Theoretical guarantees for query processing of top-k queries over materialized views

Eftychia Baikoussi¹, Panos Vassiliadis¹

¹ Dept. Of Computer Science, Univ. of Ioannina, 45110 Ioannina, 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 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. Finally, we discuss the issue of providing partial results for a query via a materialized view by splitting the range of score into appropriate sub-ranges. This way, different parts of the query answer can be obtained in parallel, by distributing their processing to different servers.

Keywords: top-k views, view usability

1 Introduction

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, A₁, A₂,...Aₘ) and a query Q over R retrieve 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: f : dom(A₁) ×...× dom(Aₘ) → ℜ⁺.

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 [Fagi96], [Fagi98] and TA [FaLN03], 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 [DGKT06], [HrKP01], [HrPa04]. In this setting, results of previous top-k queries are stored in the form of materialized views. 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 build upon the results of previous works 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 also show that the theorem for deciding view adequacy might be too strict in certain cases (thus providing room for further optimizations).

We complement the previously mentioned results 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 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 [DGKT06], we prove that it is impossible to improve our theoretical guarantees for the answering of a query via a combination of views.

Still, this negative result is compensated –to a certain extent- by providing alternative means for parallelizing the answering of a query. We explore the issue of providing partial results of a query via a materialized view by splitting the range of the query score into appropriate sub-ranges. This way, different parts of the query answer can be obtained in parallel.

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 views could be used. In Section 5, we explore the issue of splitting the range of results into subranges. In Section 6, we generalize our results from the case of views with two scoring attributes to the case of views with an arbitrary number of scoring attributes. Finally, in Section 7, 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.

2.1 Algorithms for top-\( k \) queries over databases

In this section, we give a brief overview of the basic algorithms that answer a top-\( k \) query over a relation \( R \).

Fagin’s algorithm (FA) [Fagi96], [Fagi98]. Given a relation \( R (tid, A_1, A_2, \ldots, A_m) \), from which a set of sorted lists \( L=\{(tid, A_i)|tid, A_i\in R\} \) \( \forall A_i \in R \) is formed and a query scoring function \( g(X) \) such that \( g(X) \) is a monotone aggregation function, Fagin’s algorithm \( FA \) retrieves the top-\( k \) tuples of \( R \). This is achieved by a three-step
process. First do sorted access to each of the \( m \) sorted lists, until there are at least \( k \) tuples seen in each of the \( m \) lists. Secondly, for each tuple \( X \) seen, do random accesses to each of the lists to find the \( i \)-th attribute of that tuple, which is \( x_i \). Thirdly for each \( X \) seen, compute its score \( g(X) = g(x_1, x_2, \ldots, x_m) \). The output is the ordered set \( \{(X, g(X)|X \in Y)\} \) where \( Y \) contains the \( k \) tuples with the highest scores.

FA is correct when \( g \) is a monotone aggregation function. This is important in the sense that it assures that all tuples not seen under sorted access do not participate in the top-\( k \) tuples.

**Threshold algorithm (TA)** [FaLN03] [GüBK00] [NeRa99]. FA is not an optimal algorithm whereas, the threshold algorithm is. Similarly to FA, TA can be applied over a database having \( m \) attributes. TA is expressed through a three-step process: First do sorted access in parallel to each of the \( m \) sorted lists. For each tuple \( X \) seen under a list, do random accesses to all the other lists to find the scores \( x_i \) of \( X \). Compute the score \( g(X) = g(x_1, x_2, \ldots, x_m) \) of the tuple \( X \) and remember \( X \) and its score if it is one of the \( k \) highest. Secondly, define the threshold value \( \tau \) as \( g(x_1, x_2, \ldots, x_m) \) where \( x_i \) is the score of the last tuple seen under sorted access to each of the lists. Halt when at least \( k \) tuples have been seen with score at least equal to \( \tau \). The output is then the ordered set \( \{(X, g(X)|X \in Y)\} \) where \( Y \) contains the \( k \) tuples that have been seen with the highest grades. TA is correct when \( g \) is a monotone aggregation function.

[FaLN03] have proved that TA is instance optimal. An algorithm \( B \) is *instance optimal* over a class of algorithms \( A \) and a class of legal inputs \( D \) to the algorithms when \( B \in A \) and if for every \( A \in A \) and for every \( D \in D \), we have \( \text{cost} (B, D) = O (\text{cost} (A, D)) \).

### 2.2 Algorithms for top-k queries over a database and materialized views

**Prefer** [HrKP01], [HrPa04]. PREFER is a system with a core algorithm that answers top-\( k \) queries using materialized views in a pipelined way. PREFER consists of two algorithms called (i) ViewSelection algorithm and (ii) PipelineResults algorithm. The ViewSelection algorithm decides which views should be materialized according to the system’s performance requirements and a given relation. The goal of the PipelineResults algorithm is to rank the tuples of a relation \( R(A_1, \ldots, A_n) \) of \( n \) attributes, according to a query \( q \). The query \( q \) is characterized by a preference vector. A preference vector is of the form \( (w_1, w_2, \ldots, w_n) \) where each coordinate \( w_i \) denotes the preferred weight of the \( i \)-th attribute. Therefore, the scoring function of \( q \) becomes \( \sum_{i=1}^{n} w_i \cdot A_i \). Algorithm PipelineResults employs a views \( R_v(tid, score_v) \) that contains the tuples of \( R \), ranked by another preference vector \( v \). The algorithm computes a prefix \( R_v^1 \) from \( R_v \) that ensures that the first tuple \( t_q^1 \) of the sequence \( R_v \) is in \( R_v^1 \). Then, it computes the second prefix \( R_v^2 \) in order to retrieve the second tuple \( t_q^2 \) and so on until the first \( k \) tuples of the query are retrieved. The key concept of this algorithm is the computation of a watermark value, which works as a stopping condition in each iteration of the PipelineResults algorithm. The watermark value is a
score with respect to the ranking function of the materialized view rather than the query that actually determines how deep in the ranked materialized view we should go in order to output the top result tuple of the query. The watermark value of the first iteration is the maximum value $T^1_{v,q}$ with the property $\forall t \in R, f_q(t) < T^1_{v,q} \Rightarrow f_q(t) < f_q(t^1)$.

**Linear programming adaptation of the threshold algorithm LPTA [DGKT06].**

LPTA is based on the TA algorithm. LPTA is applied on a set of materialized views in order to answer top-$k$ queries. LPTA is implemented through a two-step procedure. Assume a set of materialized views $V = (V_1, \ldots, V_r)$ 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. LPTA starts with an empty top-$k$ buffer and proceeds in the following four steps. Firstly, it does sorted accesses in parallel to each of the views. Secondly, for each tuple $X$ read from a view, random accesses are done on relation $R$ in order to find the scores $x_i$ of $X$. Thirdly, the score $t(X) = t(x_1, x_2, \ldots, x_m)$ of the tuple $X$ in regards to the query $Q$ is computed and the top-$k$ buffer is updated. Fourthly, the stopping condition is checked. In order to check the stopping condition, a linear program is solved. Assume that the last tuple read from each view $V_i$ has score $score_v$ in regards to its scoring function $SF_i$. The objective function of the linear program is the query’s score function. The constraints for the linear program are the inequalities $SF_i \leq score_v$. The stopping condition 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 form the base views $V_x$ and $V_y$, assume two materialized views $V_{x}(id, Score_1)$ and $V_{y}(id, Score_2)$. 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 $w(ax + y) = score$ is depicted as: $y = a^{-1} \cdot x$. Since the line is perpendicular to the scoring function the product of their slopes should be equal to -1. The linear scoring function is depicted as its perpendicular line for the reason that the
score of a tuple \((id, x, y)\) 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 \(ABR\).

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 \((0, 0)\). 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 grayed 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 \(T_2\).
2.3 Comparison to Related work

LPTA answers a query $Q$ using a suitable number of views, in order to minimize its execution time. [DGKT06] 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 \( \text{top}_{k_{\text{min}}} \)) 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 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 \( \text{topk}_{\text{min}} \), 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 \( \text{topk}_{\text{min}} \) 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, [DGKT06] 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.

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 \( R(ID, X, Y, ...) \) 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 [DGKT06], 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 \( y = a_Q^{-1} \cdot x \).
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, 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 $L_U$: $x_{NU}$ be the line vertical 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 $L_Q$: $x_{NU}$ which is vertical 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).
Figure 3. Example of why a view $V$ is not always reliable for answering a query $Q$.

**Lemma.** 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 vertical 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 $Q$. ■

**Theorem.** $V_U$ 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 gray 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 the theorem does not always hold, and (b) how can we decide that a point belongs in the safe area.

### 3.3 Strictness of the suitability theorem

It is not possible to infer the inverse of the theorem. Even if the gray 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 gray area and the query $Q$ asked for top-3 tuples. Then tuples $t_1, t_2, t_3$ can answer $Q$ since there are no 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 $(x_N, x_1, p_1)$ 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.** 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 $L_{Q_i}$, the $X$-axis and the line that produces the lowest possible score for $Q$ from the tuples of $V_U$ is void of tuples.

**Proof.** The point $x_i$ is the point that meets the $X$-axis and belongs to line $L_{Q_i}$ 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 $L_{Q_i}$. 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 \( x_{\text{last}} \), which is the point of the \( X \)-axis that corresponds to the line that gives the score for \( y_{\text{NU}} \) 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 gray area or not. The line \( L_Q \) is defined by the equation \( y = -a_Q \cdot x + a_Q \cdot x_{\text{NU}} \) (easy to check: being vertical 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 gray area if \( y_b \geq -a_Q \cdot x_b + a_Q \cdot x_{\text{NU}} \).

Quite similar to the above point is the computation of the point \( x_{\text{NU}} \) 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 vertical to \( V_U \), therefore with an equation \( y = -a_U \cdot x + \text{offset} \). Since \( t_b \) belongs to this line, \( \text{offset} = y_n + a_U \cdot x_n \). For \( y = 0 \), we deal with the point \( x_{\text{NU}} \) and then \( \text{offset} = a_U \cdot x_{\text{NU}} \), i.e., \( x_{\text{NU}} = a_U^{-1}(y_n + a_U \cdot x_n) \).

### 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 \( s_D = w_D \left( a_D \cdot x + y \right) \).
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 vertical to the line of $V_D$). We use point $y_{ND}$ this time and take the line $L_Q$: $y = -a_Q \cdot x + y_{ND}$ and a tuple $(x_b, y_b)$ belongs to the safe gray area above the line $L_Q$ if $y_b \geq -a_Q \cdot x_b + y_{ND}$.

**Lemma.** It is possible that $V_D$ contains more than $k$ tuples but misses the answer to $Q$.

**Proof.** Similarly to the previous case, assume a tuple $t$, outside the extent of the view, whose projection to the line $Q$ falls higher than a tuple $t_1$ which is part of the view.

**Theorem.** $V_D$ can answer $Q$ if the safe area above line $L_Q$ contains at least $k$ points.

**Proof.** Assume a tuple $t$ of $R$ (Figure 5) that (a) does not belong to $V_D$ and (b) should be part of $Q$’s top-$k$ answer set. In this case, since $t$ does not belong to $V_D$, it is lower than the line $L_D$. Still, if $t$ is lower than line $L_D$, it is also lower than line $L_Q$ and then its projection to line $Q$ cannot fall in the safe area above line $L_Q$. ■
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 \( s = x \) or \( s = y \). In this section, we describe these special cases.

(i) Assume a view with a scoring function of the form \( s_V = y \) (i.e., the attribute \( x \) does not play any role in the computation of a tuple’s score). In such a case, 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 \( s_V = y \)](image)

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 \( Q \).

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.

(ii) Assume a view with a scoring function of the form $s_V = x_V$. 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 $p(x_n, y_{\max})$.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure7.png}
\caption{Special case where $V$ is of the form $s_V = x$}
\end{figure}

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 Summary and Algorithmic Results.
Now, we are ready to give an algorithm that decides whether $Q$ can be answered by $V$ and populates $V$ if the test is positive.
**Algorithm Test 2DView Suitability**

**Input:** A materialized view $V(ID, X, Y, s_0)$\textsuperscript{n}, with its equation $s = w (a \cdot x + y)$ and its $n$ tuples, a $Q(ID, X, Y, s_0)$\textsuperscript{k}, $s_0 = w_0 (a_0 \cdot x + y)$, $k \leq n$.

**Output:** A decision on whether $Q$ can be answered by $V$ along with the population of $V$.

**Begin**

1. Let $t_n$ be the $n$-th tuple of $V$, $t_n(x_n,y_n)=V[n]$.
2. if $(\alpha^{-1} \leq \alpha^{-1})$
3.  compute point $x_{NU}$: $x_{NU} = \alpha^{-1} (y_n + \alpha \cdot x_n)$
4.  define line $L_Q$ as $y = -\alpha_Q \cdot x + \alpha_Q \cdot x_{NU}$
5. }
6. else{
7.  compute point $y_{ND}$: $y_{ND} = y_n + \alpha \cdot x_n$
8.  define line $L_Q$ as $y = -\alpha_Q \cdot x + y_{ND}$
9. }
10. Let $t_k$ be the $k$-th tuple of $V$, $t_k(x_k,y_k)=V[k]$.
11. if $t_k$ belongs to the safe area above Line $L_Q$
12.  for all tuples of the safe area of $V$
13.   compute $s_Q(V[i])$ & add it to $V$’s extent
14. } //can use TA, LPTA, or any other method to compute the top-k view tuples
15. return(true);
16. }
17. else return(false); //and avoid the temptation to test the rest of $V$

**End.**

**Figure 8. Algorithm Test 2DView Suitability**

An interesting point has to do with the computation of the top-$k$ tuples once $V$ has been detected as a viable candidate to answer $Q$. For reasons similar to the aforementioned lemmas, it is not possible to simply take the first $k$ tuples of $V$; in the worst case, all the tuples of the whole safe area must be probed (see Figure 9 for the case where tuple $t_1$ has a better score than tuple $t_2$ with respect to view $V_D$, but tuple $t_2$ has a better score than tuple $t_1$ with respect to the query).
Figure 9. All the safe area should possibly be exhausted for the determination of the top-$k$ query tuples

Remember that we assume that the materialized view is memory resident, so we do not need to resort to unnecessary I/O’s. Here, due to the simplicity of the problem, we perform a brute-force approach; still, it is quite possible to employ the well known algorithms for top-$k$ query processing that provide an earlier stopping condition than the one of Line 12 of algorithm Test 2D View Suitability.

4 Working with more than one views

[DGKT06] 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

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 vertical to the respective views and signify their last stored tuple. These lines are also used to draw the lines $L_{Q_U}$ and $L_{Q_D}$ which are vertical to $Q$ and characterize the safe areas for $V_U$ and $V_D$ respectively.
**Theorem.** 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 vertical 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.

![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](image)

5 Working with more than one views in parallel

The above negative result produces an interesting useful side effect. Assume the case where several materialized views are available; still, instead of being centrally stored, the different views are distributed among different servers. A mediator receives queries and it is responsible for assigning queries to views (or \( R \)) to be answered. It is reasonable to assume that the mediator has some global knowledge for each view’s equation, number of materialized tuples and value of the last tuple. We will also assume that the maximum and minimum values of the active domain of attributes \( X \) and \( Y \) of relation \( R \) are known to the mediator, too. Assume now that a query arrives and we want to parallelize its processing. Is it possible to assign a different part of the query to a different view and then unite the results?

In this section, we will first show that it is feasible to assign a subset of the query answer to a certain view. Since we have knowledge of the active domains of attributes \( X \) and \( Y \), we can estimate the maximum and minimum scores with respect to the query \( Q \). We will show that it is possible to split the range of values for the score and assign a sub-range of scores to specific views.
Theorem. Assume a materialized view $V$ (with a line $V : y = a_U^{-1} \cdot x$) and a query $Q$ (with a line $Q : y = a_Q^{-1} \cdot x$) over the same relation $R$. Assume also that $V$ is safe to answer $Q$ and we are interested in computing only a subset of $Q$, say $Q'$, that includes the tuples whose score falls within the range $[s_{\text{low}}, s_{\text{high}}]$ (with $s_{\text{low}} \leq s_{\text{high}}$ and $s_{\text{low}}$ and $s_{\text{high}}$ the distances of the respective points from the beginning of the axis, with both these points found in the safe area and belonging to line $Q$). $Q'$ can be computed solely from $V$, by including in its result set all the tuples that belong to the area surrounded by the lines $L_l$ and $L_h$, which we call search area, and is defined as follows:

$$L_l : y = -a_Q \cdot x + s_{\text{low}} \cdot \sqrt{a_Q^2 + 1}, \quad L_h : y = -a_Q \cdot x + s_{\text{high}} \cdot \sqrt{a_Q^2 + 1}.$$ 

Proof. Clearly, all tuples belonging to the above area also belong to the safe area of $Q$ over $R$. To compute the lines $L_l$ and $L_h$, we need to locate the coordinates of the points with distance $s_{\text{low}}$ and $s_{\text{high}}$ from the beginning of the axes. For point $p_h (x_h, y_h)$ corresponding to $s_{\text{high}}$, we know that (i) $y_h = a_Q^{-1} \cdot x_h$ and (ii) $x_h^2 + y_h^2 = s_h^2$. This way we can compute the coordinates for the point $p_h (x_h, y_h)$ and respectively, for the point $p_l (x_l, y_l)$. Then, we need to compute the equations for lines $L_l$ and $L_h$. The equation of both lines is of the form $y = -a_Q \cdot x + \text{offset}$, with $\text{offset}$ being unknown (remember that the two lines are parallel to the line $L_Q$ that bounds the safe area). To compute the offset for each line, we need to place the appropriate point in the equation (e.g., for point $p_h (x_h, y_h)$ we have $y_h = -a_Q \cdot x_h + \text{offset}$) and solve the system of equations that also comprises the equation of line $Q$. The solution gives the equations of the theorem.

---

**Figure 11.** The active zone for the range $s_{\text{low}}, s_{\text{high}}$ of query $Q$ within its safe area over view $V_U$. 

---

The theorem is proven by showing that the area defined by the lines $L_l$ and $L_h$ is contained within the safe area of $Q$ and that it includes only the tuples whose score falls within the specified range $[s_{\text{low}}, s_{\text{high}}]$. The proof involves calculating the coordinates of the points corresponding to $s_{\text{low}}$ and $s_{\text{high}}$ and then using these points to determine the equations of the lines $L_l$ and $L_h$. These lines are parallel to the line $L_Q$ that bounds the safe area, and their equations are derived to ensure that only the tuples within the range of interest are included in the result set.
Observe that it is indifferent whether $V$ is on the upper or lower side of $Q$, since we have carefully selected the scores $s_{\text{low}}$ and $s_{\text{high}}$ to be within the safe area.

Having proved the bounds of the search area, we are ready to come up with an algorithm for identifying the tuples of $V$ that belong to the search area. Observe Fig. 11. We need to identify tuples that have a score with respect to $V$’s scoring function within the range $[v_{\text{low}}, v_{\text{high}}]$. Unfortunately, we cannot solely rely on the score bounds of $v_{\text{low}}, v_{\text{high}}$ for this purpose, since it is possible that $V$ contains tuples outside the safe area of $Q$ whose score (with respect to $V$) falls within the range $[v_{\text{low}}, v_{\text{high}}]$.

**Lemma.** Given the values $s_{\text{low}}, s_{\text{high}}$ for the scores of the query $Q$, the range of scores for tuples belonging to $V$, that are candidate for being part of $Q$’s extent too, are:

\[
v_{\text{low}} = s_{\text{low}} \cdot \frac{a_{U}^{2} + 1}{1 + a_{Q} \cdot a_{U}} \cdot \sqrt{a_{U}^{2} + 1}, \quad v_{\text{high}} = s_{\text{high}} \cdot \frac{a_{U}^{2} + 1}{1 + a_{Q} \cdot a_{U}} \cdot \sqrt{a_{U}^{2} + 1}
\]

**Proof.** The point $p_{h}(x_{h}, y_{h})$ falls on the intersection of two lines, $V_U$ and $L_{h}$. Also $x_{h}^{2} + y_{h}^{2} = v_{h}^{2}$. By solving the system of three equations we can compute the score $v_{\text{high}}$. We can compute $v_{\text{low}}$ similarly.

**Theorem.** A tuple $p_{t}(x_{t}, y_{t})$ that belongs to $V$ with score $s_{t}$ (with respect to $V$), qualifies for an answer to $Q$ (with a score $a_{Q} \cdot x_{t} + y_{t}$) if it fulfils the following three conditions:

- $s_{t} \in [v_{\text{low}}, v_{\text{high}}]$ with this range computed via the above lemma,
- $y_{t} \geq -a_{Q} \cdot x_{t} + s_{\text{low}} \cdot \sqrt{a_{U}^{2} + 1}$,
- $y_{t} \geq -a_{Q} \cdot x_{t} + s_{\text{high}} \cdot \sqrt{a_{U}^{2} + 1}$

**Proof.** Obvious.

If $V$ is not sorted over the score of its tuples, then there is no alternative than scanning all its tuples and testing the above conditions. If $V$ is sorted on its score, nevertheless, the algorithm for computing the answer to $Q$ by using the tuples of $V$ is straightforward.
Algorithm Compute Query Extent

**Input**: A materialized view \( V(ID, X, Y, s_U)^n \), with its equation \( s = w(\alpha \cdot x + y) \) and its n tuples (sorted over \( s_U \)), a \( Q(ID, X, Y, s_Q)^k \), \( s_Q = w_Q(\alpha_Q \cdot x + y) \), \( k \leq n \).

**Output**: the computation of \( Q \) via the tuples of \( V \).

**Begin**

1. Compute \( v_{low} \) and \( v_{high} \).
2. Locate the first(last) tuple with score \( v_{low}(v_{high}) \) via binary search.
3. do
   4. Get the next tuple \( t \).
   5. Test the conditions of the above theorem for tuple \( t \).
   6. If \( t \) passes all tests
      7. Compute \( t \)'s score for \( Q \).
      8. Add \( t \) (sorted over \( s_Q \)) to \( Q \)'s extent.
   9. ) until the last (first) tuple with score \( v_{high} \) (\( v_{low} \)) is found.

**End.**

**Figure 12.** Algorithm Compute Query Extent

---

6. Queries and Views with more than two scoring attributes

All the above results can be generalized for an \( N \)-dimensional space. In this section, we will discuss the suitability of views to answer queries when an arbitrary number of scoring attributes is involved, explore special cases and provide a simple algorithm to check the suitability of a view to answer a query.

### 6.1 Fundamental results for the \( n \)-dimensional case

Assume a relation \( R(ID, X_1, X_2, \ldots, X_N) \). All tuples of \( R \) can be represented as points over an \( N \)-dimensional space. In addition, assume a materialized view of the form \( V(ID, X_1, X_2, \ldots, X_N, s) \) with score \( s \) being defined as \( s = w (a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + x_N) \). In an \( N \)-dimensional space, \( V \) can be represented as a line with equations \( L_V: \frac{x_1}{a_1} = \frac{x_2}{a_2} = \ldots = x_N \). The score of any point \( t \) from \( R \) is the measure of the line segment defined by the beginning of the axes (i.e., \( O(0, \ldots, 0) \)) and the projection of this point \( t \) over the line \( L_V \). Assume that the extent of \( V \) has \( n \) tuples. Let \( t_n \) be the \( n \)-th tuple in \( V \) with score \( s(t_n) \). Then, the hyperplane \( P_V \) defined by the equation \( s = w (a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + x_N) \) which is perpendicular to line \( L_V \) and contains \( t_n \), separates the space into two sub-regions. Let one sub-region denoted as \( SR_{low} \) be the one defined
from the hyperplane $P_V$ and towards infinity, whereas the other sub-region $SR_{high}$ is the one defined from the hyperplane $P_V$ and in the opposite direction towards the beginning of the axes. Sub-region $SR_{low}$ contains all the tuples that belong to $V$. Figure 13 depicts the plane $P_V$ in a three-dimensional space and the two sub-regions that it defines. Observe that the plane $P_V$ is denoted as a triangle. This illustrates the visible part of a plane intersecting all three axes when it is observed from the positive sub-axes.

Assume also the query $Q((ID, X_1, X_2, \ldots, X_N, s_Q))$ with the score $s_Q$ being defined as $s_Q = w_Q (a_{1Q} \cdot x_1 + a_{2Q} \cdot x_2 + \ldots + x_N)$. Similarly, $Q$ can be represented as a line with equations $L_Q \frac{x_1}{a_{1Q}} = \frac{x_2}{a_{2Q}} = \ldots = x_N$. In addition, assume that $Q$ requests $k \leq n$ tuples. The question is whether it is possible to answer $Q$ using only the tuples materialized in $V$. Let $x_{1V}, x_{2V}, \ldots, x_{NV}$ denote the points that the hyperplane $P_V$ meets the axes $X_1, X_2, \ldots X_N$ respectively. Now take a hyperplane $P_Q$ that is perpendicular to line $L_Q$. Sweep this plane, always perpendicular to line $L_Q$, in direction from infinity towards the hyperplane $P_V$ until it meets one of the points $x_{1V}, x_{2V}, \ldots, x_{NV}$. The sub-region defined from the hyperplane $P_Q$ towards infinity contains points that belong both in view $V$ and in query $Q$.

**Lemma.** It is possible that $V$ contains more than $k$ tuples but misses the answer to $Q$.

*Proof.* Assume $t_k$ is the $k$-th tuple of $Q$ with score $s_Q(t_k)$. Assume also a tuple $t_1$ of $R$ that (a) does not belong to $V$ and (b) should be part of $Q$’s top-$k$ answer. Then, the following inequalities hold for $t_1$: $s_v(t_1) \leq s_v(t_k)$ and $s_Q(t_1) \leq s_Q(t_k)$. Assume also a tuple $t_2$ that belongs in the sub-region defined between the two hyperplanes $P_Q$ and $P_V$. Therefore the following inequalities hold for $t_2$: $s_v(t_2) \leq s_v(t_k)$ and $s_Q(t_2) \leq s_Q(t_k)$ since hyperplane $P_Q$ lies above the hyperplane $P_V$. By combining the four inequalities
we get the following: $s_v(t_1) \leq s_v(t_0) \leq s_v(t_2)$ and $s_Q(t_2) \leq s_Q(t_k) \leq s_Q(t_1)$. This indicates that the view contains more than $k$ tuples but there are still other tuples (i.e., $t_2$) not belonging to the view that are in the top-$k$ tuples of $Q$. ■

**Theorem.** $V$ can answer $Q$ if the sub-region above the hyperplane $P_Q$ (i.e., the safe area) contains at least $k$ points.

*Proof.** By contradiction. Assume a tuple $t$ of $R$ that (a) does not belong in $V$ and (b) $t$ should be part of $Q$’s top-$k$ answer set. In this case, since $t$ does not belong in $V$, it lies in the sub-region defined by the hyperplane $P_V$ towards the beginning of the axes. However, the hyperplane $P_V$ is always below the hyperplane $P_Q$, therefore, the projection of $t$ over line $Q$ will also be lower than $P_Q$. If the area beyond $P_Q$ has more than $k$ points, these $k$ points all have scores (projections to line $L_Q$) higher than $t$, with respect to $Q$, which cannot be true, since we assume that $t$ belongs to the top-$k$ answer set of $Q$. ■

![Diagram](image.png)

**Figure 14.** Example of why a view $V$ is not always reliable for answering a query $Q$.

Much like the case of two dimensions, it is not possible to infer the inverse of the theorem. Even if the area above $P_Q$ does not contain $k$ tuples it would still be possible to answer $Q$ with tuples that belong to $V$ if a critical area below the hyperplane $P_V$ does not contain any tuples. The critical area is the one defined between the hyperplane $P_V$ and the hyperplane that is perpendicular to $L_Q$ and passes from the point belonging in $V$ and producing the lowest possible score in regards to the query $Q$.

**Theorem.** It is possible that $V$ 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 hyperplane $P_V$ and the hyperplane that produces the lowest possible score for $Q$ from the tuples of $V$ is void of tuples.

*Proof.** Assume $t_1$ be a tuple in $V$ and $s_Q(t_1)$ is its score in regards to $Q$. In addition let this tuple be the one that has the lowest score in regards to $Q$ among all the tuples from $V$. Assume $P_Q$ is the hyperplane that is perpendicular to $L_Q$ and passes through
point \( t_i \). If the region defined between the hyperplanes \( P_V \) and \( P_Q \) has no points, then all points within \( V \) are the ones producing the lowest possible scores for \( Q \). As a result, if \( V \) contains more than \( k \) points, it can answer \( Q \). ■

6.2 Discussion

Similarly to the two dimensional case, a couple of observations can be made at this point:

- In order to avoid the computation of \( Q \)'s score for all the tuples of \( V \), a safe criterion would be to use \( t_{\text{last}} \). \( t_{\text{last}} \) denotes a virtual point (which means that it does not necessarily belongs in \( V \) or \( R \)) of hyperplane \( P_V \) that produces the lowest score in respect to \( Q \).

- The above criterion can be used if one is interested in approximate results (in fact, the smaller the critical region, the higher the possibility that \( V \) can answer the query \( Q \)). In addition, sketches of the data distribution in \( R \) can also be helpful in deciding whether the region is empty or not and to what extent.

A second technical point has to do with testing whether a point belongs to the safe area or not. Assume the last tuple in \( V \) is \( t_n \) with score \( s_v(t_n) \) in regards to \( V \). Then the hyperplane \( P_V \) is described from the equation:

\[
\sum_{i=1}^{N} a_i \cdot x_i = s_v(t_n) \quad (\text{easy to check since the distance of point } O \text{ from the hyperplane } P_V \text{ equals to } s_v(t_n)).
\]

Without loss of generality assume that the hyperplanes \( P_Q \) and \( P_V \) intersect with the \( X_i \) axis, where \( i \in \{1, \ldots, N\} \), in point \( x_{iV} \). Since \( x_{iV} \) belongs in \( P_V \) its coordinates are \( x_{iV} \). Similarly, it could be any other axis \( X_i \). The hyperplane \( P_Q \) is defined by the equation:

\[
w_Q \cdot (x_1 + x_2 + \ldots + x_N) = s_Q.
\]

Consequently, \( s_Q \) can be computed by taking into consideration that \( x_{iV} \) belongs in \( P_Q \) as well. Thus, \( s_Q \) is equal to:

\[
w_Q \cdot s_v(t_n) \cdot \sqrt{a_1^2 + a_2^2 + \ldots + 1 \cdot a_i^{-1}}
\]

and the hyperplane \( P_Q \) is defined from the equation:

\[
\sum_{i=1}^{N} a_{1Q} \cdot x_i + a_{2Q} \cdot x_2 + \ldots + x_N = s_v(t_n) \cdot \sqrt{a_1^2 + a_2^2 + \ldots + 1 \cdot a_i^{-1}}.
\]

Assume a tuple \( t_b \). Tuple \( t_b \) belongs to the safe area if:

\[
x_{b1} \geq -a_{1Q} \cdot x_{b1} - a_{2Q} \cdot x_{b2} - \ldots + s_v(t_b) \cdot \sqrt{a_{1Q}^2 + a_{2Q}^2 + \ldots + 1 \cdot a_i^{-1}}.
\]

6.3 Algorithmic results

Now, we are ready to give an algorithm that decides whether \( Q \) can be answered by \( V \) and populates \( V \) if the test is positive.
Algorithm Test View Suitability

**Input:** A materialized view $V(\text{ID}, X_1, X_2, \ldots, X_N, s)^n$, with its equation $s = w (a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + x_n)$ and its $n$ tuples, a $Q(\text{ID}, X_1, X_2, \ldots, X_N, s_Q)^k$, $s_Q = w_Q (a_{1Q} \cdot x_1 + a_{2Q} \cdot x_2 + \ldots + x_N)$, $k \leq n$.

**Output:** a decision on whether $Q$ can be answered by $V$ along with the population of $V$

```
Begin
1. Let $t_n$ be the $n$-th tuple of $V$, $t_n(x_1, x_2, \ldots, x_n) = V[n]$.
2. if $(P_Q, P_V)$ intersect in $X_i$ axis{
3.     compute point $x_{iV}$: $x_{iV} = s_v(t_n) \cdot \sqrt{a_1^2 + a_2^2 + \ldots + 1} \cdot a_i^{-1}$
4.     define plane $P_Q$ as $x_N = -a_{1Q} \cdot x_1 - a_{2Q} \cdot x_2 - \ldots + x_{iV}$
5. }
6. Let $t_k$ be the $k$-th tuple of $V$, $t_k(x_1^k, x_2^k, \ldots, x_N^k) = V[k]$.
7. if $t_k$ belongs to the safe area above hyperplane $P_Q${
8.     for all tuples of the safe area of $V$
9.         compute $s_Q(V[i])$ & add it to $V$'s extent
10.     //could use LPTA, TA, or any other method to compute the top-k
11.     return(true);
12. }
13. else return(false);
End.
```

**Figure 15.** Algorithm Test View Suitability

Again, remember that we assume that the materialized view is memory resident, so we do not need to resort to unnecessary I/O’s.

### 6.4 Working with more than one views

Das et al [DGKT06] 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 $V_1$ and $V_2$. Is it possible to use these two views to answer $Q$ without referring to the relation $R$?

A query $Q$ is encompassed by two preexisting, materialized views $V_1$ and $V_2$. $L_{V1}$ and $L_{V2}$ denote the lines that represent the two views. In addition, assume $P_1$ and $P_2$ denote the hyperplanes that are perpendicular to $L_{V1}$ and $L_{V2}$ and pass from the last point contained in $V_1$ and $V_2$ respectively. The hyperplanes $P_{Q1}$ and $P_{Q2}$ are perpendicular to the line $L_Q$ of the query and assume that $P_{Q1}$ meets $P_1$ in $X_i$ axis
whereas \( P_{Q2} \) meets \( P_2 \) in \( X_i \) axis, with \( i \neq j \). The sub-region above \( P_{Q1} \) towards infinity characterizes the safe area for \( V_1 \). Similarly, the sub-region above \( P_{Q2} \) towards infinity characterizes the safe area for \( V_2 \).

**Theorem.** 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 hyperplanes \( P_{Q1} \) and \( P_{Q2} \) are both vertical 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.

### 7. 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. 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. Finally, we have discussed the issue of providing partial results for a query via a materialized view by splitting the range of score into appropriate sub-ranges. This way, different parts of the query answer can be obtained in parallel, by distributing their processing to different servers.

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.

### References


[Fagi98] 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.


