PAPER

Efficient Analyzing General Dominant Relationship based on Partial Order Models

Zhenglu YANG[†], **Lin LI**[††], ***Nonmembers, and*** **Masaru KITSUREGAWA**[†††], ***Fellow***

SUMMARY Skyline query is very important because it is the basis of many applications, e.g., decision making, user-preference queries. Given an N-dimensional dataset D, a point p is said to dominate another point q if p is better than q in at least one dimension and equal to or better than q in the remaining dimensions. In this paper, we study a generalized problem of skyline query that, users are more interested in the details of the dominant relationship in a dataset, i.e., a point p dominates how many other points and whom they are. We show that the existing framework proposed in [17] can not efficiently solve this problem.

We find the interrelated connection between the partial order and the dominant relationship. Based on this discovery, we propose a new data structure, $ParCube$, which concisely represents the dominant relationship. We propose some effective strategies to construct $ParCube$. Extensive experiments illustrate the efficiency of our methods.

key words: *skyline query, algorithm, dominant relationship analysis, performance evaluation*

1. Introduction

The skyline query [3] has attracted considerable attention these years because it is the basis of many applications, e.g., multi-criteria decision making [3], user-preference queries [11] [9] and microeconomic analysis [17]. Skyline mining aims to find those points, which are not dominated by others, in a d-dimensional spatial dataset. This problem can be seen as a special class of pareto preference queries [11], convex hull [23] or maximum vectors [14]. Fig. 1 shows one classic example of skyline query that customers are always interested in those "best" hotels that are better than others at least at one of the two criteria, the distance and the price, with smaller values. The skyline of the example dataset in Fig. 1 consists of a and c.

There are many issues related to skyline query, including the general full-space skyline points querying [3] [7] [13] [21], subspace skyline points mining [31] [26] [28], skyline points extracting in stream [18] [27] [20], Top-k and high-dimensional skyline points extracting [5] [6], mining skyline in distributed environments [2] [10] [29], approximate skyline querying [12]. All these issues, however, concerned only the pure dominant relationship among a dataset, i.e., a point p is whether dominated by others or not, and got those non-dominated ones as results.

[†]The author is a research associate in the Institute of Industrial Science, the University of Tokyo.

[††]The author is an assistant professor in the Department of Computer Science and Technology,& Wuhan University of Technology, China.

[†††]The author is a professor in the Institute of Industrial Science, the University of Tokyo.

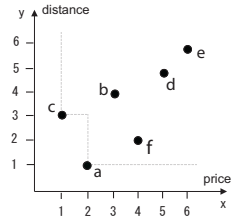


Fig. 1 Example of the skyline query

Recently, Li et al. [17] proposed to analyze the dominant relationship in a business model that, users are more interested in the detail of the dominant relationship in a dataset, i.e., a point p dominates how many other points and is dominated by how many others. Here we show an example.

Example 1: Consider you are a manager of hotel company. You want to know the business position of a local hotel b in the current market with regard to your preference, i.e., price and distance to the beach, by checking how many other hotels are better/worse than b. For the sample hotels shown in Fig. 1, you can deduct the conclusion that hotel b is better than 2 other hotels but worse than another 2 hotels with regard to your preference*.

In real world, however, users are always interested in not only "how many" objects are dominating/dominated by a specific object, but also "whom" they are, which was not mentioned in [17]. This problem can be seen as a general dominant relationship analysis to the ones proposed in [17]. It is naively thought, can be easily solved by associating each object with its corresponding cuboid in DADA [17]. So when users query the dominant relationship, these objects will be extracted simultaneously. Nevertheless, due to a huge number of duplicate existence in DADA, the storage overhead and the query time will be unacceptable for users. In this paper, we aim at proposing efficient and effective methods to answer the "whom" problem. Because of the interrelated connection between the partial order and the dominant relationship, we propose a new data structure called $ParCube$, which concisely represents the complete information of the general dominant relationship based on the partial order analysis. Specifically, we record the partial order as a Directed Acyclic Graph (DAG) for each

*Note that the analysis here can be further used to determine the price of a hotel, which should be competitive in the current market while reserving the most profit.

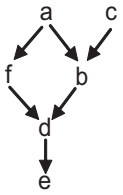


Fig. 2 DAG representation in 2-d space

cuboid in $ParCube$ and propose efficient data structures and strategies to answer the general dominant relationship queries. Moreover, we introduce efficient strategies to construct $ParCube$. The experimental results and performance study confirms the efficiency and effectiveness of our strategies.

To illustrate the core idea of this paper, here we show a simple example. Fig. 2 represents the partial order (encoded as DAG format) of the example dataset in Fig. 1 in 2-dimensional space. We can know the point b dominates the points d and e and is dominated by the points a and c, by counting the $out\text{-}link$ and $in\text{-}link$ of d, respectively.

Here we solve not only the $how\ many$ problem, but also the $whom$ problem. From this example, we know that the general dominant relationships of a dataset can be represented into their corresponding partial order representation (i.e., DAGs). In contrast, the DADA data structure [17] applies the grid-based index technique, which does not efficiently record the dominant relationship, as will be illustrated in the experimental evaluation.

Our contributions in this paper are as follows:

- We generalize the dominant relationship queries proposed in [17], as **General Dominant Relationship Query** (**GDRQ**). We find the interrelated connection between GDRQ and the partial order analysis.
- We propose a data cube, $ParCube$, which concisely represents the complete information of the general dominant relationship as DAGs based on the partial order for each cuboid. We introduce effective methods to construct the $ParCube$.
- We conduct comprehensive experiments to illustrate the effectiveness and efficiency of our methods.

The remainder of this paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we present the preliminaries of this paper. A naive method based on existing strategy to answer $GDRQ$ is introduced in Section 4. The computation of $ParCube$ is presented in Section 5.1 and the query processing strategies for generalized dominant relationship analysis using $ParCube$ is described in Section 5.2. The performance analysis are reported in Section 6. We conclude the paper and provide suggestions for future work in Section 7.

2. Related work

2.1 Skyline Query

Skyline query was first introduced in [3]. The problem comes from some old classic topics, such as convex hull [23] and maximum vectors [14].

Skyline query algorithms can be classified into two categories. The first one is non-index based method, i.e., BNL [3], SFS [7], DC [3]. The second category is index based method, i.e., NN [13], BBS [21], SUBSKY [26]. As expected, the index-based methods have been shown to be superior over the non-index-based ones and furthermore, the index-based strategies can progressively return answers without having to scan the entire data input. Specially, SUBSKY [26]† was proposed to compute low-dimensional Skylines and is the best algorithm for subspace skyline discovering. Based on the data distribution, SUBSKY creates an anchor point for each cluster, and builds a B+-tree on the L_∞ distance between each object to its corresponding anchor. Then, SUBSKY scans the tree leaf nodes according to the ascending order of the points's smallest value of d-dimension to get Skylines.

From the view point of dimension concerned, the existing algorithms can be also classified into two categories, i.e., full space based method [13] [21], and subspace based method [26] [31] [28]. Other related work on skyline mining includes mining skyline in distributed environments [2] [10] [29], skyline query in data stream [18] [27] [20], approximate skyline query [12], interesting skyline points in high-dimensional space [5] [6].

All the above works concerned only the pure dominant relationship and, outputted those points which are not "dominated" by others. Note that in addition to the original meaning in [3], "dominated" here can be a variant, i.e., k-dominant [6].

In contrast, Li et al. proposed to analyze a more general dominant relationship from a microeconomic aspect [17]. The users are always interested in not only the binary dominant relation between the points in a dataset, but also the statistical information, i.e., how many other points are dominating/dominated by a specific point. In [17], the authors proposed three basic $Dominant\ Relationship\ Queries$ ($DRQs$) and constructed a data cube, DADA, to efficiently organize the information necessary to $DRQs$. Moreover, a novel data structure, D*-tree, was proposed to fulfill efficient computation for $DRQs$.

However, users are always interested in not only "how many" objects are dominating/dominated by a specific object, but also "whom" they are, which was not mentioned in [17]. This problem cannot be easily solved by using the methodologies proposed in [17] because of the large duplicate storage cost in DADA. In this paper, we propose efficient data structure and strategies to solve such kind of general dominant relationship query based on our discovery that GDRQ has interrelated connection with partial order.

†We describe the detail of SUBSKY because it is one of the baseline algorithms in the experimental evaluation.

2.2 Partial Order Mining

Partial order has appeared in many computational models and there are a lot of applications involves with partial order issues, such as concurrent models [15], optimistic rollback recovery [25], biology [16], security [24] and preference query [11].

In this paper, we mainly consider the problem that how to convert the spatial dataset into partial order representation, which are then queried to get the general dominant relationship efficiently. As far as we know, there is no work on this problem. An interesting study investigated the problem of mining a small set of partial orders globally fitting data best [19]. Particularly, [19] addressed sequence data. Very different from the problem studied here, [19] tried to find one or a (small) set of partial orders that fit the whole dataset as well as possible, which is an optimization problem. An implicit assumption is that the whole dataset somehow follows a global order. More recently, [4] were intended for discovering several small partial orders from a set of sequences instead of only one that describes all or most of the set. They proposed to use closed partial orders to summarize sequential data in a concise manner. Yet different from this paper, they did not further explore the partial orders for a specific purpose (i.e., dominant relationship extraction).

In this paper, however, we need to determinate the partial orders given a spatial dataset. We propose a simple method of converting the spatial dataset to the corresponding sequence dataset and then, apply existing strategies such as that used in [4] with modification by considering skyline property to generate the partial orders.

3. Preliminaries

Given a d-dimension space S=$\{s_1, s_2, \ldots, s_d\}$, a set of points D=$\{p_1, p_2, \ldots, p_n\}$ is said to be a dataset on S if every $p_i \in D$ is a d-dimensional data point on S. We use $p_i.s_j$ to denote the j^{th} dimension value of point p_i. For each dimension s_i, we assume that there exists a total order relationship. For simplicity and without loss of generality, we assume smaller values are preferred [3] (i.e., MIN operation) in this paper.

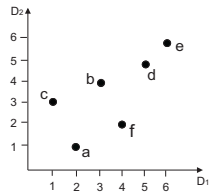
Definition 1 (dominate)**.** *A point p is said to dominate another point q on S if and only if $\forall s_k \in S$, $p.s_k \leq q.s_k$ and $\exists s_t \in S$, $p.s_t < q.s_t$.*

A partial order on D is a binary relation $\preceq$ on D such that, for all x,y,$z \in D$, (i) $x \preceq x$ (reflexivity), (ii) $x \preceq y$ and $y \preceq x$ imply x=y (antisymmetry), (iii) $x \preceq y$ and $y \preceq z$ imply $x \preceq z$ (transitivity). We use (D,$\preceq$) to denote the partial order set (or poset) of D. We denote by $\prec$ the strict partial order on D, i.e., $x \prec y$ if $x \preceq y$ and $x \neq y$. Given x,$y \in D$, x and y are said to be comparable if either $x \prec y$ or $y \prec x$; otherwise, they are said to be incomparable.

The Definition 1 can be translated into the ordering context as follows:

	a	b	c	d	e	f
D_1	2	3	1	5	6	4
D_2	1	4	3	5	6	2
D_3	3	1	6	2	5	4

(a) Example spatial dataset



(b) Representation in 2-d space {D_1,D_2}

Fig. 3 Example dataset

Definition 2 (dominate in ordering context)**.** *A point p is said to dominate another point q on S if and only if $\forall s_k \in S$, $p.s_k \preceq q.s_k$ and $\exists s_t \in S$, $p.s_t \prec q.s_t$.*

The partial order (D,$\preceq$) can be represented by a DAG $G = (D, E)$, where $(\upsilon, \omega) \in E$ if $\omega \preceq \upsilon$ and there does not exist another value $x \in D$ such that $\omega \preceq x \preceq \upsilon$. For simplicity and without loss of generality, we assume that G is a single connected component.

Definition 3 (dominating set, DGS(p, D, S'))**.** *Given a point p, we use DGS(p, D, S') to denote the set of points from D which are dominated by p in the subspace S' of S.*

Definition 4 (dominated set, DDS(p, D, S'))**.** *Given a point p, we use DDS(p, D, S') to denote the set of points from D which dominate p in the subspace S' of S.*

The problem that we want to solve is as follows:

Problem 1 (General Dominant Relationship Query (GDRQ))**.** *Given a dataset D, dimension space S' and a point p, find DGS(p, D, S') and DDS(p, D, S').*

Note that a skyline point p has the following property: DDS(p, D, S')=0. In other words, the skyline query can be thought as a special case of the general dominant relationship query.

Example 1. *Consider the 3-dimensional dataset D = {a, b, c, d, e, f} in Fig. 3 (a). Given a query point b, dimension space S'=$\{D_1, D_2\}$, the dominating set DGS(b, D, S') = {d, e} and the dominated set DDS(b, D, S') = {a, c}. We will use this dataset as a running example in the rest of this paper.*

4. A Naive Method

To solve the problems defined in Section 3, a natural idea is to extend the framework proposed in [17]. In this section, we briefly introduce this naive strategy and then, illustrate its weak points.

The authors in [17] partition the data space by using griding strategy. For example, Fig. 4 (a) shows a dataset in 2-dimensional space (i.e., $\{D_1, D_2\}$). In Fig. 4 (b), each grid records the number of the points which current grid dominates. For instance, the gray grids are all those which dominates three points. Instead of recording each grid information, [17] proposed D^*-tree to record the compressed information (upper/lower bound of a region that dominates

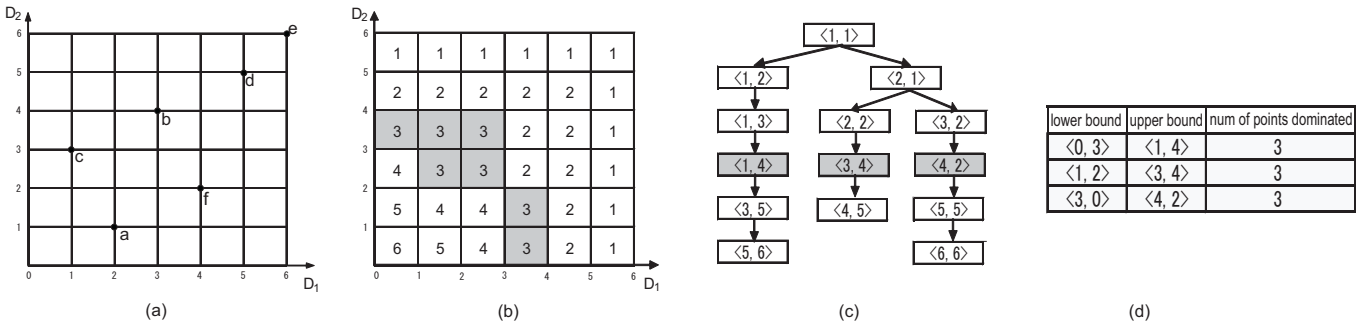


Fig. 4 The strategy of DADA for GDRQ

lower bound	upper bound	num of points dominated	points dominated
<0, 3>	<1, 4>	3	{b, d, e}
<1, 2>	<3, 4>	3	{b, d, e}
<3, 0>	<4, 2>	3	{f, d, e}

Fig. 5 The extension of DADA for general dominant relationship analysis

the same number of points). For example, the gray grids can be partitioned into three regions, which are represented by their upper bound, i.e., $\{1, 4\}$, $\{3, 4\}$ and $\{4, 2\}$, respectively. The whole D*-tree is shown in Fig. 4 (c), which is constructed based on the rule defined in [17] (Definition 4.7). Fig. 4 (d) shows the compressed information about the three gray regions, i.e., the lower bound (1st column), the upper bound (2nd column) and the number of points dominated (3rd column). Given a point P_{query}, to get the number of the points P_{query} dominates, it needs to start from the root of the D*-tree and move down the node with the upper bound that can dominate P_{query}. Once it knows that P_{query} is contained in a region that the node dominates, the desired number of the points which are dominated by P_{query} can be output. Refer [17] for more detail.

Yet there are two issues arising when processing the general dominant relationship queries by using DADA's strategy. Firstly, the "whom" problem can not be efficiently solved. For example, although the gray regions in Fig. 4 (b) all dominate three points, they have different dominating sets, i.e., {b, d, e} for blue and yellow regions and {f, d, e} for red region. Although by adding the dominating set into each node of the D*-tree can naively answer the question (as shown in Fig. 5), this simple solution will introduce serious burden of data duplication problem. Therefore, the strategy of DADA is not appropriate for the general dominant relationship analysis problem. Another issue is that the search strategy in DADA while traversing the D*-tree is inefficient, especially when the tree has many layers.

5. A Partial Order Based Method

In this section, we propose to efficiently apply the properties of the partial order to analyze the general dominant relationship. Specifically, we first introduce effective strategies to construct a partial order data cube ($ParCube$), which concisely represents the dominant relationship by using DAGs. Moreover, we propose efficient algorithms to answer the general dominant relationship queries based on $ParCube$. In the following section, we introduce our methods of constructing $ParCube$.

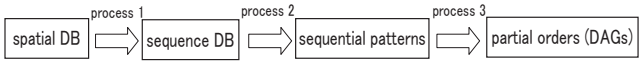


Fig. 6 The work flow of ParCube constructing

5.1 Constructing ParCube

As described in Section 3, the dominant relationship can be encoded in partial order representation (DAGs). In this section, we explain how to construct the partial order data cube (ParCube) with a spatial dataset input. As far as we know, there is no work on this problem. In this paper, we propose to apply strategies from another research context, sequential pattern mining [1], to get the partial order representation from a spatial dataset. The whole work flow is shown in Fig. 6. We propose a simple method of converting the spatial dataset to the corresponding sequence dataset in the first process and then, apply existing strategies such as that used in [4] with little modification in the second and third processes to generate DAGs from the transformed sequence dataset. Note that we mainly illustrate how to compute the cube for a dominating set since computation of a dominated set can be done in a similar fashion.

The first process in Fig. 6 is to convert the original spatial dataset to the sequence dataset. With a k-dimensional dataset, we simply get a k-customer sequence dataset, by sorting the objects in each customer (dimension) according to their value in ascending order. For example, Fig. 7 (b) shows the converted sequence dataset of the example spatial dataset in Fig. 7 (a).

Theorem 1. *The converted sequence dataset records all the dominant relationship of the points in the spatial dataset.*

Proof. Trivial because the small-large pair (dominant) relationship in the spatial dataset is equivalent to the early-late pair (dominant) relationship in the converted sequence dataset. □

The second and the third processes in Fig. 6 aim to determine a partial order that describes the point set in the subspace S' of data space S in D'. The related problem is

	a	b	c	d	e	f
D_1	2	3	1	5	6	4
D_2	1	4	3	5	6	2
D_3	3	1	6	2	5	4

process 1

Dim.	Sequence
D_1	c a b f d e
D_2	a f c b d e
D_3	b d a f e c

(a) Example spatial dataset (b) Transformed sequence dataset

Fig. 7 Process 1 of constructing ParCube

addressed in [19] and more recently in [4]. In this paper, we simply apply the approach in [4] with a minor modification that, instead of mining closed sequential patterns [30], we mine general sequential patterns [1]. In process 2 as shown in Fig. 6, we discover the sequential patterns from the transformed sequence dataset by applying PrefixSpan algorithm [22] †, which is the state-of-the-art one.

Specifically, given a n-sequence dataset, we partition it into several k-sequence datasets, where $2 \leq k < n$, and apply PrefixSpan to them, respectively, with minimum support equal to 100%. For example, given the sequence dataset as shown in Fig. 8 (a), we partition it into k-sequence datasets where k=2, i.e., $\{D_1, D_2\}$, $\{D_1, D_3\}$, and $\{D_2, D_3\}$. PrefixSpan is then applied on them. Note that for k-sequence datasets where k=1, i.e., $\{D_1\}$, $\{D_2\}$, and $\{D_3\}$, we do not need use PrefixSpan because the maximal sequential patterns are straightforward (i.e., the sequence itself). For k-sequence datasets where k=n, i.e., k=3 for the dataset shown in Fig. 8 (a), we do not need to partition it because the number of the possible partitioned dataset is one, i.e., $\{D_1, D_2, D_3\}$.

In fact, the process is the same as building common data cube, that we traverse every possible subspace (a k-sequence dataset, i.e., $\{D_1, D_2\}$), and apply PrefixSpan on it with minimum support equal to 100%.

To save space and convenient the query processing, we merge these sequential patterns as $local\ maximal\ sequential\ sequences$ [1], which are not the subsequence of other sequential patterns in the same subspace. For example, in subspace $\{D_1, D_2\}$, although there are many sequential patterns, i.e., $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle d\rangle$, $\langle e\rangle$, $\langle f\rangle$, $\langle ab\rangle$, $\langle ad\rangle$, $\langle ae\rangle$, $\langle af\rangle$, and so forth. We only record the maximal sequential patterns, i.e., $\langle afde\rangle$, $\langle abde\rangle$ and $\langle cbde\rangle$, because all the other sequential patterns are subsequences of these three maximal sequential patterns.

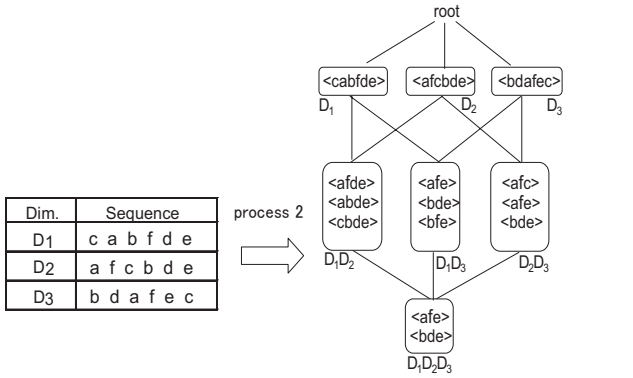
The maximal sequential patterns of a subspace S record the dominant relationship between items in S (as be verified by Theorem 1, Theorem 2). For example, the pattern $\langle afde\rangle$ indicates that a dominates f, dominates d, and dominates e in subspace $\{D_1, D_2\}$.

The result data cube ($SeqCube$) got from process 2 for the example dataset is shown in Fig. 8 (b).

Theorem 2. *SeqCube records all the dominant relationship of the points in the sequence dataset D.*

Proof. (Proof by Contradiction.) For simplicity, we only prove for a specific subspace of $SeqCube$. Assume to the contrary that there is a dominant relationship between two points, a dominates b in a subspace S', is not represented in the cuboid S' of $SeqCube$. This means that the sequential pattern $\langle ab\rangle$ is not listed in S' of $SeqCube$, which contradicts our assumption that the sequential pattern mining process can find all the sequential patterns. □

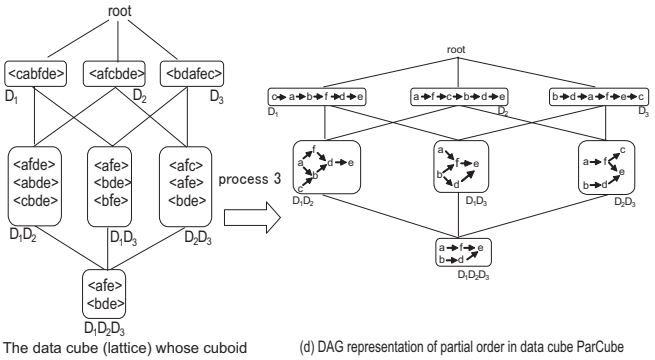
†Due to limited space, we skip the detail of PrefixSpan here. Interested users can refer [22].



(a) Transformed sequence dataset (b) The data cube (lattice) whose cuboid consists of the local maximal common sequential patterns

Fig. 8 Process 2 of constructing ParCube

In process 3, the combinations of the $local\ maximal\ sequential\ sequences$ are enumerated to generate partial orders with DAGs representation, by applying the method proposed in [4]. The result data cube ($ParCube$) got from process 3 for the example dataset is shown in Fig. 9 (b).



(c) The data cube (lattice) whose cuboid consists of the local maximal common sequential patterns (d) DAG representation of partial order in data cube ParCube

Fig. 9 Process 3 of constructing ParCube

Theorem 3. *ParCube records all the dominant relationship of the points in the spatial dataset D.*

Proof. Proof can be deduced based on Theorem 1, Theorem 2 in this paper and [4]. □

5.2 Querying ParCube Data Cube

The semantic meaning kept in the $ParCube$ data cube is the key used to extract the general dominant relationship efficiently.

5.2.1 General Dominant Relationship Query (GDRQ)

Given a dataset D, a query point P_{query} and a subspace S',

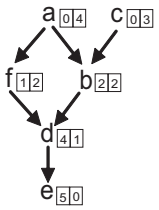


Fig. 10 DAG representation of the example dataset in 2-dimensional space $\{D_1, D_2\}$

the GDRQ is to compute the points dominate or dominated by P_{query}, where $P_{query} \in D$.

An important observation in this case is that, if P_{query} is in D, all the general dominant relationship related to P_{query} can be easily discovered by traversing the DAG in a specific subspace.

As an example, Fig. 10 shows the DAG representation in subspace $\{D_1, D_2\}$. To facilitate the counting process, the numbers of points dominating/dominated by current node (point) are inserted into each node. This process is executed in the precomputed-mode. Suppose the query point is b, we can get the points dominated by b immediately, which is 2. Upon users are interested in whom these two points are, it goes downward following the out-link of b, and gets the dominating set of b as $\{d, e\}$.

In DADA [17] framework, however, it needs to traverse the D*-tree to get the corresponding class. For example, assume the query point is b, the order of the traversed nodes in D*-tree, as shown in Fig. 4 (c), is $\{\langle 1,1\rangle,\langle 2,1\rangle,\langle 2,2\rangle,\langle 3,4\rangle\}$. Then it finds the dominating set of b by checking the class of $\{\langle 3,4\rangle\}$. Obviously, DADA consumes more time compared with our strategy.

6. Experimental Evaluation and Performance Study

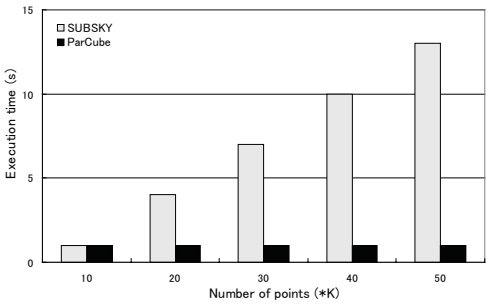
To evaluate the efficiency and effectiveness of our strategies, we conducted extensive experiments. We performed the experiments using a Intel(R) Core(TM) 2 Dual CPU PC (3GHz) with a 3G memory, running Microsoft Windows XP. All the algorithms were written in C++, and compiled in an MS Visual C++ environment. We conducted experiments on both synthetic and real life datasets.

Detailed implementation of the algorithms used to compare is described as follows:

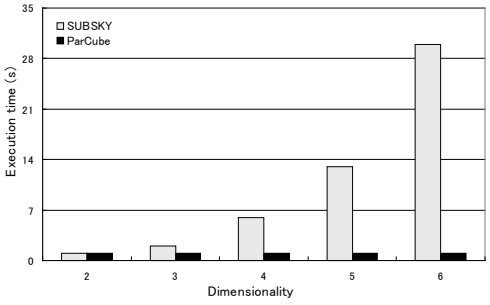
1. *SUBSKY. SUBSKY* was tested with the algorithm developed in [26], which is the state-of-the-art algorithm for subspace skyline query.
2. *Naive. Naive* was tested with the extension of DADA [17], by storing the dominated/dominating points in the corresponding class, as explained in Section 4.
3. *ParCube. ParCube* was implemented as described in this paper.

6.1 Datasets

We employ the synthetic data generator [3] to create our synthetic datasets. They have $independent$ distribution, with



(a) Execution time comparison on Skyline query against number of points (dimensionality=5)



(b) Execution time comparison on Skyline query against dimensionality (point number=50k)

Fig. 11 Execution time comparison between $SUBSKY$ and $ParCube$ on skyline query

dimensionality d in the range [3, 6] and data size in the range [10k, 50k]. The default values of dimensionality were 5. The default value of cardinality for each dimension was 50k.

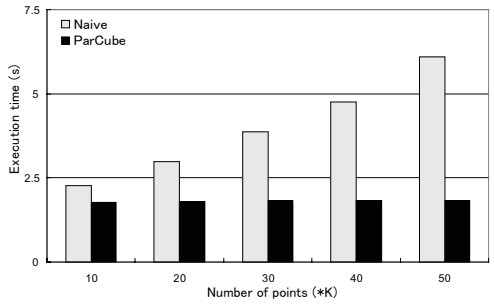
6.2 Skyline Query Performance

Because the skyline query is important and can be seen as a special case of the general dominant relationship query, in this section, we first evaluated the skyline query answering performance of $ParCube$ compared with the state-of-the-art algorithm, $SUBSKY$ [26].

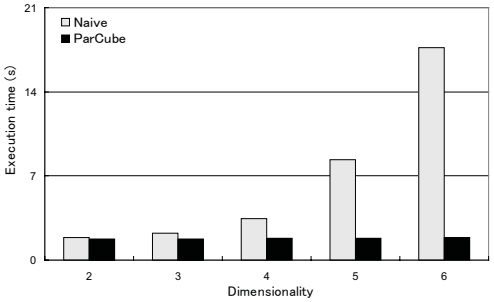
Fig. 11(a) and 11 (b) show the skyline query time against number of points in the datasets and dimensionality, respectively. We can see that the $ParCube$ algorithm outperforms the $SUBSKY$ in both cases by up to an order of magnitude. This is because the $SUBSKY$ algorithm needs to traverse the tree data structure (i.e., B-tree) to extract the skyline on the fly. On contrary, $ParCube$ pre-computes and stores the skyline points into partial order data structure, which can be easily extracted out because they exist in the first layer of DAG graph (no other points dominate them). Moreover, from the figures we can know that dimensionality has more effect on query performance compared with the number of points in the datasets.

6.3 Dominant Relationship Query Performance

To test the effect of the General Dominant Relationship query ($GDRQ$), we randomly selected 10 different points based on the synthetic dataset. Fig. 12 (a) and (b) show the query time against number of points in the datasets and di-

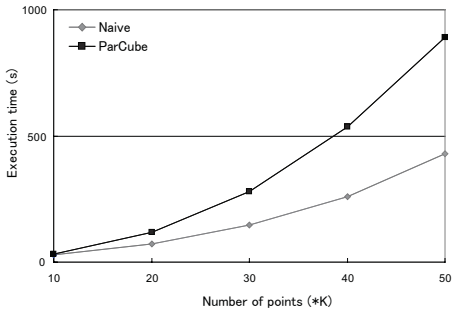


(a) General dominant relationship query against number of points(dimensionality=3)

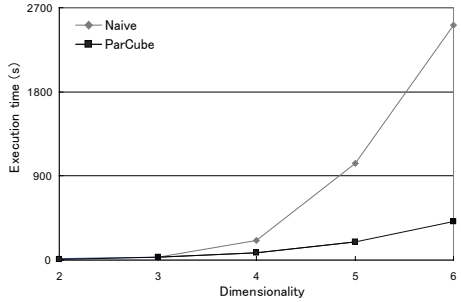


(b) General dominant relationship query against dimensionality(point number=10K)

Fig. 12 Execution time comparison between $Naive$ and $ParCube$ on general dominant relationship query



(a) Index build execution time against number of points(dimensionality=3)



(b) Index build execution time against dimensionality(number of points=10K)

Fig. 13 Execution time comparison on index building between $Naive$ and $ParCube$

mensionality, respectively. We can see that the $ParCube$ approach is better than the $Naive$ strategy. The performance of $Naive$ becomes worse as number of points or dimensionality is larger, while $ParCube$ remains almost the same. The reason is similar to that explained in Section 6.2. $Naive$ needs to traverse the index data structure (i.e., D*-tree) to compare and extract all the required points. In contrast, $ParCube$ only traverse the DAG graph to direct extract every node it passed and no comparison is necessary.

6.4 Index Data Structure Construction Performance

The efficiency of $ParCube$ is rooted in the compressed data structure it discoveries, partial order data cube ($ParCube$). In this section, we show the construction time for $ParCube$ compared with cost of building other index data structure (i.e., D*-tree) in the $Naive$ algorithm. Fig. 13 (a) and (b) show the execution time for index building against number of points in the datasets and dimensionality, respectively. We can see that the $ParCube$ is sensitive to the number of points in the datasets, that when the number gets larger, the performance of $ParCube$ construction is worse than that of D*-tree building. However, as illustrated in Fig. 13 (b), D*-tree construction becomes worse as dimensionality grows, which means that D*-tree index building is more sensitive to the dimensionality compared with $ParCube$ index building. The reason why the performance of $ParCube$ construction is good, because in high dimensional space, the probability of one point dominates another one, is very low. Hence, the sequential pattern is very few in high dimensional space and the mining process can terminate quickly.

6.5 Effectiveness of Compression

In this experiment, we explored the compression benefits of $ParCube$ compared with $Naive$ method.

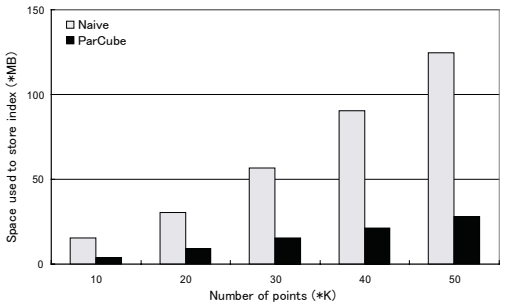
Fig. 14 (a) and (b) show the compression effect on building the data cube by partial order representation ($ParCube$), compared with D*-tree. They illustrate that using the compressed data format, DAG, is very efficient on space usage. Similar to query performance, dimensionality has more effect on the compression factor compared with the number of points in the datasets.

7. Conclusions

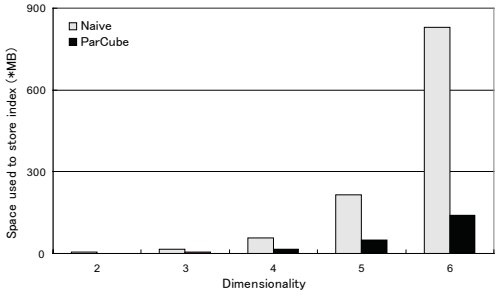
In this paper, we have introduced General Dominant Relationship Analysis, which could not be easily solved by existing strategies. Due to the interrelated connection between the partial order and the dominant relationship, we have proposed a new data structure called $ParCube$, which concisely represents the complete information of the general dominant relationship based on the partial order analysis. We have introduced efficient strategies to construct $ParCube$. The experimental results and performance study confirmed the efficiency and effectiveness of our strategies. In the future, we will investigate how to further improve the efficiency while querying the general dominant relationship.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE'95: Proceedings of the 11th International Conference on Data Engineering*, pp. 3-14, 1995.



(a) ParCube compression effect against point number (dimensionality=3)



(b) ParCube compression effect against dimensionality (point number=10K)

Fig. 14 Compression effect of ParCube against dimensionality and number of points in datasets

[2] W.T. Balke, U. Guentzer and J.X. Zheng. Efficient distributed skylining for web information systems. In *EDBT'04: Proceedings of the 9th International Conference on Extending Database Technology*, pp. 256-273, 2004.

[3] S. Borzsonyi, D. Kossmann and K. Stocker. The skyline operator. In *ICDE*, pp. 421-430, 2001.

[4] G. Casas-Garrica. Summarizing sequential data with closed partial orders. In *SDM'05: Proceedings of the 5th SIAM International Conference on Data Mining*, pp. 380-391, 2005.

[5] C.Y. Chan, H.V. Jagadish, K.L. Tan, A.K.H. Tung and Z. Zhang. On high dimensional skylines. In *EDBT'06: Proceedings of the 11th International Conference on Extending Database Technology*, pp. 478-495, 2006.

[6] C.Y. Chan, H.V. Jagadish, K.L. Tan, A.K.H. Tung and Z. Zhang. Finding k-Dominant skylines in high dimensional space. In *SIGMOD'06: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 503-514, 2006.

[7] J. Chomicki, P. Godfrey, J. Gryz and D. Liang. Skyline with presorting. In *ICDE'03: Proceedings of the 19th International Conference on Data Engineering*, pp. 717-719, 2003.

[8] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD'84: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 47-57, 1984.

[9] V. Hristidis, N. Koudas and Y. Papakonstantinou. PREFER: A system for the efficient execution of multiparametric ranked queries. In *SIGMOD'01: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 259-270, 2001.

[10] Z. Huang, C.S. Jensen, H. Lu and B.C. Ooi. Skyline queries against mobile lightweight devices in MANETs. In *ICDE'06: Proceedings of the 22th International Conference on Data Engineering*, pp. 66, 2006.

[11] W. Kieβling. Foundations of preferences in database systems. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 311-322, 2002.

[12] V. Koltun, and C.H. Papadimitriou. Approximatedly dominating representatives. In *ICDT'05: Proceedings of the 10th International Conference on Database Theory*, pp. 204-214, 2005.

[13] D. Kossmann, F. Ramsak and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 275-286, 2002.

[14] H.T. Kung, F. Luccio and F.P. Preparata. On finding the maxima of a set of vectors. In *JACM: Journal of the ACM*, 22(4), pp. 469-476, 1975.

[15] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM: Communications of the ACM*, 21(7), pp. 558-564, 1978.

[16] C. Lee, C. Grasso and M.F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3), pp. 452-464, 2002.

[17] C. Li, B.C. Ooi, A.K.H. Tung and S. Wang. DADA: A data cube for dominant relationship analysis. In *SIGMOD'06: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 659-670, 2006.

[18] X. Lin, Y. Yuan, W. Wang and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE'05: Proceedings of the 21th International Conference on Data Engineering*, pp. 502-513, 2005.

[19] H. Mannila and C. Meek. Global partial orders from sequential data. In *KDD'00: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 161-168, 2000.

[20] K. Mouratidis, S. Bakiras and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD'06: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 635-646, 2006.

[21] D. Papadias, Y. Tao, G. Fu and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD'03: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 467-478, 2003.

[22] J. Pei, J. Han, B. Mortazavi-Asl and H. Pinto. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE'01: Proceedings of the 16th International Conference on Data Engineering*, pp. 215-224, 2001.

[23] F. Preparata and M.I. Shamos. Computational geometry: In introduction. *Springer-Verlag*, 1985.

[24] S. Smith and J.D. Tygar. Security and privacy for partial order time. In *ISCA International Conference on Parallel and Distributed Computing Systems*, pp. 70-79, 1994.

[25] R. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM transactions on Computer Systems*, 3: pp. 204-226, 1985.

[26] Y. Tao, X. Xiao and J. Pei. SUBSKY: Efficient computation of skylines in subspaces. In *ICDE'06: Proceedings of the 22th International Conference on Data Engineering*, pp. 65, 2006.

[27] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. In *TKDE: IEEE Transactions on Knowledge & Data Engineering*, 18(3): pp. 377-391, 2006.

[28] T. Xia and D. Zhang. Refreshing the Sky: The compressed skycube with efficient support for frequent updates. In *SIGMOD'05: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 491-502, 2005.

[29] P. Wu, C. Zhang, Y. Feng, B.Y. Zhao, D. Agrawal and A.E. Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT'06: Proceedings of the 11th International Conference on Extending Database Technology*, pp. 112-130, 2006.

[30] X. Yan, J. Han and R. Afshar. CloSpan: mining closed sequential patterns in large datasets. In *SDM'03: Proceedings of the 3rd SIAM International Conference on Data Mining*, pp. 166-177, 2003.

[31] Y. Yuan, X. Lin, Q. Liu, W. Wang, J.X. Yu and Q. Zhang. Efficient computation of the skyline cube. In *VLDB'05: Proceedings of the 31th International Conference on Very Large Data Bases*, pp. 241-252, 2005.