Proceedings of the 12th Panhellenic Conference on Informatics (PCI 2008), Samos Island, Greece, 28-30 August, 2008

Continuous Top-k Dominating Queries in Subspaces*

Maria Kontaki
Dept.of Informatics
Aristotle University
Thessaloniki, Greece
kontaki@delab.csd.auth.gr

Apostolos N. Papadopoulos
Dept.of Informatics
Aristotle University
Thessaloniki, Greece
apostol@delab.csd.auth.gr

Yannis Manolopoulos
Dept.of Informatics
Aristotle University
Thessaloniki, Greece
manolopo@delab.csd.auth.gr

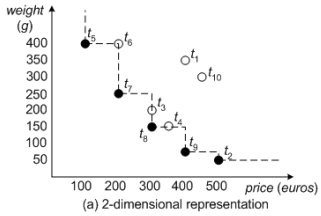
Abstract

Dominating queries are significant tools for preference-based query processing in databases and decision support applications. An important preference-based query is the top-k dominating query, which reports the k most important objects according to their domination capabilities (score). In this paper, we address the following issues to tackle two limitations of previously proposed approaches: (i) we allow dominating queries to be expressed in a subset of the available dimensions and (ii) we provide the necessary techniques to enable continuous processing of multiple queries. We use a grid-based indexing scheme to facilitate efficient search and update operations, avoiding expensive reorganization costs. In addition, several optimizations are proposed to enhance efficiency. Performance evaluation results, based on real-life and synthetic data sets, show the efficiency and scalability of the proposed scheme.

1 Introduction

Preference queries are frequently used in multicriteria decision making applications, where a number of (usually) contradictory criteria participate towards selecting the most promising answers. Assume that a customer is interested in purchasing a PDA device. Assume further that the customer focuses on two important parameters of a PDA: the price and the weight. Unfortunately, these two criteria are frequently contradictory and therefore, the number of candidates should be carefully selected. Depending on the semantics of each attribute, we may ask for maximization of the attributes, or any combination (minimization in some attributes and maximization in the others).

The *skyline* of a set of tuples comprises all tuples that are not dominated. A tuple t_i dominates another tuple t_j, if t_i is as good as t_j in all dimensions and it is better than t_j in at least one of the dimensions. Let d be the total number of attributes (dimensions) and $t_{i,j}$ denote the value of the j-th dimension of the i-th tuple. Since we have assumed that "smaller is better", t_i is better than t_j in dimension m if $t_{i,m} < t_{j,m}$. The PDA example is depicted in Figure 1. In Figure 1(a) each PDA is represented by a two-dimensional tuple (each dimension corresponds to an attribute). PDA tuples are shown in Figure 1(b). The black dots of Figure 1(a) represent the skyline.



id	price	weight
t_1	400	350
t_2	500	50
t_3	300	200
t_4	350	150
t_5	100	400
t_6	200	400
t_7	200	250
t_8	300	150
t_9	400	75
t_{10}	450	300

(b) PDA tuples

Figure 1. Skyline example.

Skyline queries have received considerable attention recently, due to their aid in selecting the most preferred items, especially when the selection criteria are contradictory. In [4] an efficient skyline query processing scheme has been proposed based on branch-and-bound, which utilizes the R-tree spatial access method. Another technique has been proposed in [2] for selecting skyline tuples according to their domination capabilities. These schemes however, assumes that the skyline is computed over the whole set of attributes, which in many cases may not be meaningful.

The fact that a tuple is contained in the skyline does not necessarily means that it will be a good choice for the user. For example, tuple t_2 although it is a skyline tuple, it does not dominate any point in the data set. On the other hand, tuple t_8 dominates four tuples, which means that it possesses significant domination power. The domination power of a tuple is expressed by the number of tuples it dominates, and leads to an intuitive ranking of tuples. Note that a tuple not in the skyline may have more domination power than a skyline tuple. For example, tuple t_3 has more domination power than the skyline tuple t_2. The k tuples with the highest domination power are called *top-k dominating tuples*. Techniques for top-k dominating queries have been developed in [5, 6]. In [6] the authors propose efficient algorithms to determine the top-k dominating tuples by using

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

an aggregate R-tree index. In [5], a method is studied to rank multidimensional tuples according to their domination power. However, the problem has not been addressed from a data stream perspective, where the data change frequently (e.g., in a sliding-window manner). In such a case, the result of the query should be updated by using the previous result, avoiding expensive recomputations. Another limitation of previously proposed methods is that they assume that the user is interested in the whole set of attributes (dimensions). However, a user may be interested only on a subset of the available attributes. In this paper, we handle both issues by proposing an organization scheme and the associated algorithms to provide efficient processing of continuous top-k dominating queries in a subset or the whole set of the attributes.

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

2 Continuous Top-k Dominating Queries

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

Assume a D-dimensional space $\mathbf{D} = \{d_1, ..., d_D\}$ and a set of tuples $\mathbf{T} = \{t_1, ..., t_T\}$. Moreover, assume a number of Q top-k dominating queries. Each query q_i is applied in a subset of dimensions $q_i.\mathbf{D} \subseteq \mathbf{D}$ and it has a parameter $q_i.k$ which specifies the desired number of tuples.

Definition 1 (*dominated tuple*)
A tuple t_i is *dominated* by a tuple t_j in query q, if and only if $\forall\ d_x \in q.\mathbf{D}$, $t_{j,x} \leq t_{i,x}$ and $\exists\ d_y \in q.\mathbf{D}$, $t_{j,y} < t_{i,y}$.

Definition 2 (*top-k dominant tuple*)
A tuple t_i is a *top-k dominant tuple* of query q if and only if it dominates one of the k-th highest number of tuples in dimensions $q.\mathbf{D}$.

Symbol	Description
D,**D'** and D,D'	dimensions sets and number of dimensions
d, d_i	a dimension
T, T	tuples set and the number of tuples
$t_{i,j}$	the value of the i-th tuple in j-th dimension
$t_i.score$	the number of tuples dominated by t_i
Q, Q	queries set and the number of queries
q, q_i	a query
$q.\mathbf{D}$, $q.D$	the set and the number of dimensions of q
$q.k$, $q_i.k$	parameter k for a query
q.**top**	the top-k dominant tuples of a query
q.**score**	the scores of k dominant tuples of q
$q.kscore$	the k-th score of k dominant tuples of q
c, c_i	grid cells
$p(c)$	the projection of c in a subspace
$c.min$, $c.max$	minimum and maximum coverage of a cell
$c.n$	the number of tuples of a cell

Table 1. Basic symbols.

The number of tuples that a tuple t dominates, is denoted as $t.score$, and this value expresses the domination power of the tuple. The above definition implies that the answer is composed of all the tuples that have the k highest scores. A top-k dominating query is a combination of skyline and top-k queries, resulting in a more complex one. Top-k dominating queries use the dominant relationship rather than a user-defined score function. The determination of the appropriate score function is not obvious, especially when the number of attributes increases. Moreover, top-k dominating queries bound the size of the resulting set of tuples, in contrast to skyline queries.

2.1 Indexing and Processing

Due to the dynamic nature of the problem, to avoid high reorganization costs, it is more preferable to use a simple structure for quick maintenance. This approach has been followed in other research proposals dealing with dynamic data sets [3]. For clarity, we assume a regular grid to illustrate the basic properties of the proposed method. Irregular grids are described later.

Without loss of generality, we focus on minimizing the attributes of interest. Each cell c_i dominates all cells located at higher positions in all dimensions. Moreover, cell c_i does not dominate a cell that lies in at least one lower positions of it. The remaining cells are partially dominated by c_i, which means that a tuple t_j of c_i may or may not dominate some tuples of the partially dominated cells. For example, cell c_6 of Figure 2 dominates cells c_{11}, c_{12}, c_{15} and c_{16}, whereas it partially dominates cells c_6, c_7, c_8, c_{10} and c_{14}. The remaining cells are not dominated by c_6.

Each cell contains the IDs of the tuples that lie in this cell and the *maximum coverage* of the cell. The maximum coverage of a cell is the maximum number of tuples that it is possible to be dominated by any tuple of this particular cell. Therefore, to calculate the maximum coverage of c_i, we first determine the cells that c_i fully and partially dominates and then, we add the numbers of the tuples of these cells. Similarly, the *minimum coverage* of a cell c_i, denoted as $c_i.min$, is the sum of the numbers of the tuples of cells that c_i fully dominates. The minimum coverage of cell c_i is equal to the maximum coverage of cell c_j that lies in the immediately higher position of c_i in all dimensions, e.g., $c_{10}.min = c_{15}.max = 1$.

Queries are maintained in a hash table for quick refer-

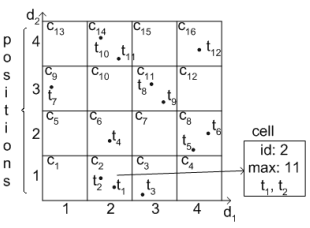


Figure 2. Grid structure.

ence. Each query has an ID, a parameter k, a subset of dimensions q.**D**, the top-k dominating tuples of this query q.**top** and their score values q.**score**.

We use the notation $p(c_i)$ to denote the result of the projection of a cell c_i in a subspace. The following propositions explain how the maximum and minimum coverage of a projected cell can be computed.

Proposition 1 (*maximum coverage of* $p(c)$)
Assume a subspace **D'** $\subset$ **D** and a cell c_i with positions $pos_{i,1}, pos_{i,2}, ..., pos_{i,D}$. Cell $p(c_i)$ has maximum coverage $p(c_i).max = c_j.max$, where c_j is the cell with positions $pos_{j,x} = pos_{i,x}\ \forall\ d_x \in$ **D'** and $pos_{j,x} = 1$ $\forall d_x \in$ **D** and $d_x \notin$ **D'**.

Proposition 2 (*minimum coverage of* $p(c)$)
Assume a subspace **D'** $\subset$ **D** and a cell c_i with positions $pos_{i,1}, pos_{i,2}, ..., pos_{i,D}$. Cell $p(c_i)$ has minimum coverage $p(c_i).min = c_j.max$, where c_j is the cell with positions $pos_{j,x} = pos_{i,x} + 1\ \forall\ d_x \in$ **D'** and $pos_{j,x} = 1\ \forall d_x \in$ **D** and $d_x \notin$ **D'**.

For example, assume the grid of Figure 2 and a query q with q.**D** $= \{d_1\}$. Cell $p(c_6)$ has $p(c_6).max = c_2.max$ and $p(c_6).min = c_3.max$. Notice that a cell $p(c_i)$ can be the same as $p(c_j)$, e.g., cells $p(c_6)$ and $p(c_{10})$.

We can use the grid structure to compute the $t_x.score$ of a tuple t_x in all dimensions. First, we find the cell c_i that t_x belongs to. The value $t_x.score$ lies between the range $[c_i.min, c_i.max]$. To compute $t_x.score$, it is necessary to check how many tuples n are dominated by t_x only in the partially dominated cells of c_i. Then $t_x.score = c_i.min + n$. In Figure 2, tuple t_4 dominates the tuples t_6 and t_{11} of the partially dominated cells thus $t_1.score = 3 + 2 = 5$.

Assume a query q with q.**D** $\subset$ **D**. There are some differences to compute $t_x.score$ in the subspace q.**D**. After we find the cell c_i that t_x belongs, we project the cell to the subspace of the query q. Now, we should check how many tuples are dominated by t_x in the partially dominated cells of $p(c_i)$. Again, if n is the number of the tuples, $t_x.score = p(c_i).min + n$.

Moreover, we can use the maximum coverage of a cell to avoid domination checks during the evaluation of $t_x.score$. Assume a query q defined in all dimensions. If c_i is the cell of t_x and if $c_i.max < q.kscore$, then t_x can not be top-k dominant tuple of the query, since $t_x.score \leq c_i.max$. If $c_i.max \geq q.kscore$, we initialize $t_x.score = c_i.min$. Then, we check each tuple of the partially dominated cells, if it is dominated by t_x. We can stop the domination checks if the remaining tuples for examination plus the current $t_x.score$ is smaller than $q.kscore$. In the case where the query has a subset of dimensions q.**D** $\subset$ **D**, we use $p(c_i)$ instead of c_i itself.

We can now proceed with the description of the insertion procedure. Tuple deletions are handled similarly. Initially, the new tuple t_n is inserted in the appropriate cell c_i. The maximum and minimum coverage of the cells that dominate c_i are increased by one, while the maximum coverage of the cells that partially dominate c_i is increased by one. Then, for each active query the following steps are taken:
1) First, we check if there are tuples of top-k of the query that dominate the new tuple t_n and we update their $score$ values accordingly. Then, we rearrange q.**top** and q.**score** in order to keep sorted the q.**score**. Next, we check if the new tuple t_n is top-k of the query. If $q.kscore$ is increased by one, then it is not possible to find another tuple that dominates more or equal than $q.kscore$ tuples and therefore the update is terminated. Notice that the insertion of a new tuple can cause the increase of a $score$ value only by one. If $q.kscore$ is not increased, then it is possible to find tuples (candidate tuples) that dominate exactly $q.kscore$ tuples.
2) Next, we determine all cells containing candidate tuples. These cells are the ones that dominate or partially dominate c_i, since tuples of these cells, it is possible to dominate t_n and therefore their $score$ values may have increased by one. For each such cell c, the corresponding projection $p(c)$ is determined. If $p(c).max < q.kscore$, then the cell is discarded. Otherwise, we compute the partially dominated cells of $p(c)$ and we apply the procedure of the computation of $score$ for each tuple t of $p(c)$, as we described above.

2.2 Adaptive Grid

The advantage of the regular grid is that insertions and deletions are processed efficiently. Unfortunately, if the data distribution in a dimension deviates from uniformity then in high dimensionality most of the cells will be empty and therefore, the regular grid can not be used efficiently.

To overcome this disadvantage and simultaneously keep the reorganization cost low, we propose the *adaptive grid*. The adaptive grid separates equally the tuples in each dimension. Figure 3(a) shows an example of an adaptive grid. The number of tuples is 12 and the number of cells of each dimension is 4. Thus, we want each column of a dimension to have 3 tuples. To obey this rule, dimension d_1 is reorganized. The dashed line indicates the new bound of the column. The reorganization procedure is invoked when tuples are not well separated in a dimension and it is applied only in this dimension without affecting the others. We explain the procedure by means of an example. Assume that a new tuple t_{13} is inserted into the grid of Figure 3(a) and

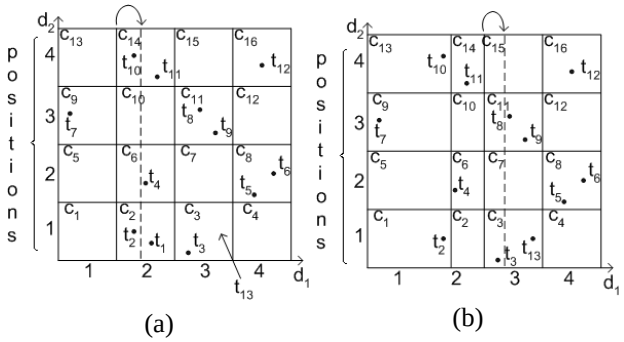


Figure 3. Adaptive grid example.

tuple t_1 is deleted. Then, the reorganization procedure is invoked for dimension d_1. We can choose the second or the fourth column of d_1 to move tuples. The column with the smaller number of tuples is chosen. Then, the chosen column is checked and if it violates the maximum number of tuples, the procedure continues. To avoid endless loops, we stop the procedure if we choose again the column from which the procedure started. Figure 3(b) illustrates a reorganization example. Notice that this is a heuristic, thus it is possible that the rule of equally separated tuples in a dimension may be violated. However, this heuristic performs well in practice.

In order to avoid repetitive reorganizations of the adaptive grid, we use the parameter rf (reorganization factor) which is a bound of the number of tuples in each column of a dimension. For example, if the number of tuples T is 1000, the number of cells of a dimension is 5, and $rf = 0.05$, the reorganization procedure is applied when the number of tuples of a column of this dimension is more than $200 + 200 \cdot 0.05 = 210$ or less than $200 - 200 \cdot 0.05 = 190$.

2.3 Optimizations

The most time consuming part of the proposed method is the computation of the $score$ value of each candidate tuple. The following optimizations can be applied towards more efficient processing:

Traverse order of the cells. Cells that have low positions in dimensions (e.g., cell c_1), contain candidate tuples that have better possibility to be part of a top-k result. Thus, we want to examine these cells earlier than others. To determine the traverse order of the cells, we can use the positions of the cells in each dimension. The sum S of the positions of a cell specifies the traverse order. A cell with lower S must be examined earlier from a cell with higher S.

Discard cells dominated by other cells. Assume cells c_i and c_j that contain some candidate tuples. Moreover, assume that c_i dominates c_j, thus c_i will be examined earlier than c_j due to the previous optimization. If c_i does not contain any top-k tuple then we can discard c_j since each tuple in c_j dominates less tuples than the tuples of c_i.

Skyline set. Since we are interested in tuples that dominate exactly $q.kscore$ tuples and it is not possible to find tuples that dominate more and not be in the top-k dominant of a query, it is safe to discard tuples of a cell that are dominated by other tuples of this cell, i.e., we examine only the tuples that belong to the skyline of this cell.

Minimum domination check. To compute $t_x.score$ of a candidate tuple t_x of c_i, we check how many tuples of the partially dominated cells, are dominated by t_x. It is sufficient to apply domination checks in a subset of dimensions of the query. We clarify this by means of an example. Assume a query q with $q.\mathbf{D} = \{d_1, d_2\}$ and that t_4 is a candidate tuple (Figure 3(a)). Tuples that belong to c_6 should be checked in all the dimensions of the query. Tuples that belong to c_7 and c_8 should be checked only in d_2 and tuples of c_{10} and c_{14} should be checked only in d_1. The minimum domination check reduces the number of the examined dimensions up to $q.D$-1 dimensions.

Upper bound of a cell. It is evident, that cells in low positions of dimensions will be examined in almost all updates. It is possible to use an extra parameter, the *upper bound*, of each cell to further improve the possibility that a cell will be pruned in other queries of the same or subsequent updates. For each candidate tuple t_x that it is not discarded from the above optimizations, we compute $t_x.score$ and we stop if $t_x.score$ can not exceed $q.kscore$. The maximum possible value of $t_x.score$ is its current value plus the number of remaining tuples of the partially dominated cells. For each cell c_i, we keep $max\{t_x.score+$ number of remaining tuples$\}$, $\forall t_x \in c_i$. Then, we can assign this value to the upper bound of the cell if it is lower than the current upper bound. The upper bound of a cell can be used instead of the maximum coverage.

3 Performance Evaluation

In this section, we report the experimental results of our study. We use the abbreviations RG and AD for the regular and the adaptive grid without optimizations while the abbreviations RGO and ADO are used for the corresponding methods with optimizations enabled. All methods are implemented in C++ and the experiments have been conducted on a Pentium IV at 3.6GHz, with 1GB RAM. Two data sets have been used: (a) Forest Cover (FC) (http://kdd.ics.uci.edu) which contains 581,012 records in 10 dimensions and (b) a synthetic anti-correlated data set (AN), generated by using the process described in [1].

In the experiments, we have used 10 dimensions both in real and synthetic data sets. The default values for the parameters (if not otherwise specified) are: the number of tuples is 100,000, the parameter $q.k$ of each query q lies in the range [1,100]. The number of dimensions $q.D$, the set of the dimensions $q.\mathbf{D}$ and the parameter $q.k$ of the queries are generated uniformly. In the diagrams below, the response time per update is given. An update, according to the sliding window paradigm, corresponds to the insertion of the new tuple and the deletion of the most obsolete one.

First, we study the effect of the grid size. We use only one query to evaluate the methods. The number of cells per dimension varies between 2 and 5, corresponding to a grid of 2^{10} up to 5^{10} cells. Figure 4 illustrates the results for the anti-correlated data set. Adaptive and regular grids

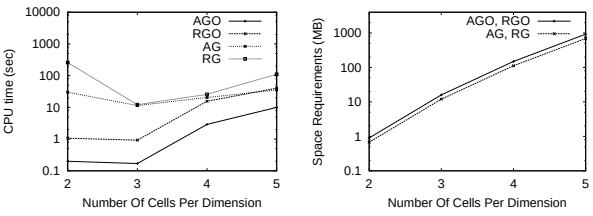


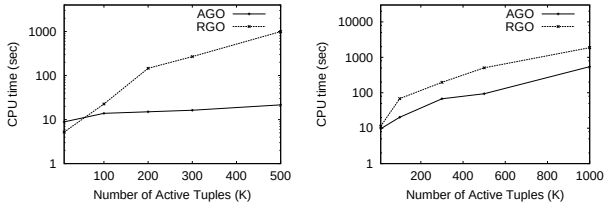
Figure 4. CPU time and space requirements vs. number of cells per dimension

have the same space requirements, thus there are only two curves in Figure 4 (right). The algorithms AGO and RGO have more space requirements than AG and RG due to the optimizations used. A grid of 3^{10} cells is the best choice for all algorithms regarding CPU time and space requirements. We use three cells per dimension for all experiments. The results for the real data set are similar and are omitted. Moreover, AGO and RGO outperform both AG and RG, especially when the number of cells is low. This gap arises due to the optimizations, since AGO and AG have the same pruning power (similarly for RGO and RG). The remaining experiments follow the same trend and therefore, AG and RG are omitted.

The next experiment studies the performance of AGO and RGO with respect to the number of active tuples T. The number of queries Q is set to 50. All queries are uniformly generated. Figure 5 shows the results for the two data sets. FC and AND use up to 0.5M and 1M active tuples respectively. The CPU time is given per update for all queries including the cost of grid update. AGO outperforms RGO, when the number of tuples is high especially in the real data set. The CPU time of RGO increases rapidly because can not prune cells of candidate tuples due to the poor segregation of tuples.

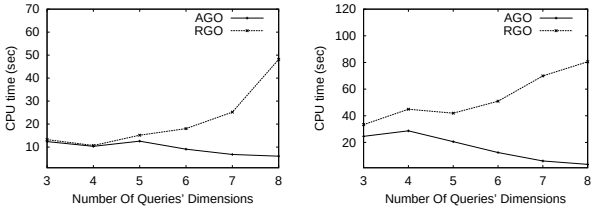
Next, we study the performance of the proposed methods in subspaces. Thus, all queries have the same number of dimensions, but the dimensions sets are different. The number of queries Q is 36. Figure 6 illustrates the results. As expected, the CPU time of AGO decreases as the number of queries' dimensions increases. RGO has different behavior and this happens because its pruning power is lower than that of AGO. As the dimensionality of queries increases each domination check is more expensive and thus, RGO is worst.

Finally, we measure the effect of the upper bound optimization. We run all queries together and we repeat the experiment by running each query separately for AGO. The abbreviation AQ_AGO (All Queries - AGO) and SQ_AGO (Separate Queries - AGO) are used respectively. The results are depicted in Figure 7. As the number of queries increases and therefore the overlapping between the subspaces of queries increases too, the gain due to the upper bound optimization becomes more significant. Notice that, queries' dimensions are generated uniformly. This is an indication that the upper bound will affect more the performance in a realistic application, where some dimensions are usually more preferable than others.



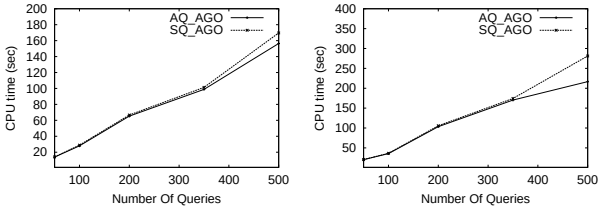
(a) FC (b) AN

Figure 5. CPU time vs. number of tuples



(a) FC (b) AN

Figure 6. CPU time vs. number of queries' dimensions



(a) FC (b) AN

Figure 7. Effect of upper bound optimization

4 Conclusions

The results of our study indicate that continuous processing of subspace top-k dominating queries can be efficiently performed, by organizing the active tuples (objects) in a simple grid-based indexing scheme which offers fast search and update operations. Both regular and irregular grids have been considered. In addition to the basic query processing mechanism, several optimizations have been proposed to increase efficiency. Performance evaluation results based on real-life and synthetic data sets have demonstrated that the irregular grid scheme enhanced by the proposed optimizations yields the best performance.

References

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

[2] Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting Stars: The k Most Representative Skyline Operator. *Proc. of ICDE*, 2007.

[3] Mouratidis, K., Bakiras, S., Papadias, D.: Continuous Monitoring of Top-k Queries over Sliding Windows. *Proc. of ACM SIGMOD*, pp.635-646, 2006.

[4] Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive Skyline Computation in Database Systems. *ACM TODS*, Vol.30, No.1, 2005.

[5] Papadopoulos, A.N., Lyritsis, A., Nanopoulos, A., Manolopoulos, Y.: Domination Mining and Querying. *Proc. of DaWaK*, Regensburg, Germany, 2007.

[6] Yiu, M.L., Mamoulis, N.: Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data. *Proc. of VLDB*, pp.483-494, 2007.