langle 1,1,1\rangle + \\ &dominating(\langle 1,1,2\rangle, C'', \{Z\}) + \\ &dominating(\langle 1,2,1\rangle, C', \{Y,Z\}) + \\ &dominating(\langle 2,1,1\rangle, C, \{X,Y,Z\})\end{aligned}$$

where C'',C' are the points that are in the plane ($X = 1, Y = 1$) and ($X = 1$) respectively while $numcell\langle 1,1,1\rangle$ contain the number of points in cell $\langle 1,1,1\rangle$

Such a property can then be applied to its three children in order to determine how many points they dominate. To understand how this can be interpreted on the cube itself, we refer to cell $\langle 3,\ 1,\ 2\rangle$ in Figure 4(b) (the blue cell) which can be obtained by summing up the value in the yellow, red and green regions of the same figure.

Generalizing this idea, for any cell p in an N-dimensional space D, |dominating(p, C, D)| can be obtained by summing up all its children's aggregation. For each cell, p, we need to compute DN_1, DN_2, ..., DN_N where DN_i represents the number of points p dominates in subspaces $\{D_i, ..., D_N\}$. For example, the cell $\langle 1,3,2\rangle$ must compute the number of points it dominates in subspaces $\{Y, Z\}$ and $\{Z\}$ because it is the second child of cell $\langle 1,2,2\rangle$ and the last child of $\langle 1,3,1\rangle$ in the lattice, and thus requires these different values for different parents.

The pseudo code of the DADA computation algorithm is shown in Algorithm 1, inspired by the BUC algorithm proposed by Bayer and Ramakrishnan [3]. It recursively partitions points in a depth-first manner so that points dominated by the same cell are grouped together when computing the value of the cell. One main difference between our algorithm and the BUC algorithm is that in ours, having partitioned the data on a certain dimension, the partition which is associated with the highest domain value in that dimension will be visited first. This is to ensure that all aggregate for the children of a cell is available when computing the value for the cell.

After initialization, the main algorithm calls *Enumerate()* with the smallest cell $\langle$1,1,...,1$\rangle$. The function *Enumerate()* implements a depth first search and performs recursive computation of the dominating number for each cell on the enumerating tree. Line 9 in Algorithm 1 performs pruning when a partition is empty. This is important for the efficiency of DADA computation and will be explained in Subsection 4.2.4. For each dimension D between dim and N, the input dataset is partitioned on dimension D (Line 6). Line 7 iterates through the partitions for each distinct value in descending order. The partition becomes the input dataset in the next recursive call to *Enumerate()*, which computes the dominating number on the partition for dimensions $D + 1$ to N.

As we have mentioned, the descending order in Line 7 is very important in computing DADA on a dominating lattice. This order guarantees that when a cell is being processed, all the required values for its children are already available. To illustrate based on Figure 5, the first enumeration call will move us from $\langle 1,1,1\rangle$ to $\langle 4,1,1\rangle$, the second call from $\langle 4,1,1\rangle$ to $\langle 4,3,1\rangle$, and the third call from $\langle 4,3,1\rangle$ to $\langle 4,3,4\rangle$. This brings us to the bottom of the lattice where the value is to be computed and propagated upwards.

Now, we look at the procedure *ComputeDN* in the second line of function *Enumerate()*. Given the input cell, *ComputeDN*'s task is to compute the number of points being dominated by the cell in all subspaces. The algorithm proceeds from the last dimension to the first dimension. For value of i from dim to 1, the number of points dominated by the cell in subspace $\{$D$_i$,...,D$_N\}$ is computed by adding the number of points the cell dominates in subspace $\{$D$_{i+1}$,...,D$_N\}$ to the number of points its i^{th} child dominate in $\{$D$_i$,...,D$_N\}$. This value is stored in $DnCount[i]$ for the cell. At the end of the loop, the number of points dominated by the cell is available in the variable $tempDominatingNum$.

To see this more clearly, consider how we can compute $dominating(\langle 1,1,1\rangle, C, \{Y,Z\})$ by adding the following two values:

1. $dominating(\langle 1,1,1\rangle, C, \{Z\})$
2. $dominating(\langle 1,2,1\rangle, C, \{Y,Z\})$

Note that we initialize the temp variable *tempDominatingNum* to the number of points in the cell. This is because although each cell is in fact a range of values along each dimension, we take the smaller value for each range to represent the cell.

The procedure *Compress()* in Line 3 of the function *Enumerate()* is used for compressing DADA. We discuss this next.

4.2.2 Compressing DADA

In this subsection, we propose a method to partition and compress DADA to support efficient searching.

Definition 4.5. **Equivalence Class** *Given a dominating lattice $\mathcal{L}$, a set of cells in $\mathcal{L}$ is said to belong to the same equivalence class, CL, if:*

1. *Given any two cells c, c' in $\mathcal{L}$ which satisfy $c \preceq c'$, any intermediate cell c'' satisfying $c \preceq c'' \preceq c'$ is also in CL.*

2. *CL is the* **maximal** *set of cells that: (1) dominate the*

Algorithm 1: DADA(C, n)

```
Input:
  C: A set of points.
  N: The total number of dimensions.
Output:
  A class index tree.
Method:

1: let cell=⟨1,...,1⟩ and Call Enumerate(cell, C, 1);
2: construct the D*-tree and output it
```

```
Function Enumerate(cell, input, dim)
Input:
  cell: the cell to be processed.
  input: the point partition.
  dim: the starting dimension for this iteration.

1.  if dim==N+1 do
2.    ComputeDN(cell, DnCount, dim-1)
3.    Compress(cell)
4.  end if
5.  for D=dim to N do
6.    partition input on dimension D
7.    for i=cardinality[D] to 1 do
8.      part=point partition for value $x_i$ of dimension D
9.      if |part| ==0 do
          ProcessDescendants(cell, part, D+1);
10.     else
11.       let cell[D]=i
12.       Enumerate(cell, part, D+1)
13.     end if
14.   end for
15. end for
```

Algorithm 2 ComputeDN(cell, DnCount, dim)

```
1.  Initialize tempDominatingNum to be numcell⟨cell⟩;
2.  for i=dim to 1 do
3.    if cell[i]<cardinality[i] do
4.      tempcell=child(cell,i)
5.    //compute tempindex of tempcell in subspace $\{D_i,...,D_N\}$
6.      tempDominatingNum += DnCount[i][tempindex];
7.      //compute index of cell in subspace $\{D_i,...,D_N\}$
8.      DnCount[i][index] = tempDominatingNum;
9.    end if
10. end for
```

Algorithm 3 Compress(cell)

```
1.  for d=N to 1 do
2.    if cell[d]<cardinality[d] do
3.      tempcell=child(cell,d)
4.      if tempcell is upper of existing class CL in temp class list
5.        if cell.Dn==tempcell.Dn
6.          merge cell into CL, CL.upp=cell, CL.num += cell.num
7.        else remove CL from temp class list and output it
8.    end if
9.  end for
10. if cell does not merge into any existing class do
11. //create a new class $CL_m$, insert it into the temp class list
12.   $CL_m$.upp=$CL_m$.low=cell, $CL_m$.Dn=cell.Dn
      $CL_m$.num=cell.num
13. end if
```

same set of points, and (2) are bounded in a regular minimum bounding box (MBR).

□

The first condition of Definition 4.5 ensures that the cell partition is convex. The second condition ensures that the cell partition is maximal while having a regular rectangular shape. For example, assuming the dataset C has only one single point ⟨1, 2⟩(⟨1, 2⟩ means column 1 and row 2) in a space 3×3, Figure 6(a) shows the maximal cell partition according to the number of points dominated (shown as the value of each cell in Figure 6).

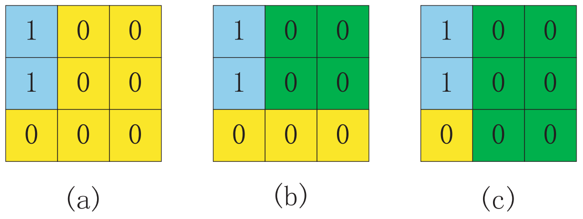


Figure 6: Example of cell partition

Although in this case we obtain the maximal classes, we lose the important property that each class has a unique **upper bound** and **lower bound**.

Definition 4.6. **Upper Bound/Lower Bound** *Given an equivalence class of cells, CL, the* **upper bound** *of CL is a cell p at the corner of the MBR that bounds CL such that p dominates all cells in CL. The* **lower bound** *of CL, is a cell p at the corner of the MBR that bounds CL such that p is dominated by all cells in CL.* □

Since the existence of unique upper and lower bounds is very useful for answering the three types of representative DRQs, we want to keep the unique property of the upper bound for each class while doing cell partition. This is the motivation for the second condition of Definition 4.5.

We use CL.upp, CL.low, CL.num and CL.Dn to represent the upper bound, the lower bound, the number of points contained and the number of points dominated by cells in an equivalence class CL, respectively. For example, the equivalence class in the green color region in Figure 6(b) can be represented as CL={⟨2,1⟩, ⟨3,2⟩, 0}. That is, CL.upp is the cell ⟨2,1⟩, CL.low is the cell ⟨3,2⟩, and CL.Dn is 0.

After having computed the value for each cell, the procedure *Compress()* in Algorithm 3 is immediately called by the function *Enumerate()*. It iterates through all dimensions, and checks if the current cell can be merged with one of its children (Lines 1-8). If it cannot merge into any existing class, it forms a new class CL_m itself (Lines 9-12). Note that CL_m is a temp class and cannot be output yet. It may merge with one of its parent cells later. A temp class list is maintained to keep all temp classes in reverse lexicographical order of upper bounds.

After a new temp class is created, it is inserted at the end of the list. Once we find that a cell cannot merge with one of its child temp classes CL_t, we delete CL_t from the temp class list and output it.

Note that the output order from head to tail is very important when the classes in the temp class list are output. It guarantees all classes are output in reverse lexicographical order. This order helps improve the efficiency of index construction, which we will discuss below.

The gray cells in Figure 5 show the class upper bounds produced by Algorithm 1 when the dataset in Figure 4 (a) is used.

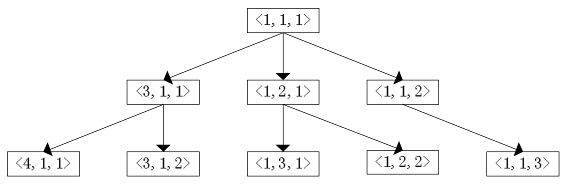


Figure 7: D*-tree

4.2.3 *Construction of D*-tree*

Since all cells in an equivalence class dominate the same set of points, and there exists a unique upper bound for each class, we can use the upper bound as a representative for answering queries. To enable efficient query processing, an index structure (called D*-tree in this paper, short for DADA index tree) can be constructed using the set of class upper bounds in a dominating lattice.

DEFINITION 4.7. **D*-tree**
Given a set of upper bounds from the equivalence classes, a D-tree is a tree such that:*

1) There is a node representing each upper bound of the equivalence classes.

2) There is an edge from one node (Node 1) to another (Node 2) if and only if:

- *The upper bound of Node 1 dominates the upper bound of Node 2.*
- *The upper bound of Node 1 is the nearest to Node 2 based on the lexicographical order among all nodes that dominate Node 2.*

□

For any cell in a dominating lattice, we can quickly obtain its values by simply accessing a node in the D*-tree corresponding to its class upper bound. Details will be discussed in Section 5.

To construct a D*-tree, we only need to read in all classes which we have output in lexicographical order. This is facilitated by the way we output the classes at the compression stage. As each class is read, we create its representing node, find its nearest parent node p on the D*-tree, and insert the node as the child of p. Figure 7 shows the D*-tree corresponding to the cells in Figure 5 .

4.2.4 *Pruning*

We now describe the pruning strategy used in DADA to improve its computation efficiency. Our emphasis here is to show that our pruning step does not prune off any equivalence classes while preventing fruitless computation. Combining this with our earlier explanation on how DADA is computed and compressed without the pruning step, the correctness of our algorithm is obvious.

The pruning explores the BUC-like property that whenever a partition on some dimension d contains an empty set of points, the number of points dominated by all cells expanded from this partition in subspace $\{D_i,...,D_N\}$ (i=d+1,...,N) will be 0.

For example, the fourth partition on the first dimension in Figure 5 contains an empty set of points. Cells expanded from this partition are: ⟨4, 1, 1⟩ , ⟨4, 1, 2⟩, ..., ⟨4, 3, 4⟩). Their dominating number in subspaces $\{D_2, D_3\}$ or $\{D_3\}$ is 0.

The procedure *ProcessDescendants()* in Line 9 of Algorithm 1 is implemented for pruning. It calls procedures *ComputeDN()* and *Compress()* for cells expanded from partition *part* just as with function *Enumerate()*. The major difference is that since all cells from the *part* onward dominate no point, *ComputeDN()* now need only iterate for the first D subspaces i.e., only from D to first dimension and not from the last dimension to the first dimension as shown in Line 2 of Algorithm 1.

By avoiding the aggregation for subspaces and partitions that do not dominate any point, considerable cost saving can be made on sparse datasets. In addition, our method has the potential to adopt iceberg-cube [3] like pruning i.e., given a user-specified threshold, we prune away all computation for cells that dominate too few points.

5. QUERY ANSWERING USING DADA

In this section, we propose efficient algorithms to answer DRQs using D*-trees on DADA. The key idea is to make efficient use of D*-trees in filtering unnecessary checks.

5.1 Linear Optimization Query

Given a plane L, a set of objects C in space D, we wish to find some cells which intersect L and dominate the most points in C.

An obvious method for processing such queries is as follows: First, start a depth-first search from the smallest cell ⟨1,...,1⟩. At any stage, if the cell is at the bottom-left of the plane L (referring to Figure 2), iterate continually on its children. Otherwise, stop the iteration, and if the cell is exactly on the plane L, add it to the result cell set. Second, return those cells which have the maximal dominating number from the result cell set. This method is clearly not efficient.

To improve efficiency, we perform the iterative procedure on the D*-tree from the root. At each node along the path, we can determine if L cuts through some cells in an equivalence class CL by checking whether $CL.upp$ and all the upper bounds of its children in the D*-tree fall on the right-upper side of L. If this is the case, CL and all the nodes below it in the D*-tree can be ignored.

Figure 8 shows an example in which a dominating lattice is partitioned into seven classes. Circles represent the upper bounds of equivalence classes. Class ⟨1,1⟩ has two children, ⟨2,1⟩ and ⟨1,4⟩. Since child ⟨2,1⟩ falls on the same side relative to the plane L as ⟨1,1⟩ does, and ⟨1,4⟩ is exactly on the plane L, we can infer that the class ⟨1,1⟩ does not intersect with plane L. All cells in ⟨1,1⟩ will not satisfy the query condition. Thus, ⟨1,1⟩ is excluded from the result class set. We next look at class ⟨2,1⟩. Since it has a child ⟨4,1⟩ which falls on the different side of plane L, ⟨2,1⟩ must be cut through by the plane L and is put into the result set. Note that at this time, we do not have to further iterate on the children of ⟨4, 1⟩ (such as ⟨6,1⟩ and ⟨4,3⟩).

We outline the LOQ query processing algorithm in Algorithm 4. The last three lines are used to process the special case where some classes have no immediate child class on some dimension. For the example shown in Figure 8, class ⟨2,1⟩ has no immediate child class on dimension Y, and class ⟨1,4⟩ has no child class on dimension X. In this case, even though all their children fall on the same side, we cannot

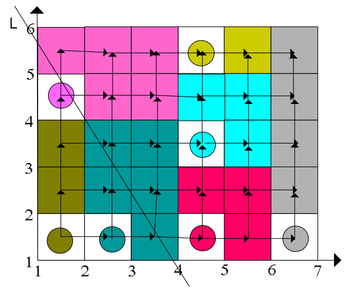


Figure 8: Example of LOQ Processing

```
Algorithm 4: LOQ_Processing(T , L)
Input:
  T: A D*-tree.   L: A plane. Output:
  LOQ(L, C, D) and a point set.
Method:
1: Initialize Node to the root node of the T
2: Find CSet using LOQ_SearchTree(newRoot, L)
3: Pick class CL$_{max}$ which dominate most points
from CSet
4: Return the cells and their value in $CL_{max}$

Function LOQ_SearchTree(Node, L)

1.  if Node.upp is at the bottom−left of L
2.    for each child c$_i$ of Node
3.      if c$_i$.upp is at the bottom-left of L
4.        LOQ_SearchTree(c$_i$, L)
5.      else //$c_i$.upp is on L or at the up-right of L
6.        if c$_i$.upp is on the plane of L
7.          put c$_i$ to the result class set CSet
8.        else //$c_i$.upp is at the up-right of L
9.          if Node is not in CSet, put it in
10.   if Node$\neg\in$Cset$\bigwedge$Node.ChildNum<N
11.     if Node.low is on L or at the up-right of L
12.         put Node in CSet
```

exclude them from the result set.

5.2 Subspace Analysis Query

Given a set of points C and a point p in the N-dimension of D, the most basic SAQ is to compute for each subspace D$'$, the number of points dominating or dominated by p. To answer subspace query on D' using DADA, we only have to compare p to all points in C in the dimensions of D' and ignore the effect of other dimensions. This implies that as long as p dominates a point q based on D', their relation in $D - D'$ is of no consequence.

To simulate this, we create a point p' such that $p'_i = p_i$ if $D_i \in D'$ and $p'_i = 1$ if $D_i \in (D - D')$. By setting the value of dimensions not in D' to 1, we remove the effect of these dimensions, and p' will dominate another point so long as that is true in D'.

As an example, Figure 9(a) shows a set S which includes

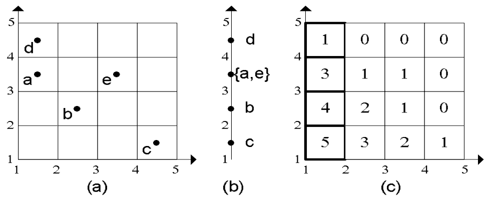


Figure 9: Example of SAQ Processing

five two-dimensional points. Figure 9(b) plots the projections of the five points in Figure 9(a) in subspace $\{Y\}$. Figure 9(c) shows the corresponding dominating cube of point set S in Figure 9(a). We can easily see that for any point p in subspace $\{Y\}$, we can get its dominating number by querying the left most column of the dominating cube in the full space. For example, the dominating number of point a in subspace $\{Y\}$ is 3, and that of c is 5. Likewise, for any point p in subspace $\{X\}$, we can get its dominating number by querying the bottom row of the dominating cube in the full space.

After having computed p' from p and D', it is relatively easy to search for p' by going down the D*-tree. All that needs to be done is to start from the root of the tree and move down the node with the upper bound that can dominate p'. By comparing p' against the upper bound and lower bound of a class, it is possible to identify whether p' is contained in the class. Once this is found, the answer can be output.

5.3 Comparative Dominant Query

Comparative dominant queries are those queries which aim to compare the set of dominated objects between competitive products. So far, we have defined two kinds of CAQs: CDQ$^-$(A,B,C,D), which retrieves |gdominating(A,C,D) - gdominating (B, C, D)|; and CDQ$^\bigcap$(A,B,C,D), which retrieves |gdominating(A,C,D)$\bigcap$gdominating(B,C,D)|.

To handle such queries, we can search the D*-tree from the root level by level. To accumulate the number of points that is dominated by A (or B), we associate a class list with A (or B). Once we find the lower bound of a class CL is dominated by at least one point in A, then all children of CL must be dominated by A. In this case, we need not search the subtrees of CL for A any more. What we need to do is to put CL and all its children to the class list of A and search the subtrees of CL further for B. If we find that the lower bound of a class CL$'$ is dominated by at least one point in B, then all children of CL$'$ must be dominated by B. In this case, the search on subtrees of CL$'$ will be pruned off. We only need to put CL$'$ and all its children into the class list of B. The switch between searching for A and B is controlled by the variable *Flag*.

If the lower bound of CL (or CL$'$) is not dominated by any point in A (or B) , the children of CL (or CL$'$)need to be recursively processed. After we finish the search of the D*-tree, we get two class list, one is for A and one is for B. What we need to do now is to accumulate the *num* of all classes on the same list. The algorithm of the CDQ answering algorithm is given in Algorithm 5. For brevity, we suppress further details.

6. PERFORMANCE ANALYSIS

To evaluate the efficiency and effectiveness of DADA, we conducted extensive experiments. We implemented all algorithms using Microsoft Visual C++ V6.0, and conducted the experiments on a PC with Intel Pentium 4 2.4GHz CPU, 3G main memory and 80G hard disk, running Microsoft Windows XP Professional Edition. We conducted experiments on both synthetic and real life datasets. However, due to space limitation, we will only report results on synthetic datasets here. Results from real life datasets mirror the result of the synthetic datasets closely.

To examine the effects of various factors on the perfor-

Algorithm 5: CDQ_Processing(T , A, B)

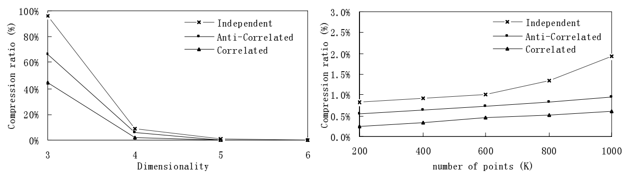
```
Input:
    T: A D*-tree.
    A, B: two object sets.
Output:
    $CDQ^{-}$(A,B,C,D) and $CDQ^{\cap}$(A,B,C,D).
Method:
1: Initialize Node to the root node of the T
2: Initialize ListA and ListB to null;
3: CDQ_SearchTree(Node, A, ListA, 1)
4: Accumulate the num of all nodes on listA to numA
5: Accumulate the num of all nodes on listB to numB
6: $CDQ^{\cap}$(A,B,C,D)=numB
7: $CDQ^{-}$(A,B,C,D) = numA - numB

    Function CDQ_SearchTree(Node, S, L, Flag)

1.  if Node.low is dominated by any point in S
2.      put all nodes on the subtree rooted with Node to list L
3.      if Flag==1 //need search further for object set B
4.          CDQ_SearchTree(Node, B, ListB, 0)
5.      else
6.          return
7.  else
8.      for each child $c_i$ of Node
9.          CDQ_SearchTree($c_i$, A, ListA, 1)
```

mance of DADA, we generated three popular synthetic skyline data sets: *correlated, independent* and *anti-correlated*, using the data generator provided by the authors in [5]. For each type of data distribution, we generated data sets of different sizes (from 100,000 to 1000,000 tuples) and of dimensionality varying from two to six. The default values of dimensionality and data size were 5 and 100,000 respectively. The default value of cardinality for each dimension was 50.

Effectiveness of Compression: In this experiment, we explored the compression benefits of DADA using a metric called compression ratio, defined as the size of compressed DADA as a proportion of uncompressed DADA. As such, a smaller ratio implied better compression. We stored a compressed DADA by explicitly storing a lower bound and an upper bound for each class.

Figure 10(a) shows the compression ratio for each type of data distribution with increasing dimensionality. Clearly, we can obtain comparable compression ratio on all three data sets. Among the three distributions, correlated data gives the most compression as many cells in DADA dominate the same set of points. Anti-correlated data sets are second in terms of compression ratio. Similar to correlated data, the skewness of anti-correlated data results in many points being grouped together, resulting in a reasonable number of cells dominating the same set of points. However, their direction of correlation is orthogonal to the direction of dominance. As such, the number of cells dominating the same points is still smaller for anti-correlated data compared to correlated data resulting in more equivalence classes being generated. Independent data has the least compression as it is more difficult for cells to dominate the same set of points. From Figure 10(a), we can see that the higher the dimensionality is, the better the compression ratio will be. This happens because the cube gets more sparse when dimensionality increases while the number of points remains the same.

Figure 10(b) shows how effectiveness of DADA scales up



(a) Varying Dimensionality (b) Varying Data Size

Figure 10: Compression Ratio

as the number of points goes up. We can see that compression performance gets worse as the number of points increases. This is because DADA becomes more dense when more points are added.

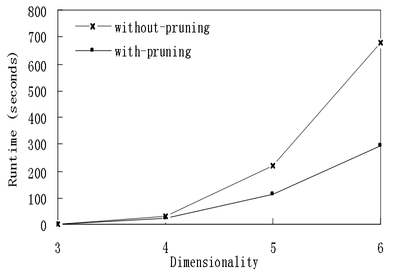
Efficiency of Computation: To evaluate the efficiency of our algorithm for computing DADA, we fixed the number of points at 100k and varied the number of dimension from three to six. Figure 11(a), Figure 11(b) and Figure 11(c) show the run time of the basic algorithm (*without-pruning*) against that of the improved algorithm (*with-pruning*) for computing DADA on three types of data sets.

From the results, we can see that pruning outperforms *without-pruning* on all data sets. As we have discussed, pruning can reduce computation for those partitions which dominate an empty point set in some subspaces. In correlated data sets, pruning helps halve running time. The difference between the results with and without pruning increases with dimensionality. The reason is that as dimensionality increases, the cube becomes more sparse and pruning the large number of empty cells yields significant saving.

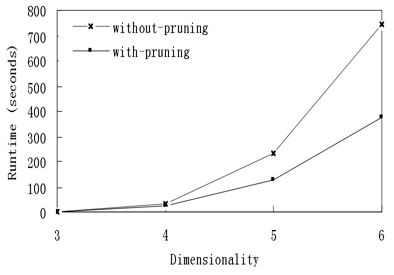
Since an exhaustive search has to be carried out if no pruning is done, the performance of the *without-pruning* approach is very sensitive to number of dimensions but less sensitive to data distribution. When pruning is done, dimensionality has less effect while data distribution becomes an important factor. From Figure 11(c), we can see that the performance difference between the two approaches on the independent data sets is not as large compared to the other two datasets.

Scalability: Next, we look at the run time of the two algorithms as the number of points increases. Since the trends are the same for all three data sets, we only show the results on the data sets with independent distribution as it is the most difficult to compress. We increase the number of points from 200k to 1 million. Figure 12(a) shows that although both algorithms are of linear scalability, the run time of the *without-pruning* algorithm scales better than that of the *with-pruning* algorithm. As the number of points increases, DADA becomes denser. The *without-pruning* approach is not greatly affected since it is insensitive to how dense the cube is. The *with-pruning* approach slightly worsens since dense data provides fewer chances for pruning to be done.

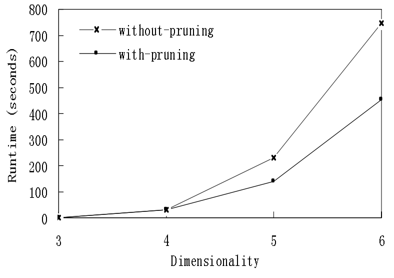
Effect of Cardinality: To evaluate the effect of cardinality on our techniques, we varied the cardinality of the data sets from 30 to 70. Figure 12(b) shows the run time of both algorithms against varying cardinality. We can see that as cardinality increases, the run time of the *without-pruning* approach increases rapidly while that of the *with-pruning* approach increases only modestly. This is because as car-



(a) Correlated

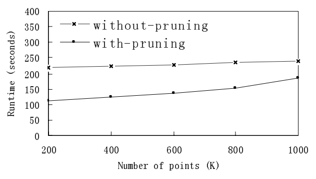


(b) Anti-Correlated

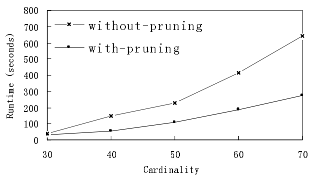


(c) Independent

Figure 11: Run Time vs. Dimensionality



(a) Varying Data Size



(b) Varying Cardinality

Figure 12: Running Time (Independent)

dinality increases, the cube becomes sparse and the *with-pruning* approach is able to make use of the sparseness to reduce running time.

Query answering performance: In this experiment, we evaluated the query answering performance of DADA. We implemented our query processing algorithm in two ways: In the first, we performed (*with-pruning*) compression on DADA and indexed the equivalence partitions using the D*-tree. In the second, we performed (*without-pruning*) scanning of individual cells.

We also compare our algorithms with a (*Naive*) method which simply stores all of the data points in an array. To answer a query, the *Naive* method will simply scan through the array and count dominating or dominated points. A brief outline of such method is given as follows:

- For LOQ, the *Naive* method scans the data and keeps one counter for every point p$'$ on the test plane. For each point p in the data set, we increment the p$'$ counter if p$'$ dominates p.
- For SAQ, we perform a linear scan on the data and for each data point p$'$, we check if the query point p is dominating or dominated by p$'$ in the given subspace. A counter (initially set to 0) will only be incremented if the result is positive. At the end of the scanning, the counter will be returned as the query result
- For CDQ, a linear scan is performed on the dataset and for each data point p, we check whether it is dominated by any object in A. If p is dominated by at least one object in A, we check whether it is also dominated by an object in B. If this is also true, we increment the counter for CDQ$^{\cap}$(A,B,C,D), otherwise, the counter for CDQ^{-}(A,B,C,D) will be increased.

We first randomly generated 10,000 different LOQs based on the synthetic data set. As the correlated and anti-correlated data sets contain too few equivalence classes to be interesting after compression, we present only the result on the independent data set.

Figure 13(a) shows query time against dimensionality. We can see that the two algorithms, *with-pruning* and *without-pruning*, outperforms the *Naive* method. This is because *Naive* method needs to do a lot of pairwise comparison between points. It can also be seen that *with-pruning* approach outperforms the *without-pruning* approach. The performance difference increases with dimensionality. This is because as the dimensionality increases, the number of cells increases drastically while the size of the D*-tree does not significant increases due to the compression. Because of this, the *without-pruning* approach entails much more checking than the *with-pruning* approach.

To test the effect of SAQ queries, we randomly generated 10,000 different points. For each point p, we queried its dominating number in all subspaces. Figure 13(b) shows query time against dimensionality. We can see that the *with-pruning* approach is clearly the best.

To test the effect of CDQ queries, we randomly generated 10,000 different CDQs based on the synthetic data set with in A and B containing 100 points each. Figure 13(c) shows query time against dimensionality. As expected, the with-pruning approach performs the best. From these results, we can see that the D*-tree together with a compressed DADA is very useful for answering of DRQs.

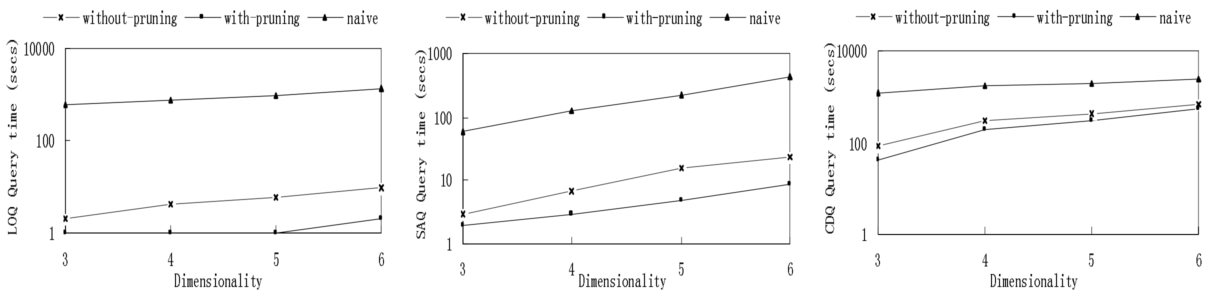
7. CONCLUSION

In this paper, we have introduced Dominant Relationship Analysis and proposed three types of queries to illustrate the various aspects of this new form of analysis. To support the queries, we have further proposed a novel data cube called DADA. The results of our extensive experiments show that DADA can be computed efficiently and is useful for handling the three types of queries. In our future work, we will look at how these three types of queries can be integrated efficiently and elegantly.

8. REFERENCES

[1] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan, and S. Sarawagi. On the Computation of Multidimensional Aggregates. In *VLDB*, pages 506–521, 1996.

[2] D. A. K. Alexander Hinneburg. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB*, 1999.



(a) LOQ Query Processing (b) SAQ Query Processing (c) CDQ Query Processing

Figure 13: Query Answering (Independent): Time vs. Dimensionality

[3] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 359–370, 1999.

[4] G. Birkhoff. *Lattice Theory.* American Mathematical Society Colloquium Publications, Rhode Island, 1973.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[6] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *KDD*, pages 254–260, 1999.

[7] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skyline in high dimensional space. In *ACM SIGMOD*, 2006.

[8] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[9] Q. Chen, M. Hsu, and U. Dayal. A data-warehouse/OLAP framework for scalable telecommunication tandem traffic analysis. In *ICDE*, pages 201–210, 2000.

[10] B. Davey and H. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[11] M. Ester, R. Ge, W. Jin, and Z. Hu. A microeconomic data mining problem: customer-oriented catalog segmentation. In *KDD*, pages 557–562, 2004.

[12] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, 11:246–267, 1995.

[13] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE*, pages 152–159, 1996.

[14] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

[15] J. Han. Olap mining: Integration of olap with data mining. In *Database Semantics-7*, pages 3–20, 1997.

[16] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD*, pages 205–216, 1996.

[17] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *SIGMOD Conference*, pages 73–88, 1997.

[18] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. In *STOC*, 1998.

[19] J. Kleinberg, C. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. In *Data Min. Knowl. Discov.*, 2(4): 311-322, 1998.

[20] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, 2002.

[21] C. Li, G. Cong, A. K. H. Tung, and S. Wang. Incremental maintenance of quotient cube for median. In *KDD*, pages 226–235, New York, NY, USA, 2004. ACM Press.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.

[23] K. Ross and D. Srivastava. Fast Computation of Sparse Datacubes. In *VLDB*, pages 116–125, 1997.

[24] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos. Cubetree: organization of and bulk incremental updates on the data cube. In *ACM SIGMOD*, pages 89–99, 1997.

[25] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: shrinking the petacube. In *SIGMOD Conference*, pages 464–475, 2002.

[26] K. L. Tan, P. K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.

[27] K. Wang, S. Zhou, and J. Han. Profit mining: From patterns to actions. In *EDBT*, pages 70–87, 2002.

[28] R. C.-W. Wong, A. W.-C. Fu, and K. Wang. Mpis: Maximal-profit item selection with cross-selling considerations. In *ICDM*, pages 371–378, 2003.

[29] J. T. Yao. Sensitivity analysis for data mining. In *Proceedings of The 22nd International Conference of NAFIPS (the North American Fuzzy Information Processing Society)*, pages 272–277, 2003.

[30] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of skyline cube. In *VLDB*, pages 241–252, 2005.

[31] Z. Zhang, X. Guo, H. Lu, A. K. H. Tung, and N. Wang. Discovering strong skyline points in high dimensional spaces. In *CIKM*, pages 247–248, 2005.