On Efficient Processing of Subspace Skyline Queries on High Dimensional Data

Wen Jin[2] Anthony K. H. Tung[1] Martin Ester[2] Jiawei Han[3]

[1]School of Computing
Natl. Univ. of Singapore
atung@comp.nus.edu.sg

[2]School of Computing
Simon Fraser Univ.
{wjin, ester}@cs.sfu.ca

[3]Department of Computer Science
Univ. of Illinois at Urbana-Champagne
hanj@cs.uiuc.edu

Abstract

Recent studies on efficiently answering subspace skyline queries can be separated into two approaches. The first focused on pre-materializing a set of skylines points in various subspaces while the second focus on dynamically answering the queries by using a set of anchors to prune off skyline points through spatial reasoning. Despite effort to compress the pre-materialized subspace skylines through removal of redundancy, the storage space for the first approach remain exponential in the number of dimensions. The query time for the second approach on the other hand also grow substantially for data with higher dimensionality where the pruning power of anchors become much weaker.

In this paper, we propose methods for answering subspace skyline query on high dimensional data such that both pre-materialization storage and query time can be moderated. We propose novel notions of maximal partial-dominating space, maximal partial-dominated space and the maximal equality space *between pairs of skyline objects in the full space and use these concepts as the foundation for answering subspace skyline queries for high dimensional data. Query processing involves mostly simple pruning operations while skyline computation is done only on a small subset of candidate skyline points in the subspace. We also develop a random sampling method to compute the subspace skyline in an on-line fashion. Extensive experiments have been conducted and demonstrated the efficiency and effectiveness of our methods.*

1 Introduction

The skyline operator, which returns a set of tuples not dominated by any other tuples, has widely been applied in preference queries in relational databases. For example, given a set of hotels with the attributes of price (*price*) and distance to the beach (*distance*). Hotel A *dominates* hotel B if $A.price \leq B.price$, $A.distance \leq B.distance$, and strictly $A.price < B.price$ or $A.distance < B.distance$. The most interesting hotels are called *skyline* hotels which are not better by any other hotels in *price* and *distance*. Many algorithms [4, 13, 15, 17, 18] have been developed to improve the efficiency of answering skyline queries in large databases.

Among these earlier work of skyline queries, it is always assumed that all the attributes are involved in the skyline queries, that is, the dominating relationship is evaluated based on every attribute of the database. However, in real world applications, different users may have specific interests in different subsets of attributes. For example, a web-based hotel information system often provides 10 attributes of the hotel databases for customers to specify their desired criteria in skyline queries such as $price\ of\ room$, $price\ of\ restaurant$, $hotel\ rank$, $chef\ level$, $size\ of\ room$, $distance\ to\ beach$, $distance\ to\ downtown$, $distance\ to\ airport$, $amount\ of\ tips$, $internet\ rate$ etc. Some users may be only interested in $price\ of\ room$ and $distance\ to\ beach$, while some may prefer to $price\ of\ room$, $distance\ to\ airport$ and $chef\ level$.

Thus skyline queries are often performed in an arbitrarily subspace according to users' preferences. We will refer to this type of query as **Subspace Skyline Query(SS-query)**:

	A	B	C	E
p_1	**4**	3	2	*2*
p_2	5	**1**	**1**	*2*
p_3	1	4	4	1
p_4	3	5	5	1
p_5	2	2	3	1

Table 1. A dataset

Subspace	Skyline	Subspace	Skyline
$\emptyset$	$\emptyset$	AC	$\{p_1, p_2, p_3, p_5\}$
A	$\{p_3\}$	AE	$\{p_3\}$
B	$\{p_2\}$	CE	$\{p_2, p_5\}$
C	$\{p_2\}$	ABC	$\{p_1, p_2, p_3, p_5\}$
E	$\{p_3,p_4,p_5\}$	BCE	$\{p_2, p_5\}$
AB	$\{p_2, p_3, p_5\}$	ABE	$\{p_2, p_2, p_5\}$
BC	$\{p_2\}$	ACE	$\{p_1, p_2, p_3, p_5\}$
BE	$\{p_2, p_5\}$	ABCE	$\{p_1, p_2, p_3, p_5\}$

Table 2. Subspace skyline objects

Example 1 For a dataset T_1 of objects in Table 1, if a query subspace is AB, the answer to *SS-query* with respect to subspace AB is $\{p_2, p_3, p_5\}$, and the answer to *SS-query* to all the subspaces of AB is $\{p_2, p_3, p_5\}$, $\{p_2\}$,$\{p_3\}$ as shown in Table 2.

Only a few of recent work involves *SS-query* and they can be separated into two approaches. The first [16, 19, 21] focused on pre-materializing a set of skylines points in various subspaces while second focus on dynamically answering the queries by using a set of anchors to prune off skyline points through spatial reasoning [20]. Despite effort to compress the pre-materialized subspace skylines through removal of redundancy, the storage space for the first approach remain exponential in the number of dimensions. The query time for the second approach on the other hand also grow substantially for data with higher dimensionality where most spatial index structures become ineffective.

In this paper, our goal is to develop an appropriate structure and efficient algorithms for answering *SS-query* for high dimensional database. As the number of skyline points in the full space can be large and many subspaces can contain non-redundant skyline points, balancing the tradeoff between storage for pre-materialized subspace skyline and query answering efficiency become an important challenge.

Our solution in this paper is motivated by the observation from pairwise objects comparisons. For simplicity, consider a dataset T_2 which consists of **only** object p_1, p_2 in Table 1, assuming "smaller is better", those dimensions where $p_1(p_2)$ is "better" than $p_2(p_1)$ or "equal" are shown in Table 3 respectively. It is very easy to infer the answers of *SS-query* from here. For example, $\{p_2\}$ is the skyline in subspace B or C since p_2 is "better" in dimension B, C; $\{p_1, p_2\}$ are the skylines in AB, AC and ABC since p_1 or p_2 contributes at least one "better" dimension in the corresponding subspace.

Dimensions where p_1 is "better"	{A}
Dimensions where p_2 is "better"	{B,C}
Dimensions where p_1 equals to p_2	{E}

Table 3. Value comparisons between p_1 and p_2

Extending this for dataset of more objects, we can compute the "better" or "equal" subspaces for each pair of objects, and group together pairs that share all these subspaces, into an index structure to efficiently facilitate the answering of *SS-query*. Accordingly, we refer to these subspaces as *dominating space, dominated space or equivalent space*.

Our contributions in this paper are as follows:

- We propose a framework for processing *SS-query* which balance the extremes of high pre-materialization storage cost or low query efficiency.
- We propose the notions of the *maximal partial-dominating space, maximal partial-dominated space* and *maximal equality space* based on the pairwise objects, and prove their equivalence to the corresponding spaces based on pairwise full skyline objects.
- We build the maximal space index, *MS-index* on the above spaces and develop efficient access methods for *SS-query* in order to achieve a good trade-off between the materialized space consumption and the query processing cost.
- We conduct comprehensive experiments and demonstrate our methods work well on datasets with different distributions.

The rest of the paper is organized as follows. Section 2 presents the notions and properties of maximal partial dominating, dominated and equality space on pairwise full space skyline objects and proposes an efficient query method. Section 3 presents a filtering-based scheme on subspace skyline query. In Section 4, a method based on maximal space and filtering-based scheme is proposed to answer dominating subspace query. We present our experimental results and related work in section 5 and Section 6. We conclude the paper in section 7.

2 Skyline Materialization

In this section, we will present an indexing scheme to support the answering of *SS-query*. As we have discussed in the introduction, the ideal approach should achieve a good trade-off between the space required and the query answering time. Clearly, materializing the skyline for all subspace will be expensive in both these aspects especially for high dimensional datasets in which many of the points are skylines. To introduce our approach, let us denote a set of objects X in an n-dimensional space $D = (D_1, \ldots, D_n)$, where dimensions $D_1, \ldots, D_n$ are in the domain of numbers.

Definition 1. (Dominating Relationship) *Given X, object $p \in X$* dominates *another object $q \in X$, denoted as $p \succeq q$, if $p.D_i \leq q.D_i$ for $(1 \leq i \leq n)$ and at least for one dimension D_{i_0} $(1 \leq i \leq n)$, $p.D_{i_0} < q.D_{i_0}$. Correspondingly, we can also say that q is a dominated object.*

The definition above assume without loss of generality that all the attributes are best minimized. In particular, a strict dominating relationship which excludes equality relationship is useful.

Definition 2. (Strictly Dominating Relationship) *Given X, object $p \in X$* strictly dominates *another object $q \in X$, denoted as $p \succ q$, if $p.D_i < q.D_i$ for $(1 \leq i \leq n)$. Here, q is a strictly dominated object.*

Definition 3. (Skyline) *Object $p \in X$ is a skyline object if p is not dominated by any other objects in X.*

Definition 4. (Subspace skyline) *A subset of dimensions $B \subseteq D$ forms a $|B|$-dimensional subspace of D. For an object u in space D, the projection of u in subspace B, denoted by u_B, is a $|B|$-tuple $(u.D_{i_1}, \ldots, u.D_{i_{|B|}})$, where D_{i_1}, ..., $D_{i_{|B|}} \in B$ and $i_1 < \ldots < i_{|B|}$.*

The projection of an object $u (u \in S)$ in subspace $B \subseteq D$ is in the subspace skyline *(of B) if u_B is not dominated by any object w_B in B for any other object $w \in S$. We call u a* subspace skyline object *(of B)*

Definition 5. (Partial-dominating Relationship) *A set of skyline objects $S \subseteq X$ is given in full space $D = (D_1, \ldots, D_n)$, for any object $p, q \in S$, p is* partial-dominating *q if under dimensions $D' \subset D$, $p.D_{i_j} \succ q.D_{i_j}$ for $D_{i_j} \in D'$. On the other hand, p is* partial-dominated

by q if under $D'' \subset D$, $q.D_{i_k} \succ p.D_{i_k}$ for $D_{i_k} \in D''$, $D' \cap D'' = \emptyset$.

Here D' is called p's *partial dominating space*, while D'' is p's *partial-dominated space* (or correspondingly q's *partial dominating space*). For example, B or C is a partial dominating(dominated) space for $p_2(p_1)$ in Table 3. We say a partial-dominating space D' is maximized if there exists no other space $D^* \supset D'$ such that D^* is a partial-dominating space. In this case, D' is called a *maximal partial-dominating space* for p and q, adding any other dimension to D' which turns into D'^* would violate the dominance of $p \succ q$ in D'^*. For example, BC is the *maximal partial-dominating space* for p_2 w.r.t p_3, but in ABC and BCE, p_2 and p_3 cannot dominate each other. Besides, there could exist duplicate values in some subspaces, for example, $p_3.E = p_4.E = p_5.E = 1$ equals each other in E dimension in Table 1.

Definition 6. (Equality Relationship) *Given a set of skyline objects S in full space $D = (D_1, \ldots, D_n)$, for any skyline object $p \in D$ and any object q(could be a non skyline object), p is* equal *to q w.r.t. D''' or if under dimensions $D''' \subset D$, $p.D_{i_l} = q.D_{i_l}$ for $D_{i_l} \in D'''$.*

Similar to previous definitions, D''' is called *equality space* of p.

Lemma 2.1. *Given a set of skyline objects S in full space $D = (D_1, \ldots, D_n)$, for any pair of objects $p, q \in S$, the* partial-dominating space D', partial-dominated space D'' *and the* equality space D''' *of p are all maximized if and only if $D' \bigcup D'' \bigcup D''' = D$.*

Since any skyline object p in the full space is not dominated by another skyline object, so the *maximal partial-dominating space* for p is not empty. Given a set of skyline objects S in full space $(D_1, \ldots, D_n)$, we can maintain the *maximal partial dominating space, maximal partial dominated space* and *maximal equality space* for any pair of objects $p, q \in S$. As the same *maximal partial dominating space, maximal partial dominated space* and *maximal equality space* may correspond to multiple pairs of skyline objects, these pairs of skyline objects can be grouped together. Each *m*aximal *s*pace can be indexed in an index called *MS-index*.

Definition 7. (MS-index) *Each entry of* MS-index *is a triple $(t(D'), t(D''), t(D'''))$ where D' is the* maximal partial dominating space, *D'', the* maximal partial dominated space, *and D''', the* maximal equality space *and $(t(D'), t(D''), t(D'''))$ denote the corresponding pair of tuples.*

Example 2 For the 4-dimensional dataset shown in the Table 1, where p_1, p_2, p_3, p_4, p_5 are five objects. The skyline objects in each subspace are listed in Table 2, where p_1, p_2, p_3, p_5 are skyline in the full space $\{A, B, C, E\}$. For all pairs of full space skyline objects, the maximal partial-dominating spaces, the maximal partial-dominated spaces and the maximal equality spaces are listed as a *MS-index* in Table 4 where there are 3 entries. For instance, in the entry (1), among ($\mathbf{p_1}$, p_3), ($\mathbf{p_2}$, p_3), and ($\mathbf{p_2}$, p_5), BC are the dimensions for $\mathbf{p_1}, \mathbf{p_2}$ to be dominating (darkened letters in the maximal partial-dominating space column), while AE are the dimensions for p_3, p_5 to be dominating (shown as darken letters in the maximal partial-dominated space).

Interestingly, we can derive the same property of subspace skyline in [16] from the concept of maximal spaces.

Theorem 2.1. *Any subspace skyline object must be either a full space skyline object or an object in* maximal equality space *which share the same values in this subspace with a full space skyline object[16].*

From the *MS-index* in Table 4, we note that for each entry, the maximal partial dominating space and maximal partial dominated space always contain the same pair of objects if both space are not empty. For example, in the first entry $D' = BC \neq \emptyset$ and $D'' = AE \neq \emptyset$, both have the same pair of objects (p_1, p_3), (p_2, p_3) and (p_2, p_5). Similar cases occur in the second and third entry, where both space are not empty and have the same pair of objects (p_1, p_5) and (p_1, p_2), (p_3, p_5).

Lemma 2.2. *Given an MS-index, for any entry $E(t(D'), t(D''), t(D'''))$, if $D' \neq \emptyset$ and $D'' \neq \emptyset$, then $t(D') = t(D'')$.*

Lemma 2.2 show the fact that the the pair of full space skyline objects in the *maximal partial dominating space* and the *maximal partial domianted space* are symmetric. As such, we can reduce the storage cost for *MS-index* by only maintaining pairwise skyline objects in one of these two spaces.

Definition 8. (Entry Invalided and Entry Acceptable) *Given a skyline object p in the full space D, a query subspace Q_s, and an entry $(t(D'), t(D''), t(D'''))$ in the MS-index. We say p is entry invalided by $(t(D'), t(D''), t(D'''))$ if p is found to be dominated by some object in Q_s exclusively based on D', D'' or D'''. Otherwise p is entry acceptable for $(t(D'), t(D''), t(D'''))$.*

Intuitively, p is entry invalided by entry $(t(D')$, $t(D'')$, $t(D'''))$ if p can be pruned off as a potential skyline object in the query space Q_s using the entry. For example, given query subspace A, for the first entry in the *MS-index*, since $A \subset AE$ which is a maximal partial dominated space for p_1 and p_2 respectively, so p_1 and p_2 cannot become skyline in A. Generally, we have the following lemma.

Lemma 2.3. *Given a skyline object p in the full space D, a query subspace Q_s, and an entry $(t(D'), t(D''), t(D'''))$ in the MS-index. p is entry invalidated by $(t(D'), t(D''), t(D'''))$ only if: (1) $Q_s \cap D' = \emptyset$, $Q_s \cap D'' \neq \emptyset$, or; (2) $Q_s \cap D'' = \emptyset$, $Q_s \cap D' \neq \emptyset$, or;*

Lemma 2.3 is very useful for answering *SS-query*. That is, when checking each entry of *MS-index*, if any maximal space (dominated or domination) in the entry is a superset of Q_s, then there must exist at least one full space skyline point in the entry which **cannot** become a skyline in Q_s.

Lemma 2.4. *Given any skyline object p in the full space D, a query subspace Q_s, and an entry $(t(D'), t(D''), t(D'''))$ in the MS-index. Entry $(t(D'), t(D''), t(D'''))$ cannot exclude p as a candidate skyline object in Q_s if (1) $Q_s \cap D' \neq \emptyset$, $Q_s \cap D'' \neq \emptyset$ or (2) $Q_s \subseteq D'''$.*

	Maximal-partial dominating space		Maximal-partial dominated space		Maximal equality space	
1	{B,C}	$\mathbf{p_1} < p_3$	{A,E}	$\mathbf{p_3} < p_1$		
		$\mathbf{p_2} < p_3$		$\mathbf{p_3} < p_2$		
		$\mathbf{p_2} < p_5$		$\mathbf{p_5} < p_2$		
2	{C}	$\mathbf{p_1} < p_5$	{A,B,E}	$\mathbf{p_5} < p_1$		
3	{A}	$\mathbf{p_1} < p_2$	{B,C}	$\mathbf{p_2} < p_1$	{E}	$p_2 = p_1$
		$\mathbf{p_3} < p_5$		$\mathbf{p_5} < p_3$		$p_3 = p_5 = \mathbf{p_4}$

Table 4. Maximal space index(MS-index)

Lemma 2.4 illustrates a very important property which can be usedd in answering *SS-query*. That is, given an entry in the *MS-index*, if both D' and D'' in the entry have at least one common dimension with Q_s, then the detailed information in the entry can play no part in pruning off any potential skyline candidate from the subspace Q_s. The intuition here is that all pairs of skyline points in this entry will be incomparable in Q_s since they would dominate each other in at least one dimension of Q_s. For the same reasoning, if Q_s is a subset of D''', then all pairs of objects in the entry are also not comparable in Q_s and thus need not be accessed.

Such an approach will help to save much cost in comparisons since we do not have to access the pairs of objects in entry. For example, if $Q_s = BE$ and given the first entry in Table 4, since BC match Q_s on B while AE match Q_s on E, the pairs of skyline objects in the first entry cannot dominate each other and no potential skyline points in Q_s can be pruned off by this entry.

While Lemma 2.3 and Lemma 2.4 help to define whether a single entry is useful for pruning off potential skyline points in a particular query space Q_s, we still need to consider multiple entries when computing the skyline points in a subspace. Next, we will show that if there exists a potential skyline point is entry invalidated with one of the entries, then it can never be a skyline point in the query subspace. On the other hand, it is still not guarantee to be a skyline point in Q_s even if it is entry acceptable with one of the entry. For example, if the query subspace is BE, then the first entry in Table 4 has no effect on the full space skyline objects, say p_1. While for the second entry, p_1 is dominated in ABE so p_1 is finally removed.

Theorem 2.2. *Given the MS-index, for any query subspace Q_s, we have the following properties:*

1. *All full space skyline objects which are entry acceptable w.r.t. to Q_s for all the entries in the MS-index, are skyline objects in Q_s.*
2. *All full space skyline objects which are entry invalidated w.r.t. to Q_s for some entries of the MS-index, are not skyline objects in Q_s.*
3. *Let S be the set of skyline in subspace Q_s based on (1) and (2) above. All objects which are equal in values for all dimensions in Q_s to some objects in S will also be skyline objects in subspace Q_s.*

3 Query Processing Based on the MS-Index

We next look at the actual processing of a *SS-query* using the MS-Index.

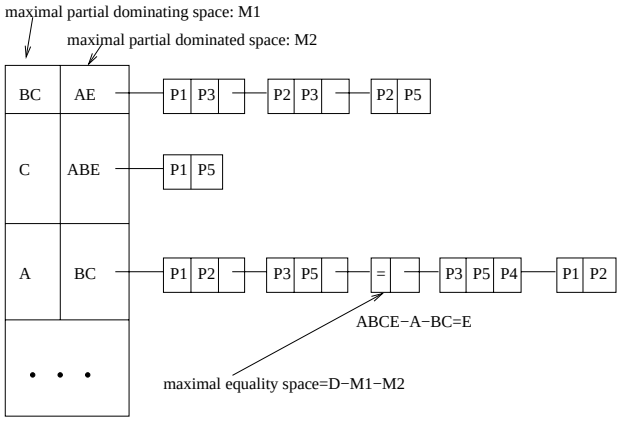


Figure 1. MS-index Structure

3.1 MS-index Structure

As presented in Section 3, the *MS-index* is built by materializing the maximal partial dominating space, the maximal partial dominated space and the maximal equality space of pairs of full space skyline objects. Nevertheless, the *MS-index* depicted in Table 4 needs to be further refined since unlike the standard relational table, multiple pairs of objects are in each entry.

To address these issues, we modify the index structure as shown in Figure 1. We use an array of list to maintain the maximal partial dominating spaces and the maximal partial dominated spaces of different pairs of full space skyline objects. Each entry links to its full space skyline objects which are stored pairwise and linked node by node.

As a rule, given a pair of objects (a, b) in an entry, the first object a is always better than the second object b in the maximal partial dominating space, and b is better than a in the maximal partial dominated space. For example, $p_1 \succ p_3$ in BC while $p_3 \succ p_2$ in AE. The equality space can be inferred by the complementary set of the maximal partial dominating set and the maximal partial dominated set. For example in the 3rd entry of *MS-index*, the maximal equality space E is the complementary space of A and BC. If equality space exists, we can represent it after the nodes of the other two spaces, by inserting a node with a label "=" (is linked by (p_3, p_5) in Figure 1), and links to the corresponding full space skyline objects in the equality space with *increasing values* (starting from (p_3, p_4, p_5)) then (p_1, p_2), i.e. $p_3 = p_4 = p_5 < p_1 = p_2$).

To efficiently maintain the entries in the *MS-index*, some

recent work for indexing transaction data [1, 11] can also be applied to indexing the maximal space of each of pair of full space skyline objects.

3.2 Answering SS-query

With the support of the *MS-index*, we start with the set skyline objects in the full space, and simply check the entries in the *MS-index*. According to Theorem 2.2, those skyline objects which satisfy condition (1) in the theorem are kept while skyline objects which satisfy condition (2) in the theorem are pruned. Finally, those objects that satisfy (3) are added.

We first give two concrete examples to illustrate the main idea of our algorithm for answering *SS-query*.

Example 3 Suppose the query space $Q_s = \{BCE\}$. We access the first entry and find that both maximal partial-dominating space ($\{BC\} \cap \{BCE\} = \{BC\}$, and maximal partial-dominated space ($\{BCE\} \cap \{AE\} = \{E\}$ have non-empty intersection with Q_s. Based on Lemma 2.4, we know they do not affect the potential candidate subspace skyline point. The result is the same for the second entry in which both maximal partial-dominating space ($\{C\} \cap \{BCE\} = \{E\}$) and maximal partial-dominated space ($\{BCE\} \cap \{ABE\} = \{BE\}$ have non-empty intersection with Q_s. Now, consider the third entry. Only the maximal partial-dominated space ($\{BC\} \subset \{BCE\}$, and maximal equality space $\{E\} \subset \{BCE\}$). Since p_1, p_3 are dominated by p_2, p_5 in $\{BC\}$, and equal with p_2, p_5 in $\{E\}$, so p_1, p_3 are pruned, and the subspace skyline objects in $\{BCE\}$ are p_2, p_5.

Sometimes, the subspace skyline objects may not be a member of the full space skyline objects, but from those objects which satisfy (3) in Theorem 2.2.

Example 4 Suppose the query space $Q_s = \{E\}$. From both the first entry and the second entry, based on Lemma 2.3, the maximal partial-dominated spaces ($\{AE\}$ and $\{ABE\}$ are both supersets of E which will pruned off p_1 since it is dominated in E. Now the candidates are p_2, p_3, p_5. Among the third entry, only the maximal equality space $\{E\}$) needs to access. Since $p_1 = p_2$ in E, so p_2 is pruned off too. On the other hand, since $p_3 = p_5 = p_4$ in E, P_4 will be a skyline object in E.

The pseudo-code of *SS-query* method is as shown as Algorithm 1.

The algorithm sequentially scans the index once, and this is done efficiently by (1) performing a low-cost "must do" pruning based on Lemma 2.3 and (2) saving a non-trivial cost of evaluating detailed pairwise skyline objects according to Lemma 2.4. Although the worst case of space cost is $O(m^2)$ where m is the number of full space skyline objects, it does not need frequent I/Os described previously and thus leads to a rather good performance. Furthermore, when the dimensionality is high, we propose several methods in next section in order to achieve a good trade-off between query processing time and index space.

Algorithm 1 MS-Index Lookup Method

Input: A MS-index build on full space skyline S, query subspace Q_s
Output: Subspace Skyline in Q_s
Method:

1. $i = 1$;
2. WHILE NOT EOF *MS-index* DO
3. $E_i = (t(D'), t(D''), t(D'''))$;
4. IF $\{Q_s \cap D' = \emptyset, Q_s \cap D'' = Q_s\}$ OR $\{Q_s \cap D'' = \emptyset, Q_s \cap D' = Q_s\}$ THEN
5. Prune objects which are "bad" in $t(D')$ or $t(D'')$;
6. ELSE IF $\{Q_s \cap D' \neq \emptyset\}$ AND $\{Q_s \cap D'' \neq \emptyset\}$ THEN
7. i=i+1;
8. ELSE IF $\{Q_s \cap D''' \neq \emptyset, Q_s \cap D' \neq \emptyset\}$ OR $\{Q_s \cap D''' \neq \emptyset, Q_s \cap D'' \neq \emptyset\}$ THEN
9. Prune objects which are "bad" in $t(D')$ or $t(D'')$
10. Add non skyline objects in $t(D''')$ with the same values as objects that are "good" in $t(D')$ or $t(D'')$;
11. i=i+1;
12. Output objects S as subspace skyline objects in Q_s;

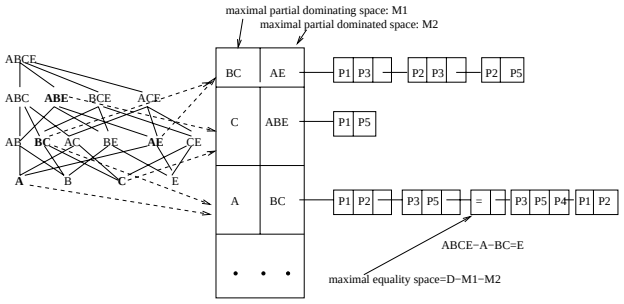


Figure 2. Hierarchical Navigation in MS-index

3.3 Employing Hierarchical Navigation

The proposed Algorithm 1 for *SS-query* needs a scan of the index. To reduce the number of entries being accessed, we introduce a hierarchical lattice structure in order to facilitate the search. Figure 2 shows the subspaces of $ABCE$ in a lattice where the nodes corresponding to the *maximal partial dominating space, maximal partial dominated space* (in bold), point to the entries with the same subspaces (seen the nodes with dashed arrows). In this case, when the query subspace Q_s is given, we can avoid accessing those entries which are entry acceptable. Basically, we only need to check the entries pointed by Q_s and those pointed by Q_s's superset if they have links to the entries. Note that in the whole lattice, only nodes ABE, BC, AE and C have links to the entries in the index. For example, if $Q_s = BC$, it has two links to the 1st and the 3rd entry respectively, while none of its superset (BCE,ABC) has link to the 2nd entry or any other entry. So it only access two entries instead of three. On average, it can reduce the number of entry accesses to a large extent. Based on this hierarchical navigation, the algorithms in subsection 3.1 and 3.2 would substantially improve the performance.

4 Filtering based on Samples

The previous method mainly focuses on materializing the *maximal partial dominating spaces,maximal partial dominated spaces* and *maximal equality spaces* between pairs of full space skyline objects into a *MS-index*, and using it to answer *SS-query* However, as the materialized size is determine by the number of skyline objects in the full space, the size of the MS-index can grow substantially at high dimensionality where the number of full space skyline objects increase. To overcome this drawback, in this section we propose several methods which make use of a sample of the full space skyline points to filter off most of the non-skyline points in the queried subspace Q_s before using any state-of-the-art skyline computation algorithm to identify the real skyline objects in Q_s. Such an approach uses only a constant size space. We will first present the basic filtering scheme before looking at a hybrid one which combine the basic filtering scheme with the maximal space approach we took in previous section.

4.1 Basic Filtering Scheme

Obviously, we wish to pick some objects with high dominating capacity in the queried subspace to prune off as many dominated objects as possible. This will reduce the number of data accesses and comparisons among objects. Motivated by the property in Theorem 2.1, we developed a sampling method to give strong pruning power with limited processing.

Sampling full space skyline objects The basic idea of the first sampling approach is as follows. We first randomly sample k objects from the full space skyline objects S as initial "seeds" in each subspace. Since any subspace skyline object is either a full space skyline object or an object in *maximal equality space* which have the same values in the subspace with a full space skyline object, the chosen "seeds" are most likely to have high dominating capacity compared with objects that are not in the skyline. After one scan of the database, each object is compared with the chosen k full space skyline objects w.r.t. the query subspace, and many non skyline objects will be pruned off. Afterwards, any existing skyline computing algorithm such as DC(divide-and-conquer) method [12] can then be applied in the remain candidates to obtain the skyline in the subspace

Several criterions are required during the sampling step: (1) we prefers to choose objects with smaller values in the dimensions of the queried subspace which will more likely to dominate more points in Q_s; (2) the k full space skyline objects are not dominated by each other in subspace Q_s. To achieve (1), we can select those full space skyline objects with small average values over all attributes. For (2), we can first obtain k samples satisfying (1), then if some of them are dominated by the some of the remaining k samples, we re-sample to replace those that are dominated until (2) is satisfied or exceed the user specified number of iterations.

Each of the k full space skyline objects (usually $k \ll |X|$), can be see as a temporary subspace skyline object in Q_s. Other full space skyline objects are added in as candidate skyline object in Q_s only if they are not dominated by these k objects in Q_s. We have the following lemma:

Lemma 4.1. *No skyline object in the subspace Q_s will be pruned off by the filtering.*

For objects which pass through the initial filtering, a skyline computation algorithm will be applied on these candidates to remove those that are not in the skyline of Q_s.

Lemma 4.2. *All false skyline candidates will be removed at the end of the skyline computation on the candidates.*

The pseudo-code for the sampling-based subspace skyline algorithm shown in Algorithm 2.

Algorithm 2 A Sampling-based Subspace Skyline Method.

Input: a set of full space (D) skyline S, query subspace Q_s
Output: subspace skyline in Q_s.
Method:

1. Sample k objects $S' \subseteq S \subseteq X$ which not dominate each other in Q_s;
2. Label 'skyline' for k objects in Q_s;
3. FOR each object $x_i \in X$ DO
4. IF any 'skyline' object $s(\in S') \succeq x_i$ THEN
5. Prune x_i from X;
6. Apply DC algorithm to X to find skyline objects SS in Q_s;
7. Output SS;

This method only need a storage cost of $O(m)$ where m is the number of skyline in full space, and compute subspace skyline on-the-fly. The runtime complexity is O(MlogM) where M is the number of objects which remain in the subspace after pruning. In practice, $M << N$.

Sampling entries in MS-index The difference between the second sampling approach and the first is that we choosing k entries from the *MS-index* and using them for pruning off any potential skyline point in a subspace skyline query. Since the maximal spaces of different pairwise full space skyline objects in *MS-index* imply "good" or "bad" situation in the corresponding subspaces, so the chosen "seeds" must contain the skyline objects in some subspaces. After comparing the dataset with the chosen k entries w.r.t. the query subspace, many non-skyline objects will be pruned off as well. The subsequent processing is similar to the first approach by applying DC (divide-and-conquer) method to the pruned dataset to obtain the subspace in the subspace as described in Algorithm 2, and here we omit the pseudo-code. Typically we can randomly sample k entries or choose k representative entries. For example, those entries whose union of maximal spaces can cover a larger dimensionality will be chosen with a high priority since the query subspace will most probably be contained in some maximal space or share some dimensions with these spaces, thus the properties in previous section can be used.

4.2 Hybrid Filtering Scheme

We have so far proposed two methods for answering *SS-query*. Basically, these two methods reflect the different requirements on materialized space and the query processing cost. In this section, we develop a method which combines

the features of previous two methods, and aim to keep a good trade-off between the materialized space and the query answering cost.

Intuitively, we wish to keep the pairs of full space skyline objects in the corresponding entries, if the number of nodes in the list is small (say, if under the a threshold c). While for those entries which contain a *large number* of skyline objects pairs(say, if above a threshold c), we sample k full space skyline objects and store them in the list as "seeds". Now, the *MS-index* has been modified into a *hybrid MS-index*. For answering *SS-query* , we also modify the approach in Section 2 to quickly retrieve subspace objects: if the queried subspace involves k samples full space skyline "seeds", the properties described in the basic filtering scheme can help to prune full space skyline objects in the queried subspace skyline or add new objects to the queried subspace skyline.

The basic idea of the hybrid materialization-filtering subspace skyline method is as follows. We randomly sample k objects from the full space skyline objects S as initial "seeds" in the subspace for entries in the MS-index which contain a large number of full skyline objects pairs, while entries which small number of skyline pairs remain unchanged.

The difference between this and the method in Section 2 is that for those subspace which maintain only the k full space skyline objects, we only need to compare them with other full space skyline objects instead of the dataset. Similarly, any existing skyline computing algorithm can be applied in the pruned full space skyline objects to identify the subspace skyline in Q_s.

The pseudo-code for the hybrid-materialization-filtering subspace skyline algorithm is shown as Algorithm 3.

Algorithm 3 A Hybrid-Materialization-Filtering Method

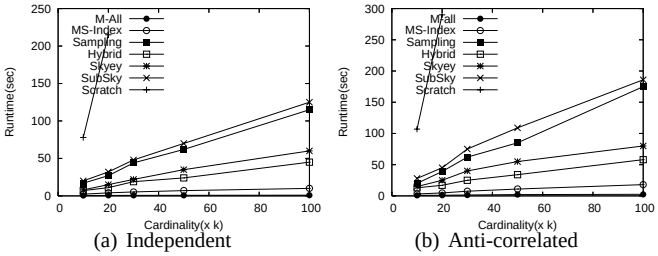
Input: A *hybrid* MS-index, a set S of full space skyline objects, query subspace Q_s
Output: Subspace Skyline in Q_s
Method:

1. $i = 1$;
2. WHILE NOT EOF *hybrid MS-index* DO
3. $E_i = (t(D_i'), t(D_i''), t(D_i'''))$;
4. IF $\{Q_s \cap D' = \emptyset$,$Q_s \cap D'' = Q_s\}$ OR $\{Q_s \cap D'' = \emptyset$, $Q_s \cap D' = Q_s\}$ THEN
5. IF D' OR D" has a small size THEN
6. Access the pairs of objects in D' OR D'';
7. ELSE apply the Sample-based Algorithm 4.1 to find skyline in D' or D'';
8. IF $p \in S$ is dominated in Q_s THEN
9. Prune p from S;
10. ELSE IF $\{Q_s \cap D''' = Q_s$, the objects which $p \in S$ shares the same values in Q_s are not skyline in $Q_s\}$ THEN
11. Prune p from S;
12. ELSE IF $\{Q_s \cap D''' = Q_s$, the objects SS which $p \in S$ shares the same values in Q_s are subspace skyline in $Q_s\}$ THEN
13. Add $s \in SS$ to S;
14. i=i+1;
15. Output objects S as subspace skyline objects in Q_s;

The storage cost of hybrid method is $O(|E_L| \cdot k + |E_s| \cdot c)$ where $|E_L|$ is the number of entries with a large number of pairwise full space objects, $|E_S|$ is the number of entries with a small number of pairwise full space objects, and c is the upper bound threshold of the small number of pairwise full space objects. The runtime is $O(|L| + MlogM)$ where $|L|$ is the total length of index and M is the number of unpruned objects in $|E_L|$ entries.

5 Related Work

The skyline computation originates from the maximal vector problem in computational geometry. which was proposed by Kung et al[12], and a set of algorithms were developed for variants of this problem. Basically, the skyline in computational geometry always involves in datasets which are not very large, and can only be suited to the main memory. In [4], Borzsonyi et al. introduce the skyline operator over large databases, and also propose a block-nested loops(BNL) method and a divide-and-conquer method.



(a) Independent (b) Anti-correlated

Figure 3. Time vs Size

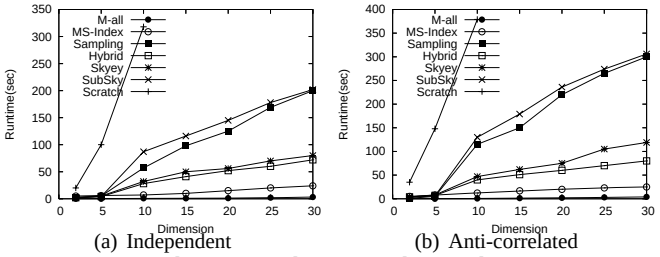
In [18], the authors proposed two progressive skyline computing methods. The first method uses bitmap to map each object to a m-bit vector, and then identify whether an object is in skyline. The second method introduces the specialized B-tree for each combination list of dimensions that a user might be interested in. Then the algorithm loads the first batch of each list, and finds the ones not dominated by any of the already-found skyline objects into the skyline list. Kossmann et al. present an online algorithm NN based on nearest neighbor search, which gives a big picture of the skyline very quickly in all situations [13]. Balke et al. [3, 2] study skyline computation in web information systems, applying the "threshold" algorithm of [8]. The sort-first-skyline (SFS) [7] sorts the input data according to a (monotone) preference function, after which the skyline can be found in another pass over the sorted list. Papadias et al. proposed a progressive algorithm to find skyline with optimal times of node accesses[17] with the help of typical R-tree indexes. Tan et.al [5] proposed layered skyline-based methods to compute skyline with partially-ordered attributes.

Recent skyline study include: [19] studies the problem of computing the skylines in all subspaces and develop efficient algorithms, and [16] explores the structure of skylines in subspaces, and uses the concepts of skyline groups and decisive subspaces to capture the semantics of subspace skylines. Both methods in [19] and [16] compute skylines for every subspace, and interestingly, both studies suggest that a top-down depth-first search framework may favor efficient computation. Lin et al. [15] consider the skyline maintenance

over data streams. Godfrey et.al. based on the complexity and performance analysis of existing skyline algorithms, proposed a scan-based skyline algorithm **LESS** which seemingly more naive, but is much better behaved in practice[9]. Zhang et. al. proposed a notion of strong skyline[22] points in high dimensional data, and also developed efficient algorithms.

6 Experiments

In this section, we conducted experiments to compare the performance of our methods using different types of datasets. We evaluated the efficiency and the scalability of the proposed methods in term of the cardinality and dimensionality of the datasets. All methods proposed in this paper were implemented using Microsoft Visual C++ V6.0, including maximal space indexing method (denote as *MS-Index*), basic filtering scheme on samples (denote as *Sampling*), and hybrid scheme (denote as *Hybrid*). For both the sampling and hybrid scheme, we set k, the number of samples or "seeds"we used for pruning to be around 5% of the full space skyline.



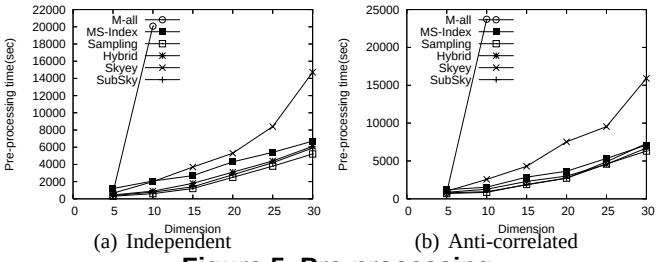
(a) Independent (b) Anti-correlated

Figure 4. Time vs Dimension

For the purpose of comparison, we also implemented the method in [16] (denote as *Skyey*) which materializes skyline groups and their signatures, another subspace skyline method *Subsky*[20][1] which is based on transforming multi dimensional data to 1D values, and indexing the dataset with a B-tree, a naive baseline method which materialize skylines in every subspace (note as *M-all* method), and a method which directly computing the subspace skyline from scratch for the chosen subspace (*Scratch*). We also obtain the algorithm for compressed skyline cube kindly given to us by the authors [21], but the algorithm was unable to run beyond a dimensionality of 7 making it impossible for us to compare again their result for high dimensional data. Due to the limitation in space, interesting readers are referred papers for details. Experiments were conducted on a PC with an Intel Pentium 4 1.6 GHz CPU, 512MB main memory and a 40 GB hard disk, running the Microsoft Windows XP Professional Edition operating system.

Using the data generator provided by the authors of [4], we generated two types of data sets as follow: (1) *Independent data sets* where the attribute values of records are uniformly distributed; (2) *Anti-correlated data sets* where if a record is good in one dimension, it is unlikely to be good in other dimensions. We suppressed experiments on a third type of distribution called correlated data as they are much more simple to process and did not bring about significant difference between various algorithms. For each type of data distribution, we generated data sets with different sizes (from $10,000$ to $100,000$ tuples) and with dimensionality varying from 10 to 30. We investigate different aspects of the methods mainly in terms of querying time, pre-processing cost and storage space.

• **Query Time Comparisons** We compare the query response time of the subspace skylines by taking average over randomly selected subspace.

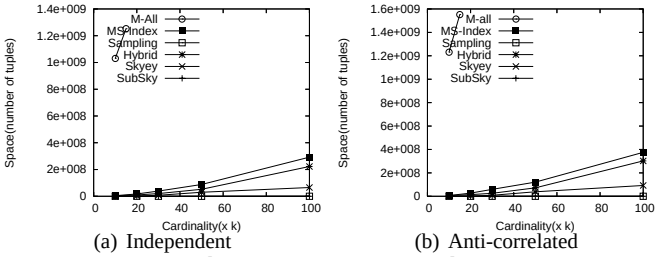


(a) Independent (b) Anti-correlated

Figure 5. Pre-processing

We first fix the dimensionality of the datasets as 20, and test the query time in varying with different cardinalities of the datasets from 10,000 to 100,000 tuples.

Figure 3(a) and Figure 3(b) show the results in the case of independent data and anti-correlated data respectively. Clearly, except for the baseline *M-all* method, our pairwise materialization method *MS-index* achieves better query response time than the remaining methods. *Hybrid* ranks third in query time and *Skyey* method is the fourth (note that *MS-index* only uses 1/3 to 1/2 the query time compared to *Skyey*). *SubSky* is slower than *Sampling* since it relies on the k-means technique and it is difficult to find meaningful clusters in high dimensions. As such, the corresponding query does not perform well for high dimensional(> 5) data. *Scratch* is the slowest among all methods.



(a) Independent (b) Anti-correlated

Figure 6. Space vs Size

Among the two data sets, the query time on the independent datasets is always faster than the computation on the anti-correlated datasets. We also evaluated the query time by varying the number of dimensions from 10 to 30, with a fixed cardinality of 100,000 tuples. Figure 4(a) and Figure 4(b) show the results for independent and anti-correlated datasets respectively. We observe similar rankings in runtime from Figure 3(a) and Figure 3(b).

In particular, our *MS-index* method is most efficient and scalable with respect to the dimensionality comparing to the baseline *M-all*. For other methods, it still achieves significant gains in query time. For example, it saves over 100 seconds over *Skyey* whenever dimensionality reaches 15.

[1] As SubSky performance is heavily dependent on the number of anchors being selected, we always vary such a number and select the result that give the best average performance

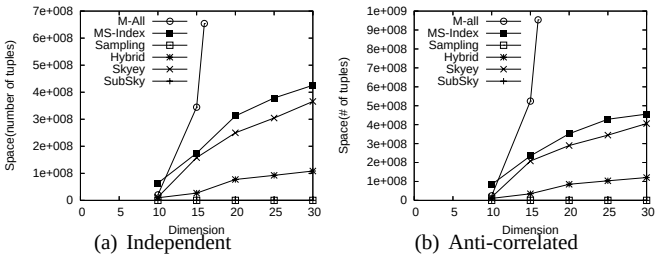
From the query point of view, all methods provide quick responses to the query when the size of the dataset is small. As the size of the data increases, the materialization of maximal space *MS-index*), the materialization of skyline group and signatures (*Skyey*) and the materialization of partial maximal space and partial sampling full space skyline objects (*Hybrid*) are still efficient.

• **Pre-processing Time** To test the efficiency of pre-processing, we compare the pre-processing time of the above five methods for both independent datasets and anti-correlated datasets.

Specifically, the components of pre-processing time for individual methods are: (1) *MS-Index* need (a) compute a set of full space skyline objects, and (b) materialize the maximal partial dominating, dominated and equality space for every pair of full space skyline objects; (2) *Sampling* method need to compute a set of full space skyline objects, and choose k full space skyline objects as "seeds" in every subspace; (3) *Hybrid* method need compute a set of full space skyline objects, and choose k full space skyline objects as "seeds" in some subspaces, and compute pairwise full skyline objects in some subspaces; (4) *Skyey* method need to compute a set of full space skyline objects, and compute skyline group and signature in every subspace; (5) *SubSky* need to find multiple anchors and transform the dataset into B-tree (6)*M-all* method need compute skyline objects in every subspace. We fix the cardinality of the datasets to 100,000 tuples, and vary the dimensionality of the datasets from 10 to 30.

Figure 5(a) and Figure 5(b) show that *Sampling* method takes the least pre-processing time since it only requires the information of full space skylines, and the time used in sampling for each subspace is a constant $O(k)$. *Hybrid* method uses less time than *MS-index* because it does not need to compute all the pairwise relationship. *SubSky* method is ranked the third in pre-processing time and *MS-index* is the fourth. *Skyey* needs more time due to pre-computing the skyline group and signature [16] as well as finding subspace skyline objects. Obviously, *M-all* is the slowest since it repeats the same routine of skyline computing in 2^d subspaces (cannot draw its entire curve when the dimensionality$\geq$ 10 since its values are too big).

• **Materialized Space Comparisons** We also evaluate the space needed by the above five approaches for the materialization of the information for *SS-query*.



(a) Independent (b) Anti-correlated

Figure 7. Space vs Dimension

We first fix the dimensionality of the datasets as 20, and observe the materialized space in varying with cardinalities of the datasets from 10000 to 100000 tuples. Furthermore, we test the materialization space in the case of varying the dimensions from 10 to 30, with a fixed cardinality of 100,000 tuples. Figure 6(a) and Figure 6(b) show the results of for independent and anti-correlated datasets respectively. As *Sampling* method basically computes the subspace skyline in an on-line fashion, so it need to materialize the full space skyline objects, while sampling k "seeds" in each subspace only occupies constant space cost. Likewise, the space required for *SubSky* is rather small (although more than *Sample*) since it only take up space for a B+ tree. *MS-index* on the other hand is comparable *Skyey* in space consumption. The distribution of space requirement is similar in Figure 7(a) and Figure 7(b), but the materialization size of each method becomes larger. If the dimensionality is larger than 15, the materialization storage of *Sampling*, *Hybrid*, *MS-index*, *SubSky* and *Skyey* increases as well respectively, but each is scalable to the increased dimension, except for *M-all* method.

Over all the above comparisons, we conclude that the pure *MS-index* method and *Hybrid* method provide the best trade-off between query time, pre-processing time and storage space.

7 Conclusions

To improve response time in answering subspace skyline queries, the materialization approach is preferable. In this paper, we first propose a novel notion of the maximal partial-dominating, partial-dominated and equality space of pairs of skyline objects in the full space. Instead of performing any existing skyline computing algorithms, the query processing only involves simple operations of keeping, pruning full space skyline objects to the queried subspace skyline or adding new objects to the queried subspace skyline. We also develop an efficient random sampling-based method to compute the subspace skyline. In the future work, we plan to develop algorithms on subspace skyline integrating with some knowledge discovery operation such as clustering.

References

[1] Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu. A New Method for Similarity Indexing of Market Basket Data. In *SIGMOD* 1999

[2] W. Balke, U. Gntzer Multi-objective Query Processing for Database Systems. In *VLDB 2004*

[3] W.-T. Balke, U. Güntzer, J. X. Zheng. Efficient Distributed Skylining for Web Information Systems. In *EBDT 2004*.

[4] S. Borzsonyi, D. Kossmann, K. Stocker. The Skyline Operator. In *ICDE* 2001.

[5] C. Y. Chan, P. K. Eng, K. L. Tan. Stratified Computation of Skylines with Partially-Ordered Domains. In *SIGMOD* 2005

[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*, pages 503–514, 2006.

[7] J. Chomicki, P. Godfrey, J. Gryz, D. Liang. Skyline with presorting. In *ICDE* 2003.

[8] R. Fagin, A. Lotem, M. Naor. Optimal aggregation algorithms for middleware. In *PODS* 2001.

[9] P. Godfrey, R. Shipley, J. Gryz. Maximal Vector Computation in Large Data Sets. In *VLDB* 2005

[10] Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan. A Microeconomic View of Data Mining. Data Min. Knowl. Discov. 2(4): 311-324 (1998)

[11] Q. Jing , R. Yang , P. Kalnis , A. K. H. Tung. Localized signature table: fast similarity search on transaction. data In *CIKM* 2004

[12] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. In *JACM*, 22(4), 1975.

[13] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB 2002*.

[14] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada: a data cube for dominant relationship analysis. In *SIGMOD Conference*, pages 659–670, 2006.

[15] X. Lin, Y. Yuan, W. Wang, H. Lu. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In *ICDE 2005*

[16] J. Pei, W. Jin, M. Ester, Y. Tao. Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. In *VLDB* 2005

[17] D. Papadias, Y. F. Tao, G. Fu and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD* 2003.

[18] K. Tan, P. Eng and B.Ooi. Efficient Progressive Skyline Computation. In *VLDB 2001*.

[19] Y. Yuan, X. Lin, Q. Liu and W. Wang, J.X. Yu and Q. Zhang. Efficient Computation of the Skyline Cube. In *VLDB 2005*.

[20] Tao, Y., Xiao, X., Pei, J. SUBSKY: Efficient Computation of Skylines in Subspaces. In *ICDE* 2006.

[21] T. Xia, D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. In SIGMOD 2006.

[22] Z. Zhang, X. Guo, H. Lu, A.K.H. Tung, N. Wang. Discovering Strong Skyline Points in High Dimensional Spaces. In *CIKM* 2005.

8 Appendix

8.1 Proof for Lemma 2.2

Proof. Suppose $t(D') \neq t(D'')$, which means at least one pair of objects , say $p_1 < p_2$) in $t(D')$ does not appear in $t(D'')$ in the form of $p_1 > p_2$. It shows p_1 cannot be larger than p_2 in subspace D'', instead $p_1 = p_2$ in D'''. So D'' for (p_1,p_2) is actually $\emptyset$, which is contradiction since $D'' \neq \emptyset$ otherwise (p_1, p_2) cannot be listed in the entry $E(t(D')$, $t(D'')$, $t(D'''))$. So we prove this lemma. □

8.2 Proof for Theorem 2.1

Proof. Given an query subspace Q_s, the skyline object in Q_s is obviously either a full space skyline object, or a non-full space skyline object. Suppose such a non-space skyline object is p, and p is not equal to any other object in each attribute value of Q_s. Since p must be dominated by some objects in D, meanwhile, as p is a subspace skyline in Q_s, p is not dominated by any other objects in Q_s, so p is dominated by some other objects in $D - Q_s$. Thus it infers that p is not dominated by any other objects in D, which contradicts the assumption. So p must share the same values in Q_s with some other objects which are skyline in Q_s. □

8.3 Proof for Lemma 2.3

Proof. Since (1) and (2) are symmetric, we prove (1). Suppose for entry $(t(D'), t(D''), t(D'''))$, $D' \neq \emptyset$, which means that for any object $p \in S$ occurring in entry $(t(D')$, $t(D'')$, $t(D'''))$, there exists another object $q \in S$ in entry $(t(D')$, $t(D'')$, $t(D'''))$ such that $p \succ q$ in D'. If $Q_s \subset D'$, q is still dominated by p in Q_s, so q cannot be a subspace skyline in Q_s, and can be pruned. The case of $Q_s \subset D''$ can be proved in a similar way. Otherwise, if $Q_s \subset D'''$, based on Theorem 2.1, if any object $q \in S$ share the same values in Q_s with objects which are not subspace skyline in Q_s, q will not become the subspace skyline in Q_s either, so q can be pruned. □

8.4 Proof for Lemma 2.4

Proof. For entry $(t(D'), t(D''), t(D'''))$, $D' \neq \emptyset, D'' \neq \emptyset$, which means for **any** object $p \in S$ occurring in (D', D''), there exists another object $q \in S$ in entry $(t(D'), t(D''))$ such that $p \succ q$ in D'. If $Q_s \cap D' \neq \emptyset$, $Q_s \neq D'$, we have $p \succ q$ in $Q_s \cap D' \subseteq D'$; meanwhile if $Q_s \cap D'' \neq \emptyset$, $Q_s \neq D'' \subseteq D''$, we have $p \succ p$ in $Q_s \cap D''$. So, p cannot be dominated by any other object occurring in entry $(t(D'), t(D''), t(D'''))$, and p cannot be pruned. □

8.5 Proof for Theorem 2.2

Proof. (1) Assume that p is not a skyline point in Q_s, then it must be dominated by some point q in Q_s. Let D',D" and D"' be the dominating, dominated and equality space of q on p. Since q dominate p in Q_s, then the dimensions in Q_s are either in D' or D''' with at least one in D'. If this is the case p will be entry invalidated by an entry $(t(D'), t(D''), t(D'''))$ which contradict our condition. (2) can be proven in a similar way to (1). Based on Theorem 2.1, it can be derived that for some non-full space skyline objects, if they share the values for all attributes in Q_s with S, they will naturally become subspace skyline objects in Q_s too. □

8.6 Proof for Lemma 4.1

Proof. Since a skyline object in the subspace Q_s will not be dominated by any other object, the k sample points will also not dominated it and thus the skyline object will not be pruned off. □

8.7 Proof for Lemma 4.2

Proof. All false skyline candidates in Q_s that pass through the filtering will be dominated by some skyline points in Q_s. Since no real skyline object in Q_s are removed based on Lemma 4.1, these skyline points will be used to remove the false skyline candidates during the skyline computation. □