View Usability and Safety for the Answering of Top-*k* Queries via Materialized Views

Eftychia Baikousi Panos Vassiliadis
Dept. of Computer Science, Univ. of Ioannina
Ioannina, 45110 Hellas
{ebaikou, pvassil}@cs.uoi.gr

ABSTRACT
In this paper, we investigate the problem of answering top-*k* queries via materialized views. We provide theoretical guarantees for the adequacy of a view to answer a top-*k* query, along with algorithmic techniques to compute the query via a view when this is possible. We explore the problem of answering a query via a combination of more than one view and show that it is impossible to improve our theoretical guarantees for the answering of a query via a combination of views. Finally, we experimentally assess our approach for its effectiveness and efficiency.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems – *query processing*

General Terms
Algorithms, Theory

Keywords
Top-k queries, materialized views

1. INTRODUCTION
Business Intelligence is extending more and more the palette of tools that the analyst is using. Apart from the traditional reporting and OLAP operations, dashboards and automated alerts are presenting analysts with relatively new and important data. To avoid the overwhelming amount of available information, analysts need tools that help them to focus their attention to few pieces of information of high importance.

To achieve this focused presentation of important, personalized data, BI tools need to allow the user to specify a profile of preferences that *rank* incoming information and *constrain* the result in order to reduce it to few valuable records. This kind of process is known in the database literature under the name of top-*k* querying. The *top-k querying problem* concerns the retrieval of the top-*k* results of a ranked query over a database. Specifically, given a relation R (tid, A_1, A_2,...A_m), a query Q over R retrieves the top-*k* tuples from R having the k highest values according to a scoring function f that accompanies Q. Typically, f is a monotone ranking function of the form $score = a_1*A_1+\ldots+a_n*A_n$. In this paper, we focus on a specific variant of the problem that concerns the *exploitation of top-k materialized views*: assuming several top-*k* queries over the underlying data, is it possible to cache them as materialized views and improve the efficiency of the querying process by answering the queries via these materialized views?

In a real case study, every morning, the high-level executives of a large telecommunications company want to see in their PDA's reports exported from the warehouse with the findings coming from the data after the last night's refreshment. It would be nice for an analyst from the sales department to be able to see a report with the top 10 regions in terms of earnings, ranked by (a) the difference of today's and yesterday's outgoing traffic and (b) the budget spent for advertisements in the local press for urban regions. The combination of these two criteria can be expressed via a score function like e.g., $score_1 = 0.6*diff_{\text{traffic}} + 0.4*budget$. Another analyst from the advertisement might ask a similar report either with different weights, or by different number of results (e.g., top 5), or by completely ignoring the traffic aspect. Is it possible to answer all these top-*k* queries fast, in an on-line fashion, without resorting to the large fact tables or data marts?

Related work has extensively dealt with the problem of efficiently computing the top-*k* results of a query. The first algorithms that occurred in this context are FA [2], [3] and TA [4], with various extensions that followed them for specific contexts (e.g., parallel or distributed computation, etc). In recent years, in an attempt to achieve improved performance, researchers solve the problem of answering top-*k* queries via materialized views [1], [6], [7]. In this setting, a *materialized top-k view* caches the results of previous top-*k* queries. Then, a new top-*k* query may be answered through these materialized views resulting in better performance than making use only of the base relation from the database. In this paper, *we extend the state of the art and provide theoretical and algorithmic results around the problem of answering top-k queries via materialized views*. Specifically, our contribution can be summarized as follows:

First, *we provide theoretical guarantees for the adequacy of a view to answer a top-k query*. We show that even if the view contains more than *k* tuples, it is possible that the correct answer cannot be provided by the view. We utilize these theoretical results to come up with *a simple algorithm that decides whether a view is suitable to answer a query or not, and computes the answer* to the query via an appropriate view. We also show that the theorem for deciding view adequacy might be too strict in certain cases (thus providing room for further optimizations).

Moreover, we explore the problem of answering a query via a combination of more than one view. Despite the efficiency of using two views instead of one for the answering of a query [1], *we prove that it is impossible to improve our theoretical guarantees for the answering of a query via a combination of views*. Consequently, in the absence of other information, the search space of candidate views for the answering of a top-k query is restricted to single view candidacies as explored by our algorithm.

Roadmap. The structure of this paper is as follows: in Section 2, we review related work. We present our results for the adequacy of a view to answer a top-k query in Section 3, along with our algorithmic results. In Section 4, we discuss our findings for the case where more than one view could be used. In Section 5, we experimentally assess the effectiveness and efficiency of the view usability method, enhanced with our theoretical guarantees. Finally, in Section 6, we summarize our findings and present possibilities for future research.

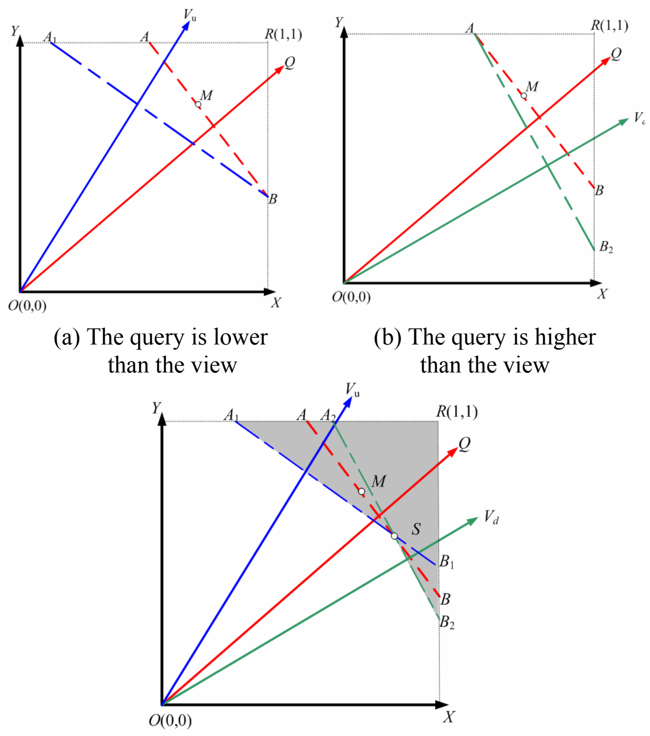
2. RELATED WORK AND BACKGROUND

In this section, we give a brief overview of the basic algorithms that answer a top-k query over a relation R. Also, we give some background information on the technique that we will employ later to come up with our theoretical and algorithmic results.

The first algorithms that dealt with the problem of computing the top-k results of a query that utilizes a monotone function over the combination of a relation's attributes are due to Fagin [2], [3]. This first algorithms (a.k.a FA algorithm) where later complemented by the highly cited TA algorithm [4], [5] that appears to provide better performance. The research community was quick to provide additional means for the computation of the top-k tuples of such a query via the exploitation of materialized views. First, the PREFER system was introduced in [6], [7]. PREFER uses a core algorithm that answers top-k queries using materialized views in a pipelined way. The results of the PREFER research were further expanded in [1], where a linear programming algorithm was introduced for the same purpose.

LPTA [1]. LPTA is implemented through a two-step procedure. Assume a set of materialized views V=(V_1, …V_r) that contain the base views. For a relation R containing an attribute A_i, a base view V_i is a materialized view of the form (id, A_i) ordered over all the tuples of relation R. The first procedure of LPTA is the *SelectViews* algorithm. Algorithm *SelectViews*(V, Q) determines the most efficient subset $U \subseteq V$ over a set of materialized views V, in order to execute a given query Q. The set U is the most efficient subset of V in the sense that it produces the answer to the top-k query most efficiently among all possible subsets of V. The *SelectViews* algorithm is based on a simple greedy heuristic procedure that selects the subset U that has the cheapest cost. Secondly, the LPTA algorithm obtains an answer to Q combining all the information conveyed by the views in U. Each view V(tid, $score_v$) is a set of pairs of the form (tuple identifier, score of that tuple) using the view's scoring function. The main idea of LPTA is based on solving a linear programming problem. The stopping condition of the algorithms holds when the solution of the linear program is at least equal to the minimum value of the top-k buffer. In case the set of views U is equal to the set of base views then LPTA becomes the TA algorithm.

The key intuition of the LPTA algorithm can be visualized through a geometric representation. Assume a relation R(id, X, Y) where without loss of generality the domains of X and Y are normalized over the interval [0, 1]. Apart from the base views V_x and V_y, assume two materialized views V_u(id, $Score_1$) and V_d(id, $Score_2$). Scores $Score_1$ and $Score_2$ are defined as linear functions over the attributes of the relation R. In addition, assume a query Q with a linear scoring function as well. The scoring functions of the views and the query can be depicted as lines. In particular, the line of a linear scoring function of the form $w(a \cdot x + y) = score$ is depicted as: $y = a^{-1} \cdot x$. The linear scoring function is depicted as its perpendicular line for the reason that the score of a tuple t(id, x, y) in regards to the scoring function can be found by projecting that point over the corresponding line. In Figure 1a we depict a view V_u and a query Q via the corresponding lines. Assume that the tuple with the k-th largest score according to Q is denoted as M. In addition, AB denotes the line that passes through M and is perpendicular to the line Q. Then, the top-k tuples according to Q belong in the region of the triangle ABR. This is due to the fact that top-k tuples will have a score higher than the score of the k-th tuple. The only possible points that can have a higher score than the point M are contained in the triangle ABR.



(a) The query is lower than the view

(b) The query is higher than the view

(c) Two views for the answering of a query

Figure 1. Visual demonstration of the LPTA technique for query answering top-*k* via views

Assume now we want to answer the query Q by using the tuples stored in a materialized view V. The way LPTA proceeds, is by performing sorted accesses over the tuples of V. In the geometric representation, this can be visualized as sweeping a line perpendicular to the line of the view towards the point O(0, 0). The order of tuples read by LPTA through sorted accesses over V

is identical to the order of the points met by sweeping the line towards O.

In case only V_u is available, the stopping condition for the algorithm is reached when the sweeping line crosses position A_1B. This occurs because, the view should encounter all tuples whose score in respect to Q are at least equal to the score of the point B. Remember that points M and B have the same score in regards to Q and therefore, the region below the line A_1B does not contain any tuples with score greater than the score of M. Similarly, in case only view V_d is available, the stopping condition is reached when the sweeping line crosses position AB_2. In case both views V_u and V_d are available, the stopping condition is reached when the sweeping lines intersect in a point that lies on the line AB where in Figure 1c is denoted as S.

In the first case, where only V_u is used for answering Q, the number of sorted accesses performed through LPTA is the number of points that belong in the region of the triangle A_1BR. Correspondingly, if only V_d is used, the number of points that belong in the region of the triangle AB_2R is the number of sorted accesses LPTA will perform.

The best choice of the set of views that will answer Q depends upon the number of points that will be accessed, since the points accessed is identical to the number of sorted accesses LPTA will perform. Assume that the number of tuples visited when only V_u is used (i.e., the number of points that belong in the triangle A_1BR) is T_1. The number of tuples visited when only V_d is used (i.e., the number of points that belong in the triangle AB_2R) is denoted as T_2. The number of tuples visited when both views V_u and V_d are used (i.e., the number of points in the region A_1SB_2R which is the shaded area in Figure 1c) is denoted as T_3. Then, V_u will be preferred in case T_1 is less than T_2 and less than T_3. Respectively, view V_d will be preferred when T_2 is less than T_1 and less than T_3. Finally, both views would be preferred in case T_3 is less than T_1 and T_2.

Comparison to Related work. LPTA answers a query Q using a suitable number of views, in order to minimize its execution time. [1] have provided the algorithm *SelectViews* that selects a suitable set of views according to the query. In order to do so, they estimate the score of the last tuple (denoted as $topk_{\min}$) in regards to the query Q. The estimation is computed through the usage of histograms for the distribution of the data. The *SelectViews* algorithm is based on this estimation. Therefore, there is no theoretically established guarantee that the selected views will be able to answer the query. In fact, there are two variants of how the set of views are selected. In the first case, views contain all the tuples from relation R ranked according to their scoring function. Since the views contain all the tuples, query Q will definitely be answered because there will not be any missed tuples that should be contained in the top-k answer of Q. However, an error in the estimation of $topk_{\min}$, might lead to a selection of views that is not the best choice in regards to execution time. In the second case, views only contain a portion of the tuples from relation R. Actually, they contain the top-k tuples according to their scoring function. An error in the estimation of $topk_{\min}$ might cause the inability to answer Q. This is because there might be tuples not included in the set of views selected, which however should be part of the top-k answer of Q. In order to overcome this problem, [1] have proposed the set of selected views to always contain the base views V_x and V_y. For a query Q over two attributes namely x and y, V_x is a materialized view of the form (id, x) ordered over all the tuples of relation R. Similarly, V_y is a materialized view of the form (id, y) ordered over all the tuples of relation R. Therefore, even if the selected views apart from V_x and V_y cannot provide an answer to the query Q, then the usage of the base views will guarantee it. Still, despite these heuristics, there are no clear theoretical or experimental results on the limits of the usability of top-k materialized views, or on the efficiency of exploiting them for the answering of queries.

3. ADEQUACY OF A MATERIALIZED VIEW TO ANSWER A QUERY

In this section, we provide theoretical and algorithmic results for answering top-k queries using materialized views. We start with our fundamental result and then proceed to investigate why our basic theorems could prove to be too strict. Finally, we present a simple algorithm for deciding the usability of a view for a top-k query.

3.1 Problem formulation

Assume a relation $R(ID, X, Y, \ldots)$ and a materialized view $V(ID, X, Y, s)$, with the score s being defined as $s=w(a\cdot x+y)$ and w, a being positive parameters. Following the setting of [1], this equation is characterized by a line $y=a^{-1}\cdot x$. Assume also the query $Q(ID, X, Y, s_Q)$ with the score s_Q being defined as $s_Q=w_Q(a_Q\cdot x+y)$ and w_Q, a_Q being positive parameters. Again, this equation is characterized by a line $y = a_Q^{-1} \cdot x$.

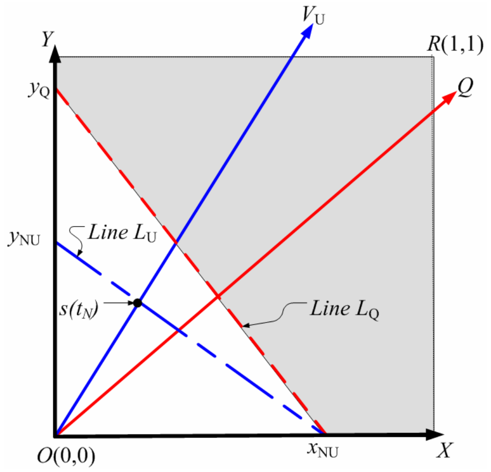


Figure 2. Answering a query Q via a view V_U when the view is "higher" than the query

Assume that the extent of V has n tuples and the query Q requests $k \leq n$ tuples. The question is whether it is possible to answer Q using *only* the tuples materialized in V. We will explore the problem based on its diagrammatic representation and we will discern two cases: in the first case, the line of the view is higher than the one of the query, in the second case, the reverse holds.

3.2 The case when the view is "higher" than the query

In this case (Fig. 2), we assume that $a_Q^{-1} \leq a^{-1}$ (which means that V is drawn "higher" than Q in their graphical representation). We will employ the subscript U for the entire notation concerning view V and refer to it as $V_U(ID, X, Y, s_U)$, with the score s_U being defined as $s_U=w_U(a_U \cdot x+y)$.

Let t_n be the n-th tuple materialized in V_U. Assume that t_n has a score $s(t_n)$. Let L_U: $x_{NU}y_{NU}$ be the line perpendicular to the line of V_U passing from point $s(t_n)$ (with x_{NU}, y_{NU} being the points were it meets the axes X, Y). The area above the line L_U contains the top-n tuples with respect to V_U. Now, take the line L_Q: $x_{NU}y_Q$, which is perpendicular to Q and starts at the point x_{NU}. This area contains points that belong both to Q and V_U (which we call *safe* area).

Lemma 1. It is possible that V_U contains more than k tuples but misses the answer to Q.

Proof. Assume a tuple t of R (Figure 3, near the X-axis) that (a) does not belong to V_U and (b) should be part of Q's top-k answer set. In this case, since t does not belong to V_U, it is lower than the line L_U. Assume also tuples t_1, t_2 placed as depicted in Figure 3. The scores of these tuples are high enough so that they can be included in the top-n for view V_U (remember that the score of a tuple with respect to a query/view involves projecting the tuple to the line of the query/view). Still, tuple t has a higher score than all of these tuples with respect to query Q (observe that the dotted line which starts from t and is perpendicular to Q produces a higher score than the respective line for t_2). Observe that this situation includes the tuple t_n which is the n-th tuple of V_U. Therefore, V_U is insufficient to answer Q. ■

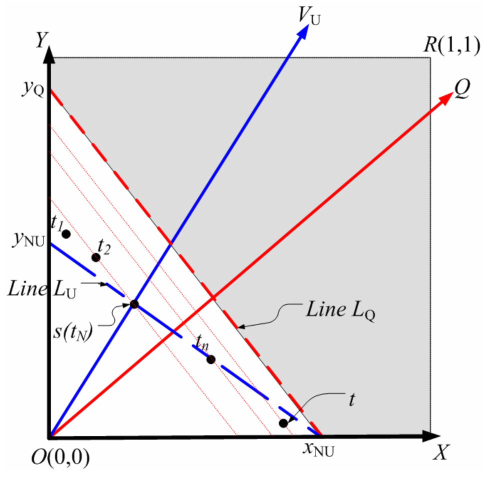


Figure 3. Example of why a view *V* is not always reliable for answering a query *Q*

Theorem 1. V_U can answer Q if the area above line L_Q contains at least k points.

Proof. We will prove the theorem by contradiction. Assume a tuple t of R (Figure 3) that (a) does not belong to V_U and (b) should be part of Q's top-k answer set. In this case, since t does not belong to V_U, it is lower than the line L_U. Still, L_U is always lower than L_Q, therefore, the projection of t over line Q will also be lower than L_Q. If the shaded area beyond L_Q has more than k points, these k points all have scores (projections to line Q) higher than t, with respect to Q, which cannot be true, since we assume that t belongs to the top-k answer set of Q.■

It is interesting to observe that (a) the inverse of Theorem 1 does not always hold, and (b) how can we decide that a point belongs to the safe area.

3.3 Strictness of the suitability theorem

It is not possible to infer the inverse of Theorem 1. Even if the shaded area of line L_Q does not contain k tuples it would still be possible to answer Q with tuples that belong to V_U if a critical area below the line V_U does not contain any tuples. For example, assume the case where tuple t was not present in R, no tuple belongs to the shaded area and the query Q asked for top-3 tuples. Then tuples t_1, t_2, t_n can answer Q since there are not other tuples below line L_U. Still, the main problem is that we need to refer to R (or to some sketch of it) to find whether such tuples lying below L_U exist or not. In fact, it is not even necessary to search the whole area below L_U, but rather a specific subset of it. In our example, it is sufficient to check whether the area of the triangle ($x_{NU}x_1p_1$) contains any tuples or not. The following theorem formalizes the conditions under which a view can answer a query even if its safe area is insufficient.

Theorem 2. It is possible that V_U can answer Q even if there are less than k tuples in the safe area. For this to hold, it is necessary that the area defined by the line L_U, the X-axis and the line that produces the lowest possible score for Q from the tuples of V_U is void of tuples.

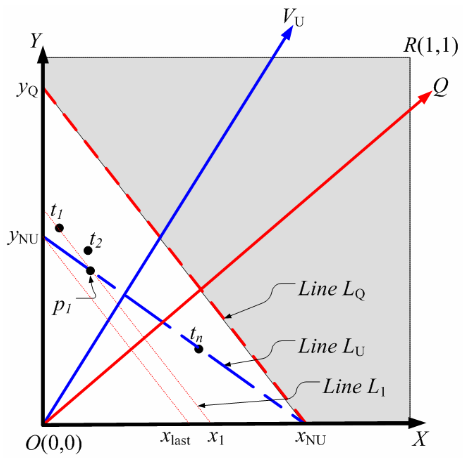


Figure 4. At least *k* points in the safe area of a view *V* make it reliable for answering a query *Q*

Proof. The point x_1 is the point that meets the X-axis and belongs to line L_1 that corresponds to the tuple in V_U with the lowest score with respect to Q (here, in the example of Figure 4, tuple t_1). The point p_1 is the point where this line meets L_U. In other words, we need to find the line that produces the lowest score for Q, for all the tuples in V_U. If the triangle defined by the X-axis, L_U and L_1 has no points, then the points within V_U are the ones producing the lowest possible scores for Q. So, if V_U contains more than k points, it can answer Q.■

3.4 Computation of offsets and safe areas

If one does not want to go through the computation of Q's score for all the tuples of V_U, then another safe criterion would be to use x_{last}, which is the point of the X-axis that corresponds to the line that gives the score for y_{NU} with respect to Q.

In any case, this property can be used if one is interested in approximate results (in fact, the smaller the area of the triangle, the higher the possibility that V_U can answer the query Q).

Moreover, sketches of the data distribution in R can also help in deciding whether the area is empty or not (and to what extent).

A second technical point has to do with whether a point belongs to the shaded area or not. The line L_Q is defined by the equation $y= - a_Q \cdot x + a_Q \cdot x_{NU}$ (easy to check: being perpendicular to line Q, the product of line Q with the line L_Q must be -1; then the offset can easily be computed by putting $y = 0$ for L_Q). Assume a tuple $t_b(x_b, y_b)$. Tuple t_b belongs to the shaded area if $y_b \geq -a_Q \cdot x_b + a_Q \cdot x_{NU}$.

Quite similar to the above point is the computation of the point x_{NU} which is needed for the equation of the line L_Q: assume we know the n-th tuple of V_U, $t_n(x_n, y_n)$. Then, this belongs to the line L_U that is perpendicular to V_U, therefore with an equation $y= -a_U \cdot x + offset$. Since t_n belongs to this line, $offset = y_n + a_U \cdot x_n$. For $y = 0$, we deal with the point x_{NU} and then $offset = a_U \cdot x_{NU}$, i.e., $x_{NU} = a_U^{-1}(y_n + a_U \cdot x_n)$.

3.5 The case when the view is "lower" than the query

In this case, we assume that $a_Q^{-1} \geq a^{-1}$ (which means that V is drawn "lower" than Q in their graphical representation). We will employ the subscript D for all the notation concerning view V and refer to it as $V_D(ID, X, Y, s_D)$, with the score s_D being defined as $s_D = w_D(a_D \cdot x + y)$.

Similarly to the previous case, we can prove that (a) it is possible for view V_D to omit tuples that should belong to the extent of Q and (b) there is a safe region that can guarantee that Q can be answered solely by V_D. Again, we will employ the line (x_{ND} y_{ND}) that passes from the n-th point of V_D and gives its score (i.e., it is perpendicular to the line of V_D). We use point y_{ND} this time and take the line L_Q: y_{ND} x_Q that is perpendicular to the line Q. The line L_Q is defined by the equation $y = -a_Q \cdot x + y_{ND}$ and a tuple t_b (x_b, y_b) belongs to the safe shaded area above the line L_Q if $y_b \geq -a_Q \cdot x_b + y_{ND}$.

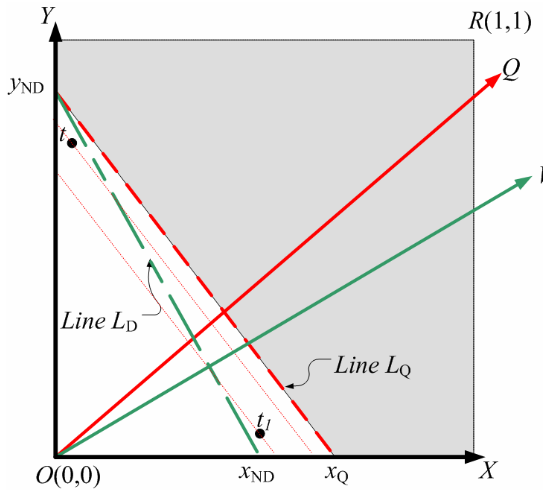


Figure 5. The case where the view is "under" the query

3.6 Special Cases

In the above we have assumed that the scoring functions of the views and the query are in the form of $w(a \cdot x + y)= s$. However, the scoring function of a view or a query can be of the form score $s = x$ or $s = y$. In this section, we describe these special cases.

(i) Assume a view with a scoring function of the form $s_V = y_V$ (i.e., the attribute x does not play any role in the computation of a tuple's score). In such a case (Fig. 6), the line that is perpendicular to V and passes through the last tuple of the view $t_n(x_n, y_n)$, is of the form $L_V : y = y_n$. In addition, assume a query Q with scoring function $w_Q(a_Q \cdot x + y)= s_Q$. In order to compute the safe area in which V can answer Q, we need to know the active domain of the attributes X and Y. Assume that the active domains of attributes X and Y are $X\in[x_{min}, x_{max}]$ and $Y\in [y_{min}, y_{max}]$. Then, the safe area is above line L_Q that is defined as the line that is perpendicular to Q and passes through the point p (x_{max}, y_n).

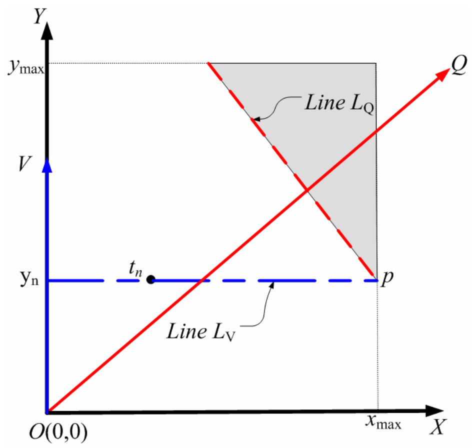


Figure 6. Special case where V is of the form $s_V = y$

An even more extreme case is when both the view and the query ignore attribute x in their scoring function (i.e., both $a_V = a_Q = 0$). In this case, both V and Q are found over axis Y. Then, V can answer Q when it contains more tuples than what Q requests. This is due to the fact that in such a case the scoring function of V is proportional to the scoring function of Q.

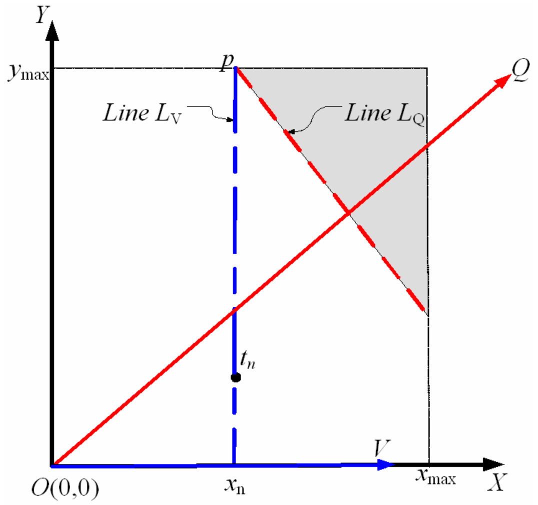


Figure 7. Special case where V is of the form $s_V = x$

An intriguing situation arises when view V is found over the Y-axis and the query Q is found over axis X. In other words, the view score s_V is defined as $s_V= y$ and the query score is defined as

$s_Q = x$. In this case, there is no guarantee that V can answer Q. Assume the case where there exist tuples with very high X values and very low Y values; then these tuples are the top-k tuples of the query; still due to their low Y values they are outside the safe area border and not part of the view. Therefore, it is obligatory to consider the full space as the safe area.

```
Algorithm Test 2DView Suitability
Input:   A materialized view V(ID, X, Y, $s_U$)$^n$,
         with its equation s = w (a · x + y) and its n tuples,
         a Q(ID, X, Y, $s_Q$)$^k$, $s_Q = w_Q (a_Q \cdot x + y)$, $k \leq n$,
Output: a decision on whether Q can be answered by V along
         with the population of V
Variables: a counter to count how many tuples V has inside the
         safe area of Q
Begin
  1.  Let $t_n$ be the n-th tuple of V, $t_n(x_n,y_n)$=V[n].
  2.  if ($\alpha_Q^{-1} \leq \alpha^{-1}$){
  3.        compute point $x_{NU}$: $x_{NU} = a^{-1} (y_n + a \cdot x_n)$
  4.        define line $L_Q$ as $y = -\alpha_Q \cdot x + \alpha_Q \cdot x_{NU}$
  5.  }
  6.  else{
  7.        compute point $y_{ND}$: $y_{ND} = y_n + a \cdot x_n$
  8.        define line $L_Q$ as $y = -\alpha_Q \cdot x + y_{ND}$
  9.  }
  10. for all tuples of V {
  11.       compute $s_Q$(V[i])
  12.       if ($s_Q$(V[i]) belongs above line $L_Q$) counter++ ;
  13. }
  14. if (counter $\geq$ k ) return(true);
  15. else return(false);
End.
```

Figure 8. Algorithm Test 2DView Suitability

(ii) Assume a view with a scoring function of the form $s_V = x_V$ (Fig. 7). In such a case, the line that is perpendicular to V and passes through the last tuple $t_n(x_n, y_n)$ materialized, is of the form $L_V : x = x_n$. In addition, assume a query Q with scoring function $w_Q(a_Q \cdot x + y)= s_Q$. In order, to compute the safe area in which V can answer Q we need to know the active domain of the attributes X and Y. Assume that the active domains of attributes X and Y are $X \in [x_{min}, x_{max}]$ and $Y \in [y_{min}, y_{max}]$. Then, the safe area is above line L_Q. L_Q is defined as the line that is perpendicular to Q and passes through the point p (x_n, y_{max}).

Similarly to the previous case, we can encounter two extreme sub cases. The first of these cases concerns the situation where the scoring function of the query has the same slope with the query. Then, V can answer Q when it contains more tuples than what Q requests for. This is because in such a case the scoring function of V is proportional to the scoring function of Q. The second of these cases, concerns the situation where the scoring function of the query has the parameter $a_Q = 0$: again, there is no guarantee that V can answer Q.

3.7 Algorithmic Results

Now, we are ready to give an algorithm (Fig. 8) that decides whether Q can be answered by V and populates V if the test is positive. As Fig. 9 indicates, the complexity of the algorithm depends on the number of tuples stored in the materialized view (i.e., the number of iterations for the *for loop* in Fig. 8).

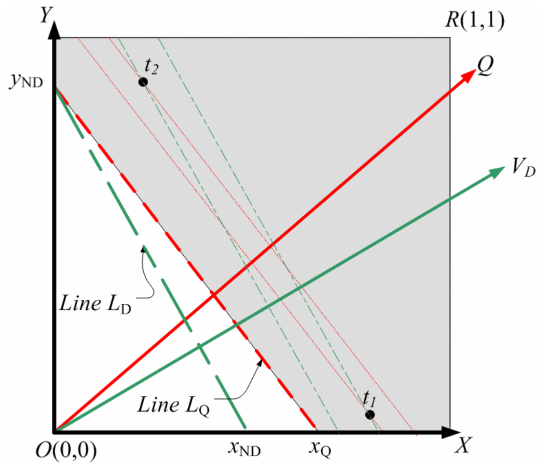


Figure 9. All the safe area should possibly be exhausted for the determination of the top-*k* query tuples

4. WORKING WITH MORE THAN ONE VIEW

[1] have proved that a query can be answered either by a single view, or by a combination of two views whose lines lie on different sides of the query's line. Assume now that for a given query Q, we do not have a single view that can answer the query, but, there exist two views V_U and V_D that lie on different sides of the query's line. Is it possible to use these two views to answer Q without referring to the relation R?

4.1 Safe area containment

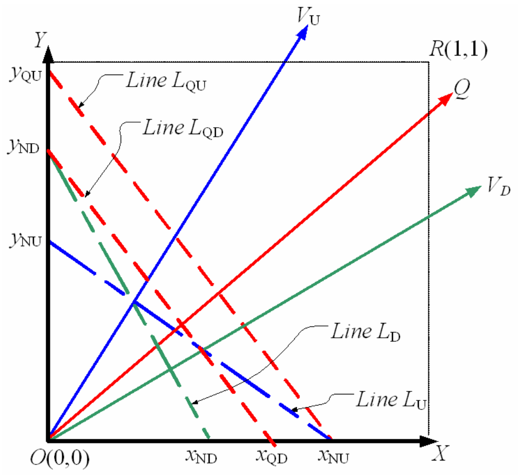


Figure 10. A query *Q* with one view on either of its sides, V_U for the upper side and V_D for the lower side

Observe Figure 10. A query Q is encompassed by two preexisting, materialized views V_U and V_D, the first on the upper and the second on the lower side of Q. Figure 10 also depicts the lines L_U and L_D, which are perpendicular to the respective views and signify their last stored tuple. These lines are also used to draw the lines L_{QU} and L_{QD} which are perpendicular to Q and characterize the safe areas for V_U and V_D respectively.

Theorem 3. Assume two views encompassing a query Q, none of which is safe to be used for answering the query by itself. It is impossible to safely guarantee the answering of the query by the combined usage of the two views.

Proof. Since lines L_{QU} and L_{QD} are both perpendicular to Q, the safe area of one view is encompassed in the safe area of the other view. Since neither view is safe for the answering of the query, it follows that the union of their safe areas is insufficient, too. ■

5. EXPERIMENTS

In this section, we report on the experimental assessment of the usage of materialized views to answer top-*k* queries.

5.1 Experimental Methodology

Our experimental study has been conducted towards assuring the following two goals:

1. *Effectiveness*. The first desideratum of the experimental study has been the verification of the hypothesis that the proposed theoretical results can actually produce a significant number of views that can be employed to answer a top-*k* query.
2. *Efficiency*. The second desideratum of the experimental study has been the testing of the hypothesis that the answering of top-*k* queries via materialized views can indeed improve the performance of query answering at a significant factor.

We have implemented our view usability method and use the only method that can guarantee view usability correctness (i.e., TA) as an opponent. We do not use auxiliary structures in our experiments (e.g., sketches of the non-covered area of a materialized view, or any other indexes). All our experiments involve a relation $R(tid,X,Y)$. All the queries were fully answered and then used as materialized views for the subsequent queries.

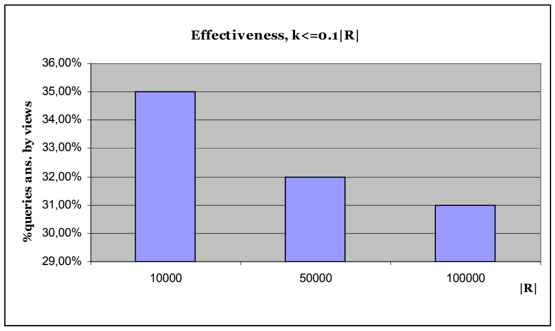


Figure 11. Percentage of views used for 100 queries

We have generated random data sets of different sizes. We generate a sequence of queries with random coefficients and result size (k). Each query's result is cached as a materialized view; so, every query tests all its previous queries as candidates. The important parameters that we have experimented with are: (a) the relation size $|R|$, (b) the number of queries asked $|Q|$ (practically testing how the method works as time passes and more views get to be materialized) and, (c) the range of the requested tuples k as compared to the underlying database size $|R|/k$. The values that we have worked are listed in the following table.

In all our experiments, we have used a server with 1GB memory and a Core 2 CPU at 2.13 GHz. All the implementations were made using BerkeleyDB and its C API.

Table 1. Experimental parameters

Size of source table R (tuples)	$\|R\|$	$1x10^4$, $5x10^4$, $1x10^5$
Size of mat. view (tuples)	k	10, 50, 100, 500, 1000
Number of queries asked	$\|Q\|$	100, 1000

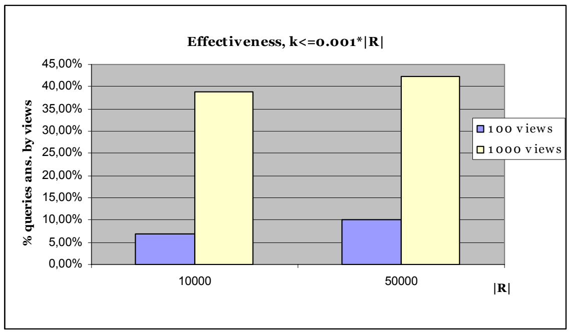


Figure 12. Percentage of views used for different time spans (numbers of posed queries)

5.2 Effectiveness

The effectiveness of the method is depicted in Figures 11 and 12. Figure 11 shows that the effectiveness of the method is quite stable and ranges around 30%-35% for different data sizes. It is also interesting to observe Figure 12, where we use different time spans and different ranges for k to observe the behavior of our method. This is practically achieved by issuing a larger number of queries (i.e., 1000 instead of 100 queries).

The first observation when comparing the two figures concerns the difference in efficiency as we vary the maximum value of k that the queries can take. Observe the dark bars of the two figures, both depicting what happens when 100 queries were issued (so, the only difference is the R/k factor). In Figure 11, the queries are large in size and can request up to 1% of the relation as a result. Frequently, it was the case that a large view that was materialized early in the query series would serve as the answering source for subsequent queries. A second observation from Figure 12, concerns the effectiveness of the method over time. So, in Figure 12, we see what happens as time passes (1000 queries), and we can observe that the effectiveness of the method rises significantly after a while (again to the height of 35%-40%), even for small k's.

5.3 Efficiency

The efficiency of the method over random data is depicted in Figure 13. We vary two parameters, specifically, the relation size, and the maximum possible number that k can take, and we assess the improvement in time when comparing our method with the opponent. The detailed numbers (including total query times) are depicted in Fig. 14.

Interestingly, the time savings present a conflicting case. As the number of stored results rises (dark bars, concerning large k's, up to 1% of the relation size) the savings drop from a 25% improvement to a decrease of 18%. This is clearly due to the size of used memory. As more results are collected in main memory there are two problems: (a) memory allocation becomes slow (in fact, we frequently brought our gnu compiler to its limits) and (ii) it is possible that a certain view will be able to answer several queries due to a very large *k* and a usable slope. Exhausting the safe area for this view might prove too slow for queries with a large *k* (remember that we can be ascertained for the correct result only once we have reached the safe area border). Thus, a caching problem has to be solved based on the grounds of this observation. In any case, if one considers realistic BI scenarios, a top-k query returning 1% is extremely too large; so this is a case in the limit of this technology. On the other hand, the efficiency increases consistently for more reasonable *k*'s of size 0.1% (still large for BI). As the memory allocation is not a problem for this setting, the improvements start from a negligible 1% for small relations and rise up to 24% for a large relation. This is clearly due to the fact that views with appropriate slopes can significantly speed-up the whole process as compared to their full evaluation.

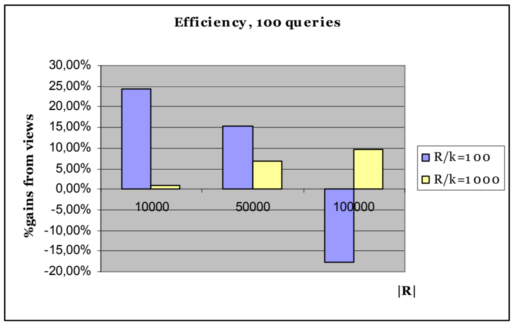


Figure 13. Time savings from the usage of queries for different database sizes and requested results.

6. CONCLUSIONS

In this paper, we have provided theoretical and algorithmic results for the problem of answering top-*k* queries via materialized views. We have provided theoretical guarantees for the adequacy of a view to answer a top-*k* query, along with algorithmic techniques to compute the query via a view when this is possible. Moreover, we have explored the problem of answering a query via a combination of more than one view and showed that despite the efficiency of using two views instead of one for the answering of a query as demonstrated in the related literature, it is impossible to improve our theoretical guarantees for the answering of a query via a combination of views.

Research can follow in different directions. The most prominent ones involve (a) the usage of the appropriate sketches of the involved data to compensate for lack of knowledge on unsafe or not-covered areas of a view with respect to a given query, and (b) the discussion of issues concerning the view caching problem, in order to efficiently accommodate large numbers of view contents that can appropriately serve as subsequent queries within a limited memory budget.

7. REFERENCES

[1] Gautam Das, Dimitrios Gunopulos, Nick Koudas, Dimitris Tsirogiannis. Answering Top-k Queries Using Views. In Proc. of the 32nd VLDB conference, pp. 451-462, Seoul Korea, 2006.

[2] Ronald Fagin. Combining fuzzy information from multiple systems. In Proc. of the 15th ACM Symposium on principles of database systems, pp. 216-226, Montreal Canada, 1996.

[3] Ronald Fagin. Fuzzy queries in multimedia database systems. In Proc. of the 17th ACM Symposium on principles of database systems, pp. 1-10, Seattle USA, 1998.

[4] Ronald Fagin, Amnon Lotem, Moni Naor. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. 66(4), pp. 614-656, 2003.

[5] Ulrich Güntzer, Wolf-Tilo Balke, Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In Proc. of the 26th VLDB conference, pp. 419-428, Cairo Egypt, 2000.

[6] Vagelis Hristidis, Nick Koudas, Yannis Papakonstantinou. PREFER a system for the efficient execution of multi-parametric ranked queries. In Proc. of the ACM Special Interest Group on Management of Data Conference (SIGMOD), pp. 259-270, Santa Barbara USA, 2001.

[7] Vagelis Hristidis, Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. VLDB journal, 13(1), pp. 49-70, 2004.

[8] Surya Nepal, M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In proc. of the 15th International Conference on Data Engineering (ICDE), pp. 22–29, Sydney, Australia, 1999.

R	k	D/k	% views used	Total time without views (sec's)	Total time via views	Total opponent time	% improved
10000	100	100	35,00%	0,351653	0,006241	0,091611	24,28%
10000	10	1000	7,00%	0,074605	0,000011	0,000749	0,99%
50000	500	100	32,00%	4,323892	0,396714	1,06225	15,39%
50000	50	1000	10,00%	1,064684	0,000193	0,072758	6,82%
100000	1000	100	31,00%	12,037897	4,599822	2,458762	-17,79%
100000	100	1000	11,00%	2,682971	0,003201	0,262244	9,66%

Figure 14. Detailed information for the efficiency of the method in time savings.