Continuous k-Dominant Skyline Computation on Multidimensional Data Streams*

M. Kontaki
Aristotle University
Department of Informatics
54124 Thessaloniki, GREECE
kontaki@csd.auth.gr

A.N. Papadopoulos
Aristotle University
Department of Informatics
54124 Thessaloniki, GREECE
papadopo@csd.auth.gr

Y. Manolopoulos
Aristotle University
Department of Informatics
54124 Thessaloniki, GREECE
manolopo@csd.auth.gr

ABSTRACT

Skyline queries are important due to their usefulness in many application domains. However, by increasing the number of attributes, the probability that a tuple dominates another one is reduced significantly. To attack this problem, k-dominant skylines have been proposed, relaxing the definition of domination. In this paper, we study the problem of continuous monitoring of k-dominant skylines, where multiple queries are running concurrently. The proposed method divides the space in pairs of attributes. For each pair, we compute skyline tuples and we exploit them to eliminate candidates tuples of the queries and we combine the partial results. The proposed scheme uses only simple domination checks and it is applicable to the streaming case as well as to ad-hoc insertions and deletions. Experiments, based on different data distributions, show the efficiency of the proposed scheme in comparison to existing methods.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*query processing*

General Terms

Algorithms, Experimentation, Performance

Keywords

Continuous Processing, Skylines, Data Streams

1. INTRODUCTION

Recently, skyline queries have attracted the research interest significantly. Skyline queries are frequently used in multicriteria decision making applications, where a number of (usually) contradictory criteria participate towards selecting the most convenient answers to the user.

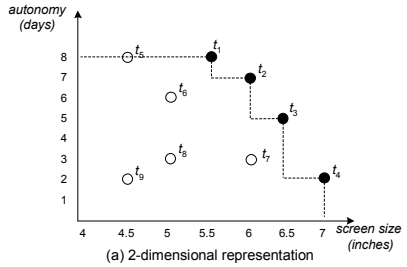
*Research supported by the PENED 2003 program, funded by the GSRT, Ministry of Development, GREECE.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil
Copyright 2008 ACM 978-1-59593-753-7/08/0003 ...$5.00.

Assume that a customer is interested in purchasing a PDA device. Assume further, that the customer focuses on two important characteristics of a PDA, the screen size and the battery autonomy (time interval between succesive battery charges). Unfortunately, these criteria are usually contradictory (the larger the screen the more significant the energy consumption) and therefore, the PDAs that best match the customer's needs should be carefully selected. In this example, each PDA is represented as a tuple containing two attributes (size and time) and the customer is interested in items that maximize these attributes. Depending on the semantics of each attribute, in other cases the customer may ask for minimization of the attributes, or any other combination. Without loss of generality, in the sequel we focus on maximizing the attributes of interest. However, the proposed technique is directly applicable to other cases as well.

The PDA example is depicted in Figure 1. In Figure 1(a) each PDA is represented by a two-dimensional point, where each dimension corresponds to an attribute (in the sequel the terms attribute and dimension are used interchangeably). PDA tuples (also termed points) are shown in Figure 1(b). Points connected by the dashed line comprise the skyline, which we will henceforth refer to as *simple skyline* and it consists of points t_1, t_2, t_3, and t_4.

The literature is reach in algorithms of skyline query processing for static [3, 5] and dynamic data [4]. All the previous efforts report the skyline result taking into consideration all dimensions. There are two problems with this perspective: (i) in high dimensionalities the number of skyline points increases substantially [1, 2], and (ii) users may not be interested in the whole set of dimensions. To handle these issues, subspace skylines have been proposed in [6] and several efforts have been performed to provide efficient subspace skyline computation [7, 8]. Towards eliminating the huge number of skyline points, a novel method has been



id	screen size	autonomy
t_1	5.5	8
t_2	6	7
t_3	6.5	5
t_4	7	2
t_5	4.5	8
t_6	5	6
t_7	6	3
t_8	5	3
t_9	4.5	2

Figure 1: Skyline example.

proposed in [2] which relaxes the dominance definition, in order to increase the probability that a point will dominate another. Instead of searching for ordinary skyline points in all dimensions, k-domimant skylines are being used.

The processing techniques reported in [2] are executed on static data sets. However, in case of insertions and deletions the skyline result should be updated accordingly. In the sliding window paradigm, insertions and deletions are very frequent. In our example, a customer may be interested in a PDA that belongs to the n most recent models. Our first contribution involves the continuous processing of k-dominant skylines. Our second contribution involves the efficient processing of multiple continuous queries. Each query may be defined in a subset of the available dimensions, since different users are usually interested in different attributes. The parameter k set by each query may vary, thus increasing the challenge even further.

The rest of the work is organized as follows. Section 2 provides background material. Our proposal is given in Section 3, whereas Section 4 gives some representative experimental results. Finally, Section 5 concludes our study.

2. BACKGROUND MATERIAL

Table 1 summarizes the basic symbols that are used throughout the study. The proofs of propositions as well as the pseudocode of algorithms are omitted due to lack of space.

Assume a D-dimensional space $\mathbf{D} = \{d_1, d_2, ..., d_D\}$ and a set of tuples $\mathbf{T} = \{t_1, t_2, ..., t_T\}$. We use $t_{i,j}$ to denote the value of the j-th dimension of the i-th tuple. Moreover, assume a number of k-dominant skyline queries. Each query q_i is defined on a set of dimensions $q_i.\mathbf{ds} \subseteq \mathbf{D}$ in which the query is applied and has a parameter $q_i.k$ which specifies the desired number of k-dominant skyline tuples.

Definition 1 (*k-dominated tuple*)
A tuple t_i is *k-dominated* by a tuple t_j in query q_l, if and only if $\exists$ **D'** $\subseteq q_l.\mathbf{ds}$, $D' = q_l.k$, $\forall\ d_x \in$ **D'**, $t_{j,x} \geq t_{i,x}$ and $\exists$ $d_y \in$ **D'**, $t_{j,y} > t_{i,y}$. □

Definition 2 (*k-dominant skyline tuple*)
A tuple t_i is *k-dominant skyline tuple* of query q_l if and only if there does not exist any tuple t_j in the data set that k-dominates t_i in dimensions $q_l.\mathbf{ds}$. □

Symbol	Description
D,**D'** and D,D'	dimensions sets and the number of dimensions
d, d_i	a dimension
T, T	tuples set and the number of tuples
t_i	the i-th tuple
$t_{i,j}$	the value of i-th tuple in j-th dimension
Q, Q	queries set and the number of queries
q, q_i	queries
$q.\mathbf{ds}$, $q_i.\mathbf{ds}$ $q.ds$, $q_i.ds$	the set and the number of dimensions of queries
$q.k$, $q_i.k$	parameter k for a query
sq, sq_j	subqueries
$sq.\mathbf{ds}$, $sq_j.\mathbf{ds}$ $sq.ds$, $sq_i.ds$	the set and the number of dimensions of subqueries
$g_{i,j}$	grid of the i-th and j-th dimensions
c,$c_{i,j}$	cells of a grid

Table 1: Basic symbols used throughout the study.

We examine the data space in pairs of dimensions. Therefore, if D is the total number of dimensions, we use $g_{i,j}$ ($1 \leq i, j \leq D \cdot (D-1)/2$ and $i < j$) to denote the grid formed by the i-th and j-th dimensions.

Definition 3 ((2,**D'**)-*skyline tuple*)
Given a subset of two dimensions **D'** $\subseteq$ **D**, a tuple t_l is (2,**D'**)-*skyline tuple* if and only if it is a simple skyline tuple in at least one $g_{i,j}$ ($d_i \in$ **D'**, $d_j \in$ **D'** and $i \neq j$). □

Definition 4 ((D',**D'**)-*skyline tuple*)
Given a subset of dimensions **D'** $\subseteq$ **D** ($D' > 2$), a tuple t_i is (D',**D'**)-*skyline tuple* if and only if it is a simple skyline tuple in **D'** and it is not a (2,**D'**)-skyline tuple. □

A query q_i, defined on dimensions $q_i.\mathbf{ds}$, with parameter $q_i.k$ has $\binom{q_i.ds}{q_i.k}$ subqueries. For example, a query q_i with $q_1.ds = 4$ and $q_i.k = 3$ has 4 subqueries.

Definition 5 (*subquery*)
Given a query q_i with $q_i.\mathbf{ds}$ and $q_i.k$, a *subquery* sq_j of q_i has one out of $\binom{q_i.ds}{q_i.k}$ possible combinations of the dimensions of q_i. □

The next proposition allows us to identify skyline points in subspaces, but it holds only if the distinct value condition holds [6].

Proposition 1
Given T D-dimensional tuples, if a tuple t_i is a simple skyline tuple in $\mathbf{D}_1 \subseteq \mathbf{D}$ then it is simple skyline tuple in every set of dimensions $\mathbf{D}_2 \supseteq \mathbf{D}_1$ ($\mathbf{D}_2 \subseteq \mathbf{D}$). □

The previous property does not hold in the case of k-dominant skylines when $k < D$. Additionally, the transitive property of simple skylines does not hold in k-dominant skylines. Therefore it is possible to have 3 tuples t_1, t_2 and t_3 such that t_1 is k-dominated by t_2, t_2 is k-dominated by t_3 and t_3 is k-dominated by t_1 (cyclic dominant relationship). Due to cyclic dominant relationships, we cannot discard a tuple that is k-dominated since it might be used to exclude another tuple. Thus, algorithms that have been proposed for skyline and continuous skyline computation are not applicable in our case. In the sequel, we describe the CoSMuQ algorithm (Continuous Skylines for Multiple Queries), which attacks the following problem:

Given a dynamic data set of D-dimensional tuples and a set of k-dominant skyline queries, each specifying a set of dimensions $q_i.\mathbf{ds} \subseteq \mathbf{D}$ and a parameter $q_i.k \leq q_i.ds$, **compute** the k-dominant skyline for each query continuously.

3. CONTINUOUS K-DOMINANT SKYLINES

3.1 Data structures

The proposed method maintains a grid for each pair of dimensions. We used grids with equal-sized cells but the method can also work with irregular grids as well. The advantage of equal-sized cells is that we can insert/delete a data to/from the grid very efficiently. Each cell contains the IDs of the tuples that lie in this cell and also the coverage of the cell. The coverage of the cell is the number of tuples that dominate this particular cell. Finally, each grid maintains

its skyline tuples.

When a new query is posed, we initially compute its subqueries. Each subquery sq_j of query q_i has an ID and a subset of dimensions of the query denoted as $sq_j.\mathbf{ds}$ ($sq_j.ds = q_i.k$). Additionally each subquery stores the (2,$sq_j.\mathbf{ds}$)-skyline tuples, the ($q_i.k$,$sq_j.\mathbf{ds}$)-skyline tuples and a set of candidate skyline tuples. Notice that it is possible, queries with the same parameter k to share common subqueries. Moreover, the method maintains the queries into the list of queries. Each query has an ID, a parameter k, a subset of dimensions, the IDs of the subqueries of the query and the k-dominant skyline of this query.

Before we proceed with the description of CoSMuQ, we explain the three types of skyline tuples that a subquery stores. Each subquery sq of a query q maintains the (2,$sq.\mathbf{ds}$)-skyline tuples, the ($q.k$, $sq.\mathbf{ds}$)-skyline tuples and the candidate skyline tuples. The (2, $sq.\mathbf{ds}$)-skyline is the union of the skylines of the grids $g_{i,j}$ with $d_i \in sq.\mathbf{ds}$, $d_j \in sq.\mathbf{ds}$ and $i \neq j$. A tuple is candidate skyline tuple if it is not dominated by the (2,$sq.\mathbf{ds}$)-skyline tuples of the subquery, therefore the candidate skylines is a superset of the ($q.k$,$sq.\mathbf{ds}$)-skylines. A candidate skyline tuple is ($q.k$,$sq.\mathbf{ds}$)-skyline tuple if it is not dominated by any existing ($q.k$,$sq.\mathbf{ds}$)-skyline tuple.

3.2 The CoSMuQ Approach

Our main focus is to compute k-dominant skylines of multiple queries continuously. Thus, we deal with the tuple insertions and deletions rather than with the initial phase of discovering k-dominant skylines. Besides, the initial phase can be processed by applying consecutive insertions.

First, let us examine a subquery as a simple skyline query. We assume that the distinct value condition holds. Then, due to Proposition 1, the skyline of the subquery is the union of (2,$sq.\mathbf{ds}$)-skyline and ($sq.ds$,$sq.\mathbf{ds}$)-skyline. The following proposition explains:

Proposition 2 (*subquery skylines*)
The simple skyline of a subquery sq of a query q with a set of dimensions $sq.\mathbf{ds}$ is given by the union of (2, $sq.\mathbf{ds}$)-skyline and ($q.k$,$sq.\mathbf{ds}$)-skyline of the subquery. □

In the case where the distinct value condition does not hold, elements with equal values in the two dimensions of a grid can be stored in a buffer enabling further examination during the update of the subqueries. The key observation in order to answer k-dominant skyline queries, is that the intersection of the skylines of the subqueries is the skyline of the k-dominant query.

Proposition 3 (*query k-dominant skyline*)
The k-dominant skyline of a query q with $q.\mathbf{ds}$ and $q.k$ is given by the intersection of the simple skylines of the subqueries of the query. □

We proceed now with the detailed description of our algorithm. The proposed method comprises three basic stages. The first stage updates the grids and the skylines of the grids. The second stage updates the subqueries, whereas the third stage updates the queries. We begin the description of the CoSMuQ algorithm with the insert operation. We will illustrate the CoSMuQ-insert procedure by means of an example depicted in Figures 2, 3 and 4. Assume that there are four 4-dimensional tuples, therefore $T = D = 4$ and there is only one query with $q_1.\mathbf{ds} = \{d_1, d_2, d_3, d_4\}$) ($q_1.ds = 4$) and $q_1.k = 3$. Figure 2 depicts the tuples and the 2 out of 6 grids, one for each pair of dimensions. Each cell contains its coverage and the tuples that lie in this cell. Assume that a new tuple t_5 arrives.

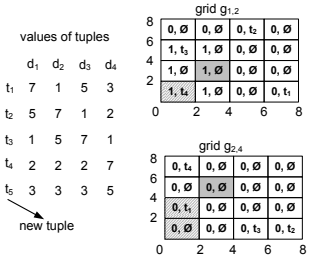


Figure 2: Data values and grids.

The first step is to insert tuple t_5 into the grids. The tuple is inserted into the shaded cells of Figure 2. Moreover the coverage of the cells should be updated. The coverage of the cells that are dominated by the shaded cell is increased by one. In Figure 2 these are the striped cells.

The next step is to update the skyline tuples of each grid. Figure 3 depicts the skylines of the grids before the insertion of t_5 (left) and after the insertion (right). First, the new tuple is checked if it is skyline tuple in the grid. If the coverage of its cell is above 0 then the new tuple is definitely not a skyline tuple. If the coverage is 0, then the tuple is compared with the skyline tuples of the grid. Notice that we compare the new tuple only with the skyline tuples of the grid and not with any other tuple since we compute simple skylines and not k-dominant ones. Therefore, the transitive property holds. If the new tuple is a skyline tuple of a grid, it should be checked if it dominates existing skyline tuples of this grid. These skyline tuples are deleted and are temporarily stored. Let us examine the update of the skylines of the grids $g_{1,2}$ and $g_{2,4}$. The cell $c_{2,2}$ of $g_{1,2}$ has coverage 1 due to t_2, thus t_5 is not skyline tuple. The cell $c_{2,3}$ of $g_{2,4}$ has coverage 0, therefore we check t_5 with the skyline tuples t_2 and t_4 of the grid to conclude that t_5 belongs to the skyline of this grid.

Figure 4 illustrates the subqueries before (left) and after (right) the insertion of t_5. Before the insertion, the tuple t_4 is a candidate skyline tuple for sq_1 because is not dominated by the (2,$sq_1.\mathbf{ds}$)-skyline and is (3,$sq.\mathbf{ds}$)-skyline tuple since it is not dominated by any other (3,$sq.\mathbf{ds}$)-skyline tuple.

The update of the subqueries follows. First the new tuple t_5 is checked if it should be inserted in the (2,$sq.\mathbf{ds}$)-skyline. Remember that the (2,$sq.\mathbf{ds}$)-skyline is the union of the skylines of the grids that belong to $sq.\mathbf{ds}$. Since t_5

Grids	Before the insertion of t_5 Skylines	After the insertion of t_5 Skylines
$g_{1,2}$	t_1, t_2	t_1, t_2
$g_{1,3}$	t_1, t_3	t_1, t_3
$g_{1,4}$	t_1, t_4	t_1, t_4, t_5
$g_{2,3}$	t_2, t_3	t_2, t_3
$g_{2,4}$	t_2, t_4	t_2, t_4, t_5
$g_{3,4}$	t_1, t_3, t_4	t_1, t_3, t_4, t_5

Figure 3: Skylines of Grids.

subquery	dimensions	Before the insertion of t_5			After the insertion of t_5		
		(2, {d_1,d_2,d_3})	(3, {d_1,d_2,d_3})	cand	(2, {d_1,d_2,d_3})	(3, {d_1,d_2,d_3})	cand
sq_1	{d_1, d_2, d_3}	t_1, t_2, t_3	t_4	t_4	t_1, t_2, t_3	t_5	t_4, t_5
sq_2	{d_1, d_2, d_4}	t_1, t_2, t_4			t_1, t_2, t_4, t_5		
sq_3	{d_1, d_3, d_4}	t_1, t_3, t_4			t_1, t_3, t_4, t_5		
sq_4	{d_2, d_3, d_4}	t_1, t_2, t_3, t_4			t_1, t_2, t_3, t_4, t_5		

Figure 4: Information maintained for Subqueries.

is skyline tuple in $g_{1,4}$, $g_{2,4}$ and $g_{3,4}$, it will be inserted in the (2,sq.**ds**)-skylines of subqueries sq_2, sq_3 and sq_4. In the case where the distinct value condition does not hold and the new tuple is equal with at least one skyline tuple of a grid, we should further examine these tuples. If the new tuple is (2,sq.**ds**)-skyline tuple, the ($sq.ds$,sq.**ds**)-skyline and the candidate skyline tuples that are dominated by the new tuple are discarded. Next, we examine the deleted skyline tuples of the grids, that were temporarily stored during the first stage. We need to check if the deleted skyline tuples of the grids should be removed from the (2,sq.**ds**)-skyline of the subquery. This can happen if a deleted skyline tuple does not exist as a skyline tuple in any other grid affecting the subquery. In this case, the deleted skyline tuple is removed from (2,sq.**ds**)-skyline and if it is not dominated by the new tuple, it is inserted in the candidate skyline and in the ($sq.ds$,sq.**ds**)-skyline, since it is not dominated by any other tuple.

If the new tuple is not (2,sq.**ds**)-skyline tuple, we compare it with all the (2,sq.**ds**)-skyline tuples. If the new tuple is not dominated by them it is candidate skyline tuple and if it is not dominated by any ($sq.ds$,sq.**ds**)-skyline tuple then it is ($sq.ds$,sq.**ds**)-skyline tuple. If the new tuple is ($sq.ds$,sq.**ds**)-skyline tuple, we check if it dominates some existing ($sq.ds$,sq.**ds**)-skyline tuples and we discard them. In our example t_5 is not skyline tuple of $g_{1,2}$, $g_{1,3}$ and $g_{2,3}$, therefore it is not (2,sq.**ds**)-skyline tuple of sq_1. Then we check if the new tuple is dominated by the (2,sq.**ds**)-skyline tuples of sq_1. It is not dominated, thus it is inserted in the candidate skyline. It is not dominated by any ($sq.ds$,sq.**ds**)-skyline tuple, thus it is inserted in them. Finally, we compare t_5 with t_4 and we discard t_4, since it is dominated by t_5. Notice that in the procedure of updating subqueries, we perform only simple skyline comparisons. Moreover if a new tuple is not inserted in the (2,sq.**ds**)-skyline or in the ($sq.ds$,sq.**ds**)-skyline, we do not perform extra operations. Finally, to update a subquery only the tuples of the (2,sq.**ds**)-skyline, ($sq.ds$,sq.**ds**)-skyline and candidate skyline are needed.

The third stage involves the update of the result of k-dominant queries. Due to Proposition 2 and 3, we first compute the union of (2,sq.**ds**)-skyline and ($sq.ds$,sq.**ds**)-skyline of each subquery of the query and then we intersect the results. For the query of the example, the k-dominant skyline consists of tuples t_1 and t_5.

The CoSMuQ-delete procedure handles deletions. A delete operation comprises similar operations of the insertion procedure. The description of CoSMuQ-delete procedure is omitted due to space limitation. Both the CoSMuQ-insert and CoSMuQ-delete procedures do not use k-dominant comparisons but only simple skyline comparisons. An advantage of the method is that it can distinguish cases and perform a minimum set of operations. A second advantage is that the method exploits the overlap between different queries. This is based on the observation that skyline queries of different users have frequently common dimensions (attributes). If there is an adequate number of queries, the gain of the usage of the pairs outperforms the cost of updating the grids. Extensive experimentation shows that even a number of 10 queries sharing a subset of dimensions, is adequate to overcome the cost introduced by the usage of the grids. Moreover, the cost of updating the grids is low since a) only grids that participate in at least one query should be updated and b) the skyline points in two dimensions can be computed very easily. Additionally, CoSMuQ can handle insertions and deletions of queries. Finally, the method is appropriate both in ad-hoc and streaming scenarios.

4. PERFORMANCE EVALUATION

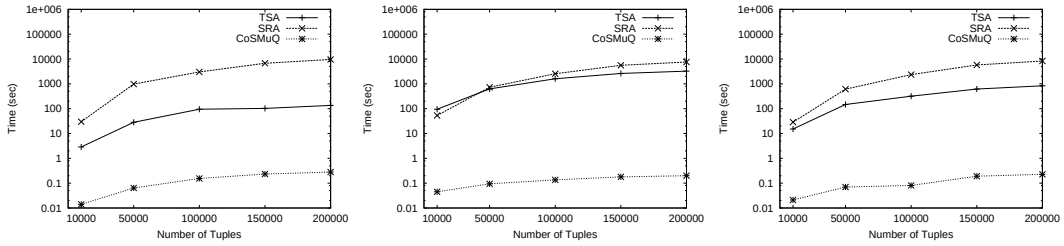
We have conducted a series of experiments to evaluate the performance of our scheme. We use the TSA and SRA algorithms, proposed in [2], as the competitors of our technique, bearing in mind that, although very efficient, they were not originally developed for dynamic environments and multiple queries. For fairness, we have modified the SRA algorithm to apply binary search for tuple insertions or deletions instead of sorting all tuples in each update. All algorithms have been implemented in C++ and the experiments have been conducted on a Pentium IV system at 3GHz, with 1GB of main memory running Windows XP.

The three data sets (Correlated, Independent and Anti-Correlated) have been generated by using the process described in [1]. We examine the performance of the methods by varying the most important parameters (i.e., number of tuples, number of dimensions and number of queries). The default parameter values (if not otherwise specified) are: the number of active tuples is 10,000, the number of dimensions is 10, the number of dimensions of each query varies between 6 and 10 and the number of queries is 500. The parameter $q.k$ of each query q lies in the range $[q.ds/2, q.ds)$. The number of dimensions $q.ds$, the set of the dimensions q.**ds** and the parameter $q.k$ of the queries are generated uniformly. The grid size is set to 30 cells per dimension. The number of updates varies between 10,000 and 200,000. Performance is measured by the CPU time per update.

The first experiment measures the response time with respect to the number of tuples. Figure 5 depicts the results for all the data sets. The gap between TSA and SRA is reduced in the Anti-Correlated and Independent data sets because the number of k-dominant skyline tuples is higher and therefore TSA cannot prune many tuples in the first scan. CoSMuQ outperforms both TSA and SRA in all data sets. CoSMuQ is almost unaffected by the number of tuples, since it does not evaluate queries from scratch.

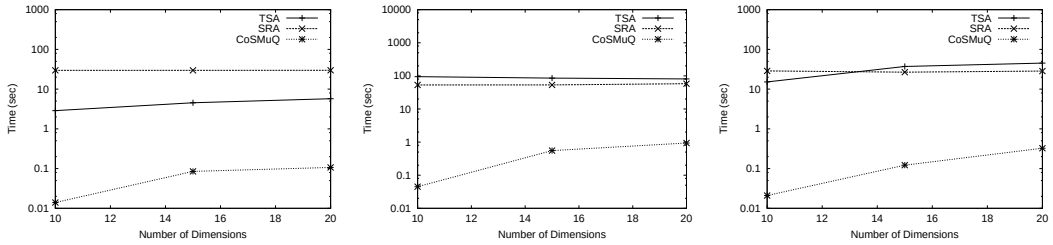
The second experiment studies the response time with respect to the number of dimensions. Figure 6 illustrates the results. SRA is better than TSA in the Anti-Correlated and Independent data sets in almost all cases for the same reason. In this experiment, the number of comparisons is more due to the increase of the number of dimensions, which affects more the TSA algorithm. The response time of our method increases as the number of dimensions increases but it remains lower than 1sec even for 20 dimensions.

Finally, in the third experiment (Figure 7) we show the response time vs. the number of queries. The number of queries varies from 100 to 10,000. CoSMuQ outperforms both TSA and SRA even for 100 queries. Moreover the pro-



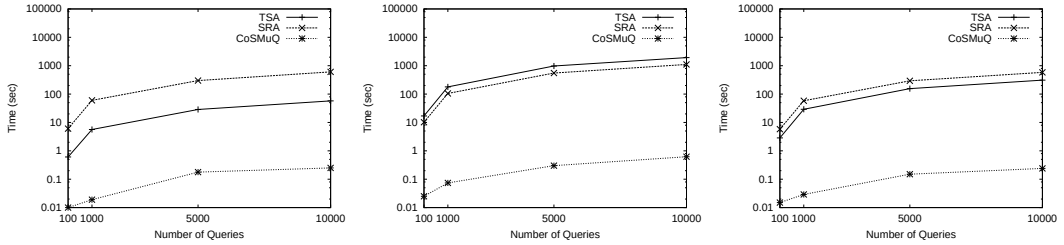
(a) Correlated (b) Independent (c) Anti-Correlated

Figure 5: CPU Time vs. Number of Tuples



(a) Correlated (b) Independent (c) Anti-Correlated

Figure 6: CPU Time vs. Number of Dimensions



(a) Correlated (b) Independent (c) Anti-Correlated

Figure 7: CPU time vs. number of queries (the time axis is shown in logarithmic scale)

posed method has response time less than 1 sec in all cases. Note that, the queries have randomly generated subsets of dimensions that cover all the dimensions and various values for the parameter k. This is an indication that the proposed method will have even better performance in the case where some dimensions are usually more preferable than others.

5. CONCLUSIONS

In this paper, a novel method has been proposed towards efficient processing of continuous k-dominant skylines for multiple queries. The proposed algorithm, CoSMuQ, is appropriate for ad-hoc as well as streaming scenarios. Performance evaluation illustrates the superiority of the proposed method against the TSA and SRA algorithms. Moreover, it demonstrates the capability of the proposed method to manage multiple queries. Future work may include the extension of our method to handle δ-skyline queries.

6. REFERENCES

[1] S. Borzsonyi, D. Kossmann, K. Stocker: "The Skyline Operator", *Proc. of ICDE*, pp.421-430, 2001.

[2] C. Chan, H.V. Jagadish, K. Tan, A. Tung, Z. Zhang: "Finding k-dominant skylines in high dimensional space", *Proc. of SIGMOD*, pp.503-514, 2006.

[3] W. Jin, J. Han, M. Ester. "Mining Thick Skylines over Large Databases", *Proc. of PKDD*, pp.255-266, 2004.

[4] X. Lin, Y. Yuan, W. Wang, H. Lu: "Stabbing the Sky: Efficient Skyline Computation over Sliding Windows", *Proc. of ICDE*, pp.502-513, 2005.

[5] D. Papadias, Y. Tao, G. Fu, B. Seeger. "Progressive Skyline Computation in Database Systems", *ACM TODS*, Vol.30, No.1, pp.41-82, 2005.

[6] J. Pei, W. Jin, M. Ester, Y. Tao. "Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces", *Proc. of VLDB*, pp.253-264, 2005.

[7] Y. Tao, X. Xiao, J. Pei. "SUBSKY: Efficient Computation of Skylines in Subspaces", *Proc. of ICDE*, pp.65-74, 2006.

[8] T. Xia, D. Zhang. "Refreshing the Sky: The Compressed Skycube with Efficient Support for Frequent Updates", *Proc. SIGMOD*, pp.491-502, 2006.