Αποθηκευμένες Όψεις για Κατατακτήριες Ερωτήσεις με Άνω Όριο Αποτελεσμάτων: Επεξεργασία ερωτήσεων, Ενημέρωση και Ομοιότητα

Η
ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από την

Ευτυχία Μπαϊκούση

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΔΙΔΑΚΤΟΡΙΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Ιανουάριος 2012

Τριμελής Συμβουλευτική Επιτροπή

- **Βασιλειάδης Παναγιώτης,** Επίκουρος Καθηγητής του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων (επιβλέπων).
- **Πιτουρά Ευαγγελία,** Αναπληρώτρια Καθηγήτρια του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων.
- **Σελλής Τιμολέων,** Καθηγητής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πανεπιστημίου.

Επταμελής Εξεταστική Επιτροπή

- **Βασιλειάδης Παναγιώτης,** Επίκουρος Καθηγητής του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων (επιβλέπων).
- **Δημακόπουλος Βασίλειος,** Επίκουρος Καθηγητής του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων.
- **Ζάρρας Απόστολος,** Επίκουρος Καθηγητής του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων.
- **Παληός Λεωνίδας,** Αναπληρωτής Καθηγητής του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων.
- **Πιτουρά Ευαγγελία,** Αναπληρώτρια Καθηγήτρια του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων (μέλος τριμελούς συμβουλευτικής επιτροπής).
- **Σελλής Τιμολέων,** Καθηγητής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πανεπιστημίου (μέλος τριμελούς συμβουλευτικής επιτροπής).
- **Τσαπάρας Παναγιώτης,** Επίκουρος Καθηγητής του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων.

ACKNOWLEGMENTS

I would like to express my warmest thanks and gratitude to my advisor Prof. Panos Vassiliadis for the valuable guidance, advice and encouragement he has offered while supervising my thesis. For the time and effort he spent on my work throughout all these years. Collaborating with him has been a pleasant and memorable experience. I am also very grateful to Prof. Evaggelia Pitoura and Prof. Timos Sellis for acting as secondary referees for my thesis. Many thanks also to Prof. Vassilios Dimakopoulos, Prof. Apostolos Zarras, Prof. Leonidas Palios, Prof. Panayiotis Tsaparas for serving in my thesis committee.

I would also like to express my gratitude to the state scholarship foundation IKY for the funding provided throughout my studies as well as to Prof. Nikos Mamoulis for his time and effort spent while collaborating with him and for the funding he provided while visiting the Hong Kong University.

Last but not least, I would like to thank all people of the DMOD Laboratory that throughout all these years have provided helpful feedback and turned the countless hours in the office into a joyful and pleasant experience. Especial thanks to my office-mate Giorgos Rogkakos for his collaboration while working on the similarity of multidimensional data points. Finally, I would like to thank my family and parents for their continuous support and understanding all these years.

This thesis has been supported by the state scholarship foundation IKY.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

Eftychia Baikousi.
PhD, Computer Science Department, University of Ioannina, Greece, January 2012.
Title of Dissertation: Materialized Views for top-*k* queries: Query Processing, View Refreshment and Similarity.
Supervisor: Panos Vassiliadis.

Nowadays, there is a huge amount of data available to users. Due to the variety and great volume of data, retrieving the most important pieces of information, can become an overwhelming task. In the areas of Information Retrieval and Data Management, researchers have paid attention to the generic problem of retrieving the top-*k* similar objects from a repository according to a users preference query. In the field of Data Management, this problem is known as *top-k querying problem*. In the field of Information Retrieval, in applications such as multimedia retrieval, the problem is mainly addressed as finding the most similar objects to a given one according to a similarity metric. The goal of this thesis is to explore and investigate the answering of top-*k* queries through the exploitation of materialized top-*k* views. In addition, we study the problem of capturing the distance function that best complies with human perception for finding the similarity between two data collections of multidimensional points under the form of OLAP cubes.

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) and a query Q over R the desideratum is to retrieve the top-*k* tuples from R having the k highest values according to a scoring function f that accompanies Q. In an effort to improve the performance of the retrieval of top-*k* tuples from R, we study the problem by taking into consideration results from previously posed queries that are cached as *materialized views*. We study the problem by acquainting a geometric representation and we provide theoretical guarantees on whether a materialized view is able to

answer a top-*k* query. We proceed by proposing the SafArI algorithm for deciding the usability of a materialized view as well as the answer of the top-*k* query, in case the view is suitable for the query.

In the presence of updates in the relation over which a set of views is defined, we provide a method for keeping the top-*k* materialized views up to date without needing to re-compute them and provide results in two directions. Firstly, we deal with the problem of maintaining top-k views in the presence of high deletion rates and provide a principled method that is independent of the statistical properties of the data and the characteristics of the update streams. Secondly, we assess the problem of efficiently maintaining multiple top-*k* views, where we provide theoretical guarantees for the nucleation of a view with respect to another view and the reflection of this property to the management of updates. Further on, we propose an algorithm that maintains a large number of views, via their appropriate structuring in hierarchies of views.

Apart from finding top-*k* answers for data in the form of multidimensional points, we also assess the problem of finding how similar are two collections of data according to human perception. To put the question a little more precisely, given two sets of points in a multidimensional hierarchical space, what is the distance between these two collections? In applications such as multimedia information retrieval and digital libraries, where contemporary data lead to huge repositories of heterogeneous data stored in data warehouses, there is a need of similarity search that complements the traditional exact match search. We address the problem by (a) organizing alternative distance functions in a taxonomy of functions and (b) experimentally assessing the effectiveness of each distance function via a user study in order to discover which distance function is mostly preferred by the users.

ΕΚΤΕΝΗΣ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Ευτυχία Μπαϊκούση.
PhD, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιανουάριος 2012.
Τίτος Διατριβής:
Επιβλέπων: Παναγιώτης Βασιλειάδης.

Λόγω του μεγάλου όγκου δεδομένων και της πληθώρας πληροφοριών που είναι διαθέσιμες στους χρήστες μέσω διαδικτύου και όχι μόνο, είναι αναγκαία η αποδοτική ανάκτηση των πιο ενδιαφερόντων και προτιμητέων πληροφοριών. Τόσο στην περιοχή της Ανάκτησης Δεδομένων όσο και στο χώρο των Βάσεων Δεδομένων, οι ερευνητές έχουν ασχοληθεί με το γενικότερο πρόβλημα της ανάκτησης και εξόρυξης των κ πιο όμοιων αντικειμένων από ένα σύνολο αντικειμένων σύμφωνα με τις προτιμήσεις που θέτουν οι χρήστες. Συγκεκριμένα, στο χώρο των Βάσεων Δεδομένων, το πρόβλημα διατυπώνεται ως η ανάκτηση της απάντησης κατατακτήριων ερωτήσεων με άνω όριο αποτελεσμάτων. Στο χώρο της Ανάκτησης Δεδομένων, το πρόβλημα κυρίως απαντάται ως η εύρεση των πιο όμοιων αντικειμένων ως προς ένα δεδομένο αντικείμενο, όπως, παραδείγματος χάριν, σε εφαρμογές ανάκτησης δεδομένων από βάσεις πολυμέσων. Ο στόχος της παρούσας διατριβής είναι η μελέτη και η έρευνα του προβλήματος της ανάκτησης των αποτελεσμάτων κατατακτήριων ερωτήσεων με άνω όριο αποτελεσμάτων μέσω της χρήσης υλοποιημένων όψεων. Επιπρόσθετα, μελετάται το πρόβλημα του εντοπισμού της συνάρτησης απόστασης που μπορεί να χρησιμοποιηθεί ώστε να εκφράσει την ανθρώπινη αντίληψη για την εύρεση της ομοιότητας από δυο συλλογές δεδομένων στο πολυδιάστατο χώρο.

Το πρόβλημα της απάντησης κατατακτήριων ερωτήσεων με άνω όριο αποτελεσμάτων αφορά την ανάκτηση των κ αποτελεσμάτων με την υψηλότερη τιμή σύμφωνα με μια κατατακτήρια ερώτηση που τίθεται σε μία βάση δεδομένων. Συγκεκριμένα, δοθείσης μιας σχέσης R (tid, A_1, A_2,..., A_m) και μιας ερώτησης Q πάνω στην R, ο στόχος είναι η ανάκτηση των κ πλειάδων από τη σχέση R τέτοιων ώστε να

έχουν τις κ υψηλότερες τιμές σύμφωνα με μια συνάρτηση βαθμολόγησης που συνοδεύει την ερώτηση Q. Σε μια προσπάθεια να βελτιώσουμε την απόδοση της ανάκτησης των κ υψηλότερων, ως προς την τιμή, πλειάδων από τη σχέση R, μελετάμε το πρόβλημα κάνοντας χρήση των αποτελεσμάτων που ανακτήθηκαν από προηγούμενες ερωτήσεις και τα οποία έχουμε αποθηκεύσει με την μορφή υλοποιημένων όψεων. Μελετάμε το πρόβλημα υιοθετώντας μια γεωμετρική αναπαράσταση και παρέχουμε θεωρητικές εγγυήσεις για το κατά πόσο τα αποτελέσματα της υλοποιημένης όψης μπορούν να προβούν αρκετά ώστε να απαντηθεί η κατατακτήρια ερώτηση με άνω όριο αποτελεσμάτων. Προτείνουμε τον SafArI αλγόριθμο για την απόφαση της χρησιμότητας της υλοποιημένης όψης καθώς και για την απάντηση της κατατακτήριας ερώτησης με άνω όριο αποτελεσμάτων όταν η χρήση της όψης είναι κατάλληλη για την δοθείσα ερώτηση. Ο αλγόριθμος στηρίζεται στην αποκαλούμενη *ασφαλή περιοχή* (*safe area*) μιας υλοποιημένης όψης, η οποία ορίζεται αφενός από την όψη και αφετέρου από μία ερώτηση με άνω όριο αποτελεσμάτων.

Επιπλέον, προτείνουμε μια μέθοδο για την διατήρηση της ενημερότητας των υλοποιημένων όψεων με άνω όριο αποτελεσμάτων χωρίς να χρειαστεί ο επανυπολογισμός τους, όταν προκύπτουν ενημερώσεις σε μια σχέση πάνω στην οποία είναι ορισμένες οι όψεις. Το πρόβλημα μελετάται προς δύο κατευθύνσεις. Πρώτον, αντιμετωπίζουμε το πρόβλημα σε συνθήκες αυξημένου ρυθμού διαγραφών και προτείνουμε μια καλά ορισμένη μέθοδο ανεξάρτητη των στατιστικών ιδιοτήτων που έχουν αφενός τα δεδομένα και αφετέρου, οι ενημερώσεις. Δεύτερον, επιλύουμε το πρόβλημα της αποδοτικής ενημέρωσης υλοποιημένων όψεων με άνω όριο αποτελεσμάτων, όπου προτείνουμε θεωρητικές εγγυήσεις για τον εγκλεισμό μιας όψης από μια άλλη όψη και το αντίκτυπο αυτής της ιδιότητας στην διαδικασία διατήρησης της ενημερότητας πολλαπλών όψεων. Συγκεκριμένα, προτείνουμε αλγοριθμικές τεχνικές για την διατήρηση της ενημερότητας πολλών όψεων κατασκευάζοντας και κάνοντας χρήση κατάλληλων ιεραρχικών δομών των όψεων.

Εκτός της απάντησης ερωτήσεων με άνω όριο αποτελεσμάτων από δεδομένα του πολυδιάστατου χώρου, επιλύουμε το πρόβλημα της εύρεσης της ομοιότητας δυο συλλογών δεδομένων. Συγκεκριμένα, το πρόβλημα εντοπίζεται στην εύρεση της

κατάλληλης έκφρασης της ομοιότητας δυο συλλογών δεδομένων σύμφωνα με την ανθρώπινη αντίληψη. Με άλλα λόγια απαντάμε στο εξής ερώτημα: Δοθέντων δυο συνόλων που περιέχουν σημεία του πολυδιάστατου ιεραρχικού χώρου, ποια είναι η απόσταση ανάμεσα στα δεδομένα δύο σύνολα; Ειδικά σε εφαρμογές όπως η ανάκτηση πληροφοριών από βάσεις δεδομένων υπό τη μορφή πολυμέσων καθώς και ψηφιακές βιβλιοθήκες, υπάρχει επιτακτικά η ανάγκη της εύρεσης ομοιότητας δεδομένων που συμπληρώνει την παραδοσιακή εύρεση του απόλυτα ταιριαστού αντικειμένου ως προς ένα άλλο δεδομένο. Ιδιαίτερα σε τέτοιου είδους εφαρμογές, η φύση των δεδομένων οδηγεί σε τεράστιες συλλογές δεδομένων διαφορετικού τύπου τα οποία αποθηκεύονται σε αποθήκες δεδομένων. Μελετάμε το πρόβλημα σε δύο άξονες. Πρώτον, οργανώνουμε διάφορα είδη συναρτήσεων αποστάσεων σε μια ταξινομία συναρτήσεων. Δεύτερον, αποτιμούμε πειραματικά την αποτελεσματικότητα της κάθε συνάρτησης απόστασης μέσω μιας πειραματικής μελέτης με πραγματικούς χρήστες ώστε να ανακαλύψουμε την συνάρτηση απόστασης που προτιμάται κατά κύριο λόγο από τους χρήστες.

CHAPTER 1. INTRODUCTION

Due to the vast amount of data and information available to users (especially via the Web), the problem of retrieving the most important pieces of information can become an overwhelming task. In the areas of Information Retrieval and Data Management, researchers have been attracted to the generic problem of retrieving the top-*k* similar objects from a repository according to a user's preference query. In the field of Data Management, this problem is known as *top-k querying problem*. In the field of Information Retrieval (e.g., in applications such as multimedia retrieval), the problem is mainly addressed as finding the most similar objects to a given one according to a similarity metric. Consider for example, a database containing data about hotels, restaurants and attractive places to see in a designated area where travelers arrive at an airport. When airplanes arrive, several potential sightseers arrive with it, at the same time a massive number of travelers depart. Assume that travelers are equipped with wireless devices such as smart phones or tablets and can connect to the airport's server. Assume a relation *Traveler* (*t_id*, *t_age*, *t_maritalStatus*, …) as well as relations about the traveler's profile and travelling history. For a municipal employee who is assigned to advertise the interesting places to travelers, it is important to find the top-*k* attractions according to their profiles. In order to do so, the employee uses queries with scoring functions over the traveler's characteristics. For instance, assume the employee wants to advertise the Christmas Village that the municipal built for the Christmas Holidays at present. Thus, the employee needs to create a profile for the new attraction. The profile includes a formula that assigns a score for potential sightseers according to similarity functions that match the characteristics of the

attraction to the characteristics of the traveler. To speed things up, it is reasonable to find the top-*k* travelers in order to send them the related advertisement. In other words, the employee's task is reduced to finding the top-*k* travelers according to the employee's scoring function. Due to the departures of the airplanes, the top-*k* list of travelers needs to be refreshed so that the remaining possible sightseers are notified. Therefore, the top-*k* list of travelers should be maintained when updates occur in the relation *Travelers*.

1.1. Terminology and Contribution in a Nutshell

The goal of this thesis is to explore and investigate the answering of top-k queries through the exploitation of materialized top-k views. To clarify the aforementioned statement for the non-expert reader, we have to provide informal explanations of the two terms that define its essence.

- A *top-k*, or *ranking query* requests the *k* highest tuples of a relation *R* according to a scoring function over the attributes of the relation.
- In the field of databases the term (relational) *view* comes in two flavors [Rous97]. The first category of views comes under the terms *plain view*, *unmaterialized view*, or simply *view*, and it is actually a query expression in the form of a macro with no extensional attachments which is executed at run-time. In simple terms, we can register a query to the database management system as a view; then, subsequent queries can reuse this query as a data source. Note that the results of the query are not cached in the system and, thus, whenever used in another query, a plain view acts as a macro that is resolved to its original constituents, integrated in the new query and executed as part of it. If, on the other hand, we wish to speed up the execution of the subsequent queries, we can register a view as a materialized view. A *materialized view* caches the results of the query and therefore, it can acts as a typical relation in the execution of subsequent queries. At the same time, whenever updates occur to the relations that are used in the definition of the materialized view, the latter has to be refreshed with the new data. In both their families, views are characterized by the duality of coming with (a) a query expression that defines them (thus they are queries in a sense) and (b) a

set of tuples (the result of the query) that makes them appear as relations, too (either computed on the fly, or appropriately materialized in the background). In any case, views are a powerful mechanism, frequently used to make the life of the developer easier and the execution of the system faster.

In this thesis, we refer to the notion *materialized ranking view* in order to describe a materialized view that contains the results of a top-*k* query. Apart from answering top-*k* queries through materialized views, we also study the problem of maintaining top-*k* materialized views in the presence of updates in the relation such that the views can be up to date and useful for the answering of top-*k* queries. In addition, in order to express similarity between objects there is the need of discovering the distance functions that users prefer for computing the similarity of two data collections. In order to do so we resort to the simplest framework that can be given to a user to work with and that is OLAP Cubes. Thus, we provide a taxonomy of the distance functions used for collections of multidimensional data and conduct an extensive user study analysis in order to reveal the most preferred function by users.

1.2. Thesis Contribution & Outline

The technical contributions of this thesis are organized in three chapters, each solving one of the three aforementioned problems. In the sequel, we give an overview of the technical contributions of each chapter; in the final chapter of this thesis, we conclude our results and present insights for future work.

Answering top-*k* Queries via Materialized Views

In Chapter 2, we work on the problem of answering top-*k* queries by making use of materialized ranked views. We provide theoretical and algorithmic results for the above problem. Firstly, we adopt a geometric representation of the top-*k* query problem and then we conduct a theoretical analysis for providing theoretical guarantees for the suitability of a materialized view in order to answer a top-*k* query. Specifically, 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. Initially, we study the problem for a top-k query answering in the 2-dimensional space. Following, we generalize the problem for the n dimensional

space. In addition, 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. We also 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. We demonstrate the efficiency and effectiveness of our method over a set of extensive experiments over both synthetic and real datasets. The results of this chapter have previously been published in [BaVa09].

Maintaining Materialized top-k Views

In Chapter 3, we study the problem of maintaining materialized top-*k* views and provide results in two directions. The first direction is towards maintaining top-k materialized views in the presence of high deletion rates. We propose a principled method that complements the inefficiency of the state of the art independently of the statistical properties of the data and the characteristics of the update streams. Our method consists of the following steps: (a) a computation of the rate that actually affects the materialized view, (b) a computation of the necessary extension to *k* in order to handle the augmented number of deletions that occur and (c) a fine tuning part that adjusts this value to take the fluctuation of the statistical properties of this value into consideration. Secondly, we deal with the problem maintaining multiple top-*k* views and their efficient maintenance in the presence of updates to their base relation. To this end, we provide theoretical guarantees for the establishment of the effect of updates to a certain view, whenever we know that another view has been updated. We introduce the notion of nucleation (i.e., dominance relationship) between views and based on this notion we propose a hierarchical structure of the materialized views. Through the appropriate hierarchical structuring of the views we provide algorithmic results towards the maintenance of a large number of views. Finally, we show that our method accurately sustains intervals with high deletion activity in the workload through our experiments. In addition, we show that our method outperforms the state-of-the-art, as the computation of the exact number of auxiliary view tuples is

faster than the computation of refill queries as proposed in the related literature. The results of this chapter have previously been published in [BaVa07], [BaVa10].

Similarity Measures for Multidimensional Data

As already mentioned, in Chapter 2 and 3 we deal with the problem of answering top-k queries from data in the form of points in the multidimensional space. Each top-k view or query is a collection of such points, ranked according to a scoring function. However, although we have answered the question "Given a query, can we use a view to answer it?" we have not answered the question "Given a query and a set of views, can we find the one that is most similar to it?". We believe that in the heart of this problem of view similarity is the answer to the question "How similar are two data collections?". In Chapter 4 we study the problem of discovering the distance functions for computing the similarity of two data collections, *according to what real users actually think*. In order to do so, we resort to the simplest framework that can be given to a user to work with and that is OLAP Cubes and hierarchical multidimensional spaces. OLAP is preferred for simplicity as it organizes data in dimensions and measures that are most intuitive to users. We model a collection of data in the form of a multi-dimensional array called *Cube*. Specifically, we provide a taxonomy of distance functions that are applied between two OLAP cubes. We provide an extensive user study that reveals the distance functions that more close to human perception. In the first user study analysis we discover, which distance function between two values of a dimension is best with regard to the user needs. We show that our findings indicate that the distance function $\delta_{\text{LCA,P}}$, which is expressed as the length of the path between two values and their common ancestor in the dimension's hierarchy is the most preferred by users in our experiments. Moreover, two more functions are widely chosen by users. These are the highway functions δ_{Anc} that is expressed with regard to the ancestor x_y and $\delta_{\text{H,Desc}}$ that is expressed by selecting the representative from a descendant. According to this findings, in the second user study we aim in discovering which distance function (the *closest relative* or the *Hausdorff* distance function) from the category of distance function between two data cubes, users prefer. Overall, the former function was preferred by the users than the latter; however the individual scores of the tests indicate that this advantage is rather narrow. The results of this chapter have previously been published in [BaRV11].

CHAPTER 2. VIEW USABILITY FOR ANSWERING TOP-K QUERIES OVER MATERIALIZED VIEWS

The first problem that we address in this thesis is finding an answer to the question on how we can decide on the suitability of a materialized ranking view to answer a ranking query.

Before proceeding, we formally define a top-*k* or ranking query.

Given *a relation R (tid, A_1, A_2, ... A_m) and a query Q over R having the form of a score function* $f : dom(A_1) \times ... \times dom(A_m) \rightarrow \Re$,

Retrieve *the top-k tuples from R*

Having *the k highest values according to the scoring function of Q.*

In this Chapter, we first describe the related literature and background. For reasons of presentation, we start our technical analysis in Section 2.2 with an analysis of the problem of answering a top-*k* query through the usage of materialized views for the 2-dimensional case. Specifically, we provide theoretical guarantees for the suitability of a materialized view in the answering of the query and propose the adequate algorithm, the 2DSafArI algorithm. In Section 2.3 we generalize the problem and our

findings for the *n* dimensional case. Then, in Section 2.4 we deal with the problem of answering top-*k* queries through the usage of more than one materialized views. Firstly, we show that the usage of the union of the safe areas of two views do not add better guarantees for the answering of a query. Secondly, we exploit the problem of answering a top-*k* query by parallelizing its process and assigning different parts of the query's answer to a different view and then uniting the results. In Section 2.5 we present the results of our experiments for our proposed methods. Finally, in Section 2.6 we summarize our findings.

2.1. Background and Related Work

In this section, we give an overview of the basic algorithms that answer a top-*k* query over a relation *R*. Firstly we describe the algorithms that provide an answer to a top-*k* query. Secondly, we describe the algorithms that make use of materialized views in order to answer a top-*k* query. Although, we discuss the related work that pertains to the problems of this thesis in detail, it is possible that a reader is interested for a more extensive coverage of the area, outside the bounds of this thesis' problems; in this case, we refer the interested reader to a comprehensive survey by Ilyas et. al [IlBS08] that covers the area of top-k query processing in a broad, yet structured perspective.

2.1.1. Algorithms for top-k Queries over Relations

In this section, we give an overview of the basic algorithms that answer a top-*k* query over a relation *R*.

Fagin's Algorithm (FA) [Fagi96], [Fagi98]

In 1996, R. Fagin published his seminal paper [Fagi96] in PODS on the topic of combining fuzzy information from multimedia information systems. The problem that Fagin attacks is motivated by the area of multimedia databases where a multimedia information system integrates data that reside in different database systems and posed queries ask for the *k* highest objects according to a monotone function over the fuzzy sets that describe the multimedia object. The problem then is that a user wants to score the tuples of the relation according to a scoring function (e.g., rank high the

photos with high amounts of blue and low contrast) and keep a fixed amount of them e.g., the best (top) k tuples, according to their score. The main idea of the algorithm is that every relation is accompanied by several sorted lists, one for each attribute. For example, assume a relation $R(id, x_1, x_2)$ from which we need to retrieve the top-k tuples under the scoring function Q: $min(x_1, x_2)$ where for each attribute there is one sorted list (Figure 2.1). Then the goal of the proposed algorithm is to exploit the lists in order to speed up the identification of the top-k tuples. Formally, the problem addressed by Fagin is as follows. 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.

Definition 2.1 (Monotone Aggregation Function). A scoring function $g(X)$ is a monotone aggregation function if for any tuple $t(x_1, \ldots, x_m)$ the following hold

1. $g(t)$ is an aggregation function over the attribute values of the tuple t and
2. *if* for every attribute value x_i of tuple t and x'_i of tuple t' such that $x_i \leq x'_i$, then $g(t) \leq g(t')$ (monotone).

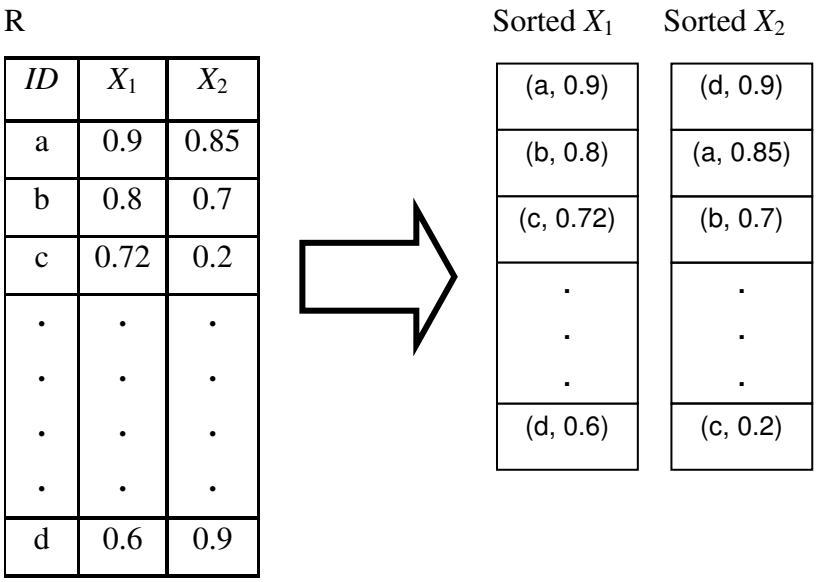


Figure 2.1. Example of Sorted Lists of a Relation's Attributes.

The FA algorithm consists of 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. The properties for function *g* are important in the sense that they assure that all tuples not seen under sorted access do not participate in the top-*k* tuples.

Threshold algorithm (TA) [FaLN01] [GüBK00] [NeRa99]

FA is optimal in high probability sense whereas, the threshold algorithm is instance optimal. Similarly to *FA*, *TA* can be applied over a relation 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 τ as $g(\underline{x}_1, \underline{x}_2 \ldots \underline{x}_m)$ where $\underline{x}_i$ 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 τ. 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.

TA is correct when *g* is a monotone aggregation function. In addition, [FaLN01] have proved that *TA* is instance optimal. An algorithm *B* is *instance optimal* over a class of algorithms $\boldsymbol{A}$ and a class of legal inputs $\boldsymbol{D}$ to the algorithms when $B \in \boldsymbol{A}$ and if for every $A \in \boldsymbol{A}$ and for every $D \in \boldsymbol{D}$, we have *cost* (*B*, *D*) =O (*cost* (*A*, *D*)), where *cost* (*B*, *D*) is the middleware cost incurring by running the algorithm *B* over database *D*.

Variations of the Threshold Algorithm

Apart from *TA* algorithm, there were a number of variations proposed by researchers. The *NRA* algorithm proposed by [FaLN01] and the *LARA* algorithm proposed by [MCYC06] finds the top-k tuples by conducting only sorted accesses and without supporting random accesses over the relation R. The top-k tuples are retrieved but their actual scores may not be reported, since the algorithm retrieves the tuples based on bounds of their scores. Moreover, [FaLN01] describe the TAz algorithm that is a variation the TA algorithm in case sorted accesses are prohibited to all of the sorted lists. In addition, [FaLN01] describe the TA-θ algorithm that is an approximation of the TA algorithm. Specifically, TA-θ finds a θ approximation of top-k tuples in the sense that the algorithm's stopping condition is reached when at least k tuples with score at least equal to τ / θ are retrieved. Finally, [FaLN01] also describe the CA (Combined Algorithm) algorithm that allows random accesses but takes into consideration the cost of a random access relatively to the cost of a sorted access.

2.1.2. Algorithms for top-k Queries over a Relation and Materialized Views

FA and *TA* are two well-known algorithms that solve the problem of answering top-k queries over a database with a quite good 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 indices or/and materialized views. In the setting of materialized views, results of previous top-k queries are stored in the form of materialized views. Then, a new top-k query may be answered through materialized views resulting in better performance than making use only of the base relation from the database.

The Onion Technique: Indexing for Linear Optimization Queries [CBC++00]

The onion technique [CBC+00] consists of the so called *onion indices* that involve layered convex hulls. Specifically, assuming that each tuple is represented as a point in the N dimensional space, with a dimension representing the values of an attribute, the onion indices are a set of layered convex hulls of these points (Figure 2.2). These convex hulls can be used in order to retrieve in a consecutive way the top-1 tuple, top-2 tuple and so on until all top-k tuples are retrieved. The top-1 tuple is retrieved by

exploiting the outmost convex hull, the top-2 tuple is retrieved by exploiting the remaining points of the outmost convex hull along with the points of the next layered convex hull and so on, and until all top-*k* points are retrieved. Since this method consists of an indexing technique, it provides performance gains. Nevertheless the main drawback of this technique is the fact that it cannot be used when the top-*k* query involves constraints such as predicates on attribute values. Also, the construction of the convex hulls is time consuming and thus it is not suitable especially in the presence of updates in the relation.

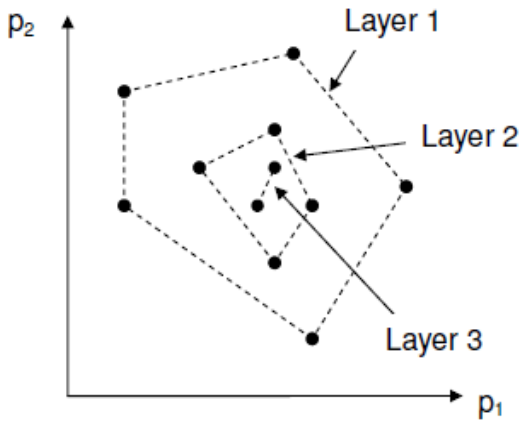


Figure 2.2. Convex Hulls in 2 Dimensional Space.

Rank Join Indices [TPK++03]

Apart from the Onion indices, the Rank-join indices [TPK+03] is another type of indices for the retrieval of top-*k* results. Tsaparas et al. ([TPK+03]) solve the problem of top-*k* query answering under the following setting:

Given Two relations $R(A_1, A_2, \ldots A_n)$, $S(B_1, B_2, \ldots B_m)$ with A_1 and B_1 being rank attributes, from which a new relation $R \bowtie_\theta S$ (A_1, B_1) is obtained by joining R and S over the attributes $A_2, \ldots A_n$ and $B_2, \ldots B_m$.,

Find the top-*k* tuples from $R \bowtie_\theta S$ according to a linear scoring function over the attributes A_1 and B_1.

The problem addressed consists of two sub-problems. The first sub-problem is to prune unnecessary tuples that will not be part of the top-k answer prior to the join result. The second sub-problem is to index and materialize the remaining tuples in order to answer any top-k linear query. In order to retrieve the top-k tuples from the join results there is no need to join all tuples from R with all tuples of S. Thus, the first sub-problem is to prune the join results. Given a relation R of size n (i.e., $|R| = n$) and a relation S, in order to find the top-K tuples of the joined results, we can join each tuple in R only with the top-K tuples of S. Thus, in the worst case, the join result will produce $n \cdot K$ tuples instead of $|R| \times |S|$. In addition, if a tuple in the joined result is dominated by at least K tuples then this tuple can be excluded from the joined ($n \cdot K$) tuples since this tuple will never be part of any top-k answer with $k \leq K$. For a tuple t (s_1, s_2) in the 2-D space, t is dominated by t' (s'_1, s'_2) if and only if $s_1 \leq s'_1$ and $s_2 \leq s'_2$.

In order to retrieve the top-K tuples from $R \bowtie_\theta S$, the two initial steps are:

a. Find $n \cdot K$ joined tuples, denoted as C and
b. Exclude from C the dominated tuples, i.e., exclude all tuples that are dominated by at least K tuples from C, this new set is denoted as D_K.

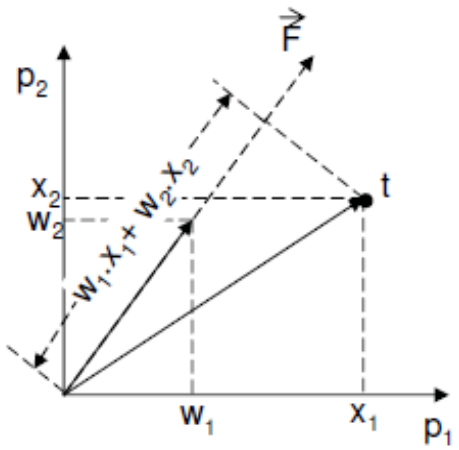


Figure 2.3. Vector Representation of Scoring Function and Rank Attributes.

For any linear function f and a value $k \leq K$, the top-k tuples of $R \bowtie_\theta S$ in regards to f can be retrieved by only taking into consideration tuples in the set D_K. So, the second sub-problem is to manage to index-materialize D_K in order to answer any top-k linear

query (with $k \leq K$). Assume that any tuple from D_K, is represented as a point in the 2-D space whose dimensions are the two rank attributes A_1 and B_1. Any linear scoring function $f = w_1 \cdot x + w_2 \cdot y$ is represented as a vector beginning from the origin of the axes and ending at the point (w_1, w_2). The score of a tuple t in regards to a scoring function f is found by computing the length of the projection of tuple t over the vector f as shown in Figure 2.3.

The ordering of the tuples in regards to the scoring function of f can be found by ranking the length of the projections of the tuples over f. The problem is to determine how the ordering of tuples alters when the scoring function f sweeps the 2-D space. The 2-D space is swept by using a vector of increasing angle in order to represent any possible linear scoring function by changing the weight factors of f. Thus, the scores of a tuple t in regards to any linear scoring function f can be materialized. The ordering of the tuples from D_k in regards to a scoring function f can be found by ranking the length of the projections of the tuples over f. The problem is to determine how the ordering of tuples alters when the scoring function f sweeps the 2-D space.

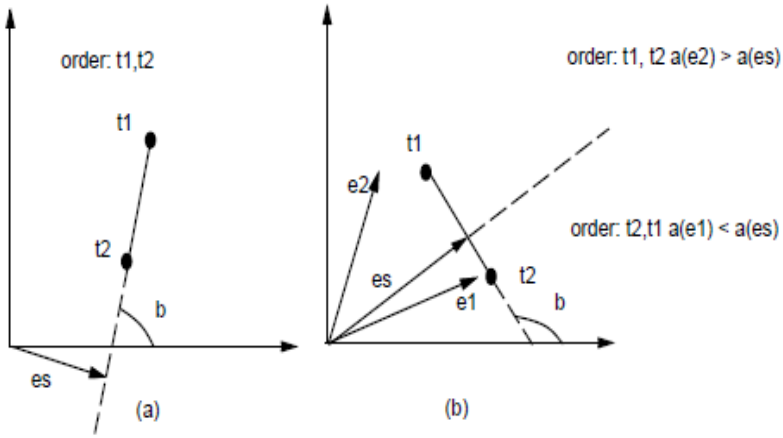


Figure 2.4.Possible Orderings of Tuples t_1 and t_2. (a)

Assume two tuples t_1 and t_2 with t_1t_2 being the line segment of the two tuples. In case the slope of t_1t_2 is positive (Figure 2.4a), then the ordering between t_1 and t_2 is the same for all possible scoring functions. In case the slope of t_1t_2 is negative (Figure 2.4b), then the ordering between t_1 and t_2 is reversed depending on the position of the

vector *f*. For a vector *f* being positioned "lower" from the perpendicular line to t_1t_2 and a vector *f'* being "upper" from the perpendicular line to t_1t_2 the ordering of the two tuples t_1 and t_2 is reversed for these two vectors.

For each pair of tuples, the *separating vector* is constructed. The separating vector of two tuples t_1 and t_2 is the vector which is perpendicular to the line segment t_1t_2. The set of all separating vectors for the tuples from the set D_K is denoted as *V*. In addition, the separating vectors in *V* are sorted in descending order in regards to the angle of the separating vector with the *X*-axis. The scoring function *f* sweeps the 2-D plane staring from the *X*-axis towards the *Y*-axis. When *f* meets a separating vector the new ordering of tuples is computed and materialized. Thus, if there are *M* separating vectors, the space is partitioned into *M*+1 regions. For each region the ordering of tuples is pre-computed and materialized. Therefore, when a top-*k* query with a linear scoring function arises, it is only needed to find the position of *f* in regards to the separating sector of the *M*+1 sectors of space. By determining the separating region, the answer of the top-*k* query is already pre-computed. In order to efficiently determine the separating region in which a new posed top-*k* query belongs, the authors propose an index structure. The index structure consists of B-tree index that contains all *M* separating vectors along with their top-*k* set, ordered according to the vector's angle in regards to the X-axis. Thus, when a new top-*k* query arises, the angle of the scoring function's vector is used and searched over the B-tree index, where then the corresponding top-*k* set is returned.

Prefer [HrKP01], [HrPa04]

The PREFER system introduced in [HrKP01], [HrPa04] answers preference queries through the usage of materialized views in a pipelined way. PREFER consists of a pre-processing step, the *ViewSelection* algorithm and the core algorithm *PipelineResults*. In the pre-processing part, PREFER decides which views should be materialized according to the system's performance requirements and a given relation. Thus, firstly PREFER executes the *ViewSelection* algorithm. Given a multidimensional space of *k* dimensions, each normalized in the interval [0, 1] and a set of views *V* over this multidimensional space, the *ViewSelection* algorithm computes a set of views *V'*, $V' \subseteq V$ that maximizes the number of points covered in

$[0, 1]^k$. Each view contains all tuples from the relation ordered according to each scoring function. In order to answer a new posed top-*k* query, the PREFER system selects the materialized view that best matches the new top-*k* query. Since every materialized view contains all tuples of the relation, any one of them could be used and would definitely provide the answer to the new top-*k* query. The materialized view that best matches the new query is the one that will access the less number of tuples in order to provide the new answer.

The answer of the new query *q* is retrieved in a pipelined way through the tuples of the materialized view *v*. The goal of the *PipelineResults* algorithm is to rank the tuples of a relation $R(A_1, ...,A_m)$ of *m* attributes, according to a query *q*. The query *q* is characterized by a preference vector. A preference vector is of the form $(w_1, w_2, \ldots w_m)$ where each coordinate w_i denotes the preferred weight of the *i*-th attribute. Therefore, the scoring function of *q* becomes $\sum_{i=1}^{m} w_i \cdot A_i$. Algorithm *PipelineResults* employs a views $R_v(tid, score_v)$ that contains the tuples of *R*, ranked by another preference vector *v*. Assume that the first set of tuples seen from the view *v* contains *l* tuples. In case $l \geq k$ the answer of the top-*k* query *q* can be computed. In case $l < k$, the next set of tuples from *v* are scanned. This procedure is repeated until *k* tuples have been seen. For each set of tuples from *v*, the number of tuples seen is based on the following property. Assume a top-*k* query *q* over a relation *R* with a scoring function F_q and a materialized view *v* with the scoring function F_v that orders the tuples of *R*. The l^{th} tuple from the first set of tuples seen from *v* is the maximum value of $T^1_{v,q}$ such that for every *t* in *R* : $F_v(t) < T^1_{v,q} \Rightarrow F_q(t) < F_q(t^1_v)$ where t^1_v is the top tuple in *v*. For any next iteration, tuple t^1_v is replaced with the tuple that has the highest score in *v* and has not been seen yet. The maximum value $T^1_{v,q}$ is called the *watermark* value. The watermark value is a score with respect to the ranking function of the materialized view that determines how deep in the ranked materialized view we should go in order to output the top result tuple of the query This way, the PREFER system can answer a top-*k* query by making use of one materialized view from a set of views that rank the entire relation R according to different linear scoring functions.

Linear Programming Adaptation of the Threshold Algorithm LPTA [DGKT06]

The LPTA algorithm is an algorithm that combines the results from materialized ranking views in order to answer a top-*k* query. Informally, a materialized ranking view is the materialized results of the tuples of a previously posed top-*k* query according to a linear scoring function. In other words, the LPTA algorithm answers a top-k query by making use of the tuples stored in materialized views Therefore, for each top-*k* query LPTA needs to solve two sub-problems: (a) Find the most suitable materialized views in order to answer the query and (b) retrieve the answer of the query by exploiting the materialized views chosen from the previous sub-problem.

LPTA is based on the TA algorithm and is applied on a set of materialized views in order to answer top-*k* queries. 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. In the sequel we assume a set of materialized views $V=(V_1, \ldots V_r)$ that contain the base views. LPTA is implemented through a two-step procedure.

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.

1. It does sorted accesses in parallel to each of the views.
2. 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.
3. 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.
4. 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 $score_i$ 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 $SF_i \leq score_i$. 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.

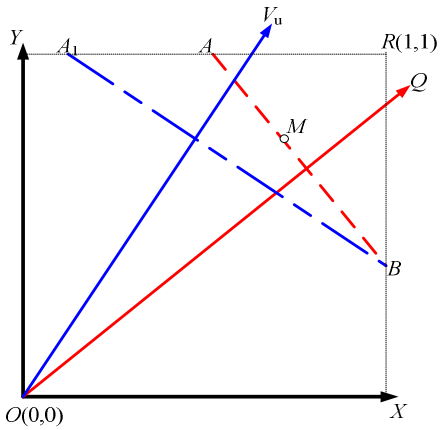
Assume a relation $R(id, X, Y)$ where without loss of generality the domains of X and Y are normalized over the interval [0, 1]. Apart from the base views V_x and V_y, assume two materialized views $V_u(id, Score_1)$ and $V_d(id, Score_2)$. Scores $Score_1$ and $Score_2$ are defined as linear functions 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: $y = a^{-1} \cdot x$. 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 $t(id, x, y)$ in regards to the scoring function can be found by projecting that point over the corresponding line. In Figure 2.5a 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 the materialized view V. LPTA performs sorted accesses over the tuples of V. This can be visualized as sweeping a line perpendicular to the vector of the view towards the point $O(0, 0)$. The order of tuples read by LPTA through sorted accesses over V is identical

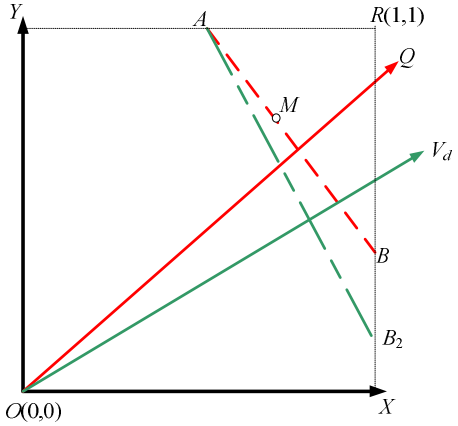
to the order of the points met by sweeping the line towards O. This means that the number of sorted accesses performed through the algorithm is the number of points that belong in the region of the triangle A_1BR for view V_u and the number of points that belong in the region of the triangle AB_2R for view V_d.

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 2.5c 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.

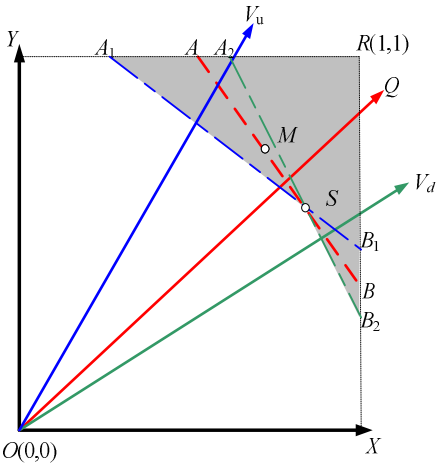
So far, in the above we describe the intuition of the geometric representation of the LPTA algorithm in order to answer a top-k query through the usage of the tuples materialized in a view. In the following, we see how the LPTA algorithm chooses the most suitable materialized views to use in order to answer the top-k query. 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 2.5c) 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.



(a) The query is lower than the view



(a) The query is higher than the view



(c) Two views for the answering of a query

Figure 2.5. Visual Demonstration of the LPTA Technique for Query Answering top-*k* via Views.

2.1.3. Related Problems in Different Context

Top-*k* queries have been extensively studied in research in centralized [ChGr99] systems and have proved very beneficial for applications such as multimedia retrieval and digital libraries. The growth of information available to users through internet has emerged researchers to support top-*k* queries in different contexts such as distributed systems and Peer to Peer systems. In addition, in an effort to improve performance issues researchers have studied the problem of answering top-*k* queries by making use of caching techniques.

Distributed Environments

Top-*k* queries have been extensively studied in research in centralized [ChGr99], as well as distributed environments such as Peer to Peer systems. Due to the growth of information available and the increased number of users accessing them over the Web, distributed systems have been proved to be very popular. Therefore, there is an emergence demand of supporting top-*k* queries in distributed environments. Most research has focused on answering top-*k* queries over a distributed system where data are partitioned either vertically [MaBG04, ChGM04, GuBK00, MiTW05, CaWa04], or horizontally [BNST05, VDNV08]. However, the common factor is that the relational data are distributed over sources and a newly posed query accesses part of them in order to retrieve the answer. The focus of these works has been the optimization of response times and scalability. Some techniques use a centralized node that describes which source contains which partition of data [CaWa04] or employ indexing techniques of the distribution of the data [MiTW05]. Other techniques adopt a model that contains super-peers that cache results of their peers [VDNV08], or address a network topology such as HyperCup [BNST05].

Caching Techniques

One way of overcoming problems such as network communication overhead and response times is through the usage of caching techniques. Caching previously posed queries and their results is an efficient method for dealing with issues of network overhead either in centralized systems [TrNY04] or in distributed systems such as P2P [SGAE04], where the latter support range queries. The exploitation of the result set of a previous query for the answering of a subsequent query is frequently encountered in the research literature (see for example [Koss00] and [Graef00]) under the name of query or view caching. Once a query is maintained in main memory for this purpose, it practically becomes a materialized view. Considering the case of top-*k* queries, in [ZhTZ07], the authors describe a system called BRANCA that answers top-*k* queries over an acyclic network of servers. The main idea of this system is based on the rationale of caching the results and information from previously posed top-*k* queries in order to make use of them for future ones. Specifically, each server contains a cache for each of its sub-graphs over the network. The cache retains results of previously posed top-*k* queries over the specific sub-graph. This technique results

in less communication cost over the network when a new top-*k* query arrives. [VDNV08] propose a system called SPEERTO that supports top-*k* query processing in a distributed environment making use of caching techniques through *K*-skybands. In this line of work, caching queries and their results is done through materialized views. The problem of answering queries using materialized views has been studied extensively for query optimizing, data integration, data warehousing and semantic data caching in client-server systems as well as top-*k* querying such as in [DGKT06] described earlier.

2.1.4. Research Opportunities and Comparison to Related Work

Related work has extensively dealt with the problem of answering top-*k* queries under various contexts [IlBS08]. To this end, previous efforts have provided various algorithms for efficiently answering such queries by making use of indexing techniques or taking into consideration results from previously materialized ranked views. In addition, top-*k* queries have been studied under the context of distributed databases and through caching techniques. However, there are still problems that remain open in the context of top-*k* query processing. Following, we highlight a set of interesting, fundamental problems that remain open in the context of query processing in the presence of materialized views for top-*k* queries.

1) Surprisingly, a missing piece in the related literature concerns the establishment of theoretical guarantees for the suitability of a materialized top-*k* view in order to answer a newly posed top-*k* query, regardless of probability estimations or statistical properties of the underlying dataset.
2) In a similar vein, another absent piece of theoretical groundwork concerns the efficient answering of top-*k* queries from materialized top-*k* views solely, without accessing the base relation over which the views are defined.
3) Finally, a theoretical analysis on the appropriate and needed number of materialized views for the answering of a top-*k* query is also missing from the current body of knowledge.

In this Chapter, we study the problem of answering top-*k* queries by making use of materialized ranked views in order to provide better performance. To this end we

provide a theoretical analysis based on geometric representation of the problem of whether and when a materialized view can be proved useful for answering such queries, something that has been missing from related work.

In the related work, the LPTA algorithm dealt with the problem of answering top-k queries through materialized ranked views. According to the estimation on the score of the last tuple of the query LPTA decides on the suitability of a materialized view in regards to the query. Specifically, [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 $topk_{\min}$) 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. To overcome this problem we conduct a theoretical analysis and provide theoretical guarantees along with the appropriate theorems that state whether and when a materialized ranking view is suitable for the answering of a top-k query.

In fact, [DGKT06] provide 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 $topk_{\min}$, 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, [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. In contrast, we propose algorithms that according to the theoretical establishments we provide, we retrieve the answer to a top-*k* query from exclusively the results of a materialized view, when this is possible, without having to scan all the tuples of the relation *R*.

Moreover, the LPTA algorithm selects the suitable views (usually more than one) in order to provide the answer to the top-k query. We theoretically prove that the theoretical guarantees of more than one views in regards to a top-*k* query do not offer further usefulness for answering the query compared to the guarantees of a single view. Specifically, through these deliberations we overcome the problems of the related work and provide answers to the remaining open problems.

2.2. Adequacy of a Materialized View to Answer a Query for the 2D Case

In this section, we provide theoretical and algorithmic results for answering top-*k* queries using materialized views. For reasons of perception and intuition we initially examine the problem of answering top-*k* queries of a relation in the 2-dimensional space. In the next sections we generalize the problem for the *n*-dimensional space as well. 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.

2.2.1. Problem Formulation

Given *a relation R (id, X, Y) a materialized view V (id, X, Y, s) over R having the top-n tuples from R where* $s = w\,(a \cdot x + y)$ *and w, a being positive parameters and a query Q over R having the form of a score function* s_Q *where* $s_Q = w_Q\,(a_Q \cdot x + y)$ *and* w_Q, a_Q *being positive parameters,*

Retrieve *the top-k tuples from R*

Having *the k highest values according to the scoring function of Q.*

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]. In addition, we assume that the weight

factors of the linear scoring function are positive. In case the weight factors are negative, we can always convert the equivalent scoring function to one with positive weight factors with suitable transformations. Thus, without loss of generality we assume the attribute values of a tuple being normalized into the interval [0,1] and the weight factors of the scoring functions of the query as well as the materialized view being positive parameters. This way, any tuple of the relation R can be represented as a point (Figure 2.6). The area that we are interested in is the area that contains all tuples from R, and we call this area the *active area*. The active area is formally defined from the following definition.

Definition 2.2 (Active Area). The rectangle defined by the line segments OX, OY, XR, YR (where $O(0,0)$ $X(1,0)$, $Y(0,1)$, $R(1,1)$) is the *active area* that contains all tuples of a relation a relation $R(ID, X, Y)$ where without loss of generality the domains of X and Y are normalized over the interval [0, 1].

In addition, assume a top-n 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. Then, this equation is characterized by a line $y = a^{-1} \cdot x$. The score of any tuple in R in regards to the view V can be found by projecting the point that represents this tuple over the line that characterizes the view. We define as the *border line* L_V of the view V, a perpendicular line over the line $y = a^{-1} \cdot x$ that splits the active area into two sub-areas. Observe in Figure 2.6 the border line L_V that splits the area into two sub-areas from which the one is actually the area that contains all tuples materialized in the view. Specifically, the sub-area above the border line L_V contains all top-n tuples of the view and we call this sub-area the *extent* of V.

Definition 2.3 (View Border Line L_V). The border line L_V, of a top-n materialized view V having the scoring function $s_V = w_V(a_V \cdot x + y)$ and t_n being the n^{th} tuple of V, is the line drawn perpendicular to the line that describes the scoring function of V ($y = a_V^{-1} \cdot x$) and passing from the point $s_V(t_n)$ (with x_{NV}, y_{NV} being the points where it meets the axes X, Y).

Definition 2.4 (Extent of V). The area defined above the line L_V towards the point $R(1,1)$ (within the active area) is the extent of the materialized view that contains the top-n tuples with respect to V.

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. Similar to the border line L_V we define the border line of the query, this time within the extent of V. Specifically, the border line L_Q of the query, as shown in Figure 2.6, splits the active area into two sub-areas such that the sub-area above the border line L_Q contains all those tuples of R with the higher scores in regards to the query that are also part of the view's extent. In other words, the border line L_Q depicts a border of the active area such that any point above L_Q will e the query's answer and simultaneously will be part of the view's result.

Definition 2.5 (Query Border Line L_Q). The border line L_Q, for a combination of a view V and a query Q, is the line drawn perpendicular to the line that describes the scoring function of the query Q ($y_Q = a_Q^{-1} \cdot x$) and meets the view's border line L_V in one point such that any point of L_Q within the active area belongs to the extent of V.

The sub-area above the border line L_Q within the extent of V is the area that can be proved helpful in order to answer a top-k query by exploiting only the tuples materialized in V. This occurs from the fact that the points belonging above the border line L_Q are all points from the relation that are contained in the materialized view and will definitely be part of the top-k query's answer. We refer to this area as the *safe area*, shown in Figure 2.6 as the shaded area.

Definition 2.6 (Safe Area). The area defined above the border line L_Q towards the point $R(1, 1)$ within the (active area) is called the safe area of the query Q with respect to the materialized view V.

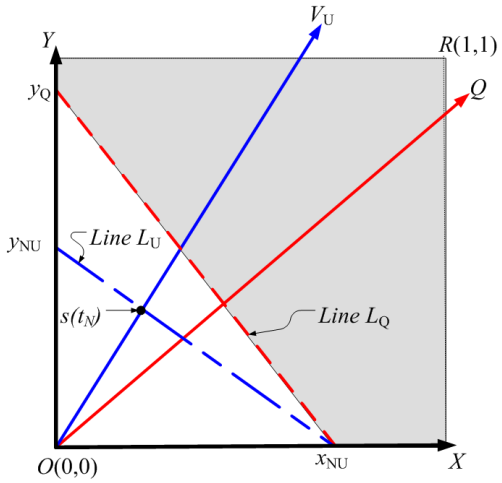


Figure 2.6. Answering a Query Q via a View V_U when the View is "Higher" than the Query.

We will explore the problem of answering a top-k query Q through the tuples materialized in a view 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.

2.2.2. The Case when the View is "Higher" than the Query

In this case (Figure. 2.6), 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 border line of V passing from point $s(t_n)$ (with x_{NU}, y_{NU} being the points where 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 the border line of Q and starts at the point x_{NU}. The safe area of Q with respect to V contains points that belong both to Q and V_U.

Lemma 2.1. 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 2.7, near the X-axis) that (a) does not belong to the extent of 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 2.7. 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. □

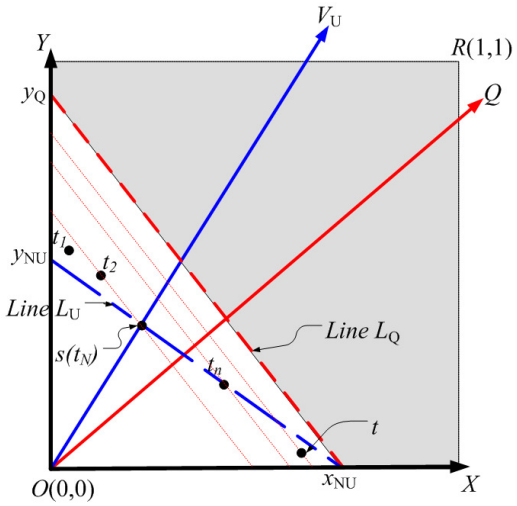


Figure 2.7. Example of Why a View V is Not Always Reliable for Answering a Query Q.

Theorem 2.1. V_U can answer Q if the safe area of Q in regards to V_U contains at least k points.

Proof. We will prove the theorem by contradiction. Assume a tuple t of R (Figure 2.7) 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 safe area 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 2.1 does not always hold, and (b) how can we decide that a point belongs to the safe area. We discuss these two aspects in the following sub-sections.

2.2.3. Strictness of the Suitability Theorem

It is not possible to infer the inverse of Theorem 2.1. Even if the safe area 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 safe 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.

Definition 2.7 (Critical Area). The area in the active region defined by the lines L_V and the line that produces the lowest possible score for Q from the tuples in V is the critical area of Q in regards to V.

The following theorem formalizes the conditions under which a view can answer a query even if it's safe area is insufficient.

Theorem 2.2. 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 critical area of Q with respect to V_U is void of tuples.

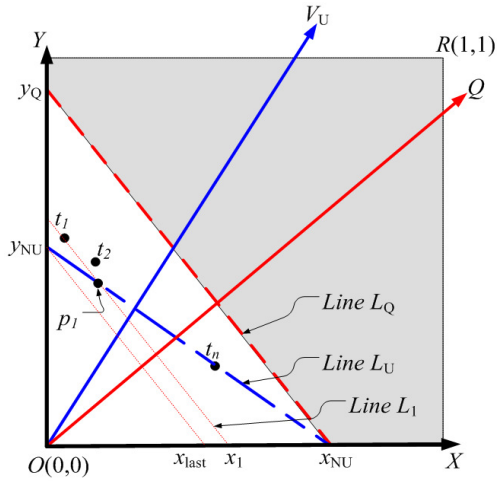


Figure 2.8. At Least k Points in the Safe Area of a View V Make it Reliable for Answering a Query Q.

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 2.8, 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. □

2.2.4. Computation of Offsets and Safe Areas

A technical point has to do with whether a point belongs to the safe 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 safe 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 of

the form $y = -a_U \cdot x + offset$. 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., $x_{NU} = a_U^{-1}(y_n + a_U \cdot x_n)$.

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} (Figure 2.8), 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).

2.2.5. 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 $s_D = w_D(a_D \cdot x + y)$.

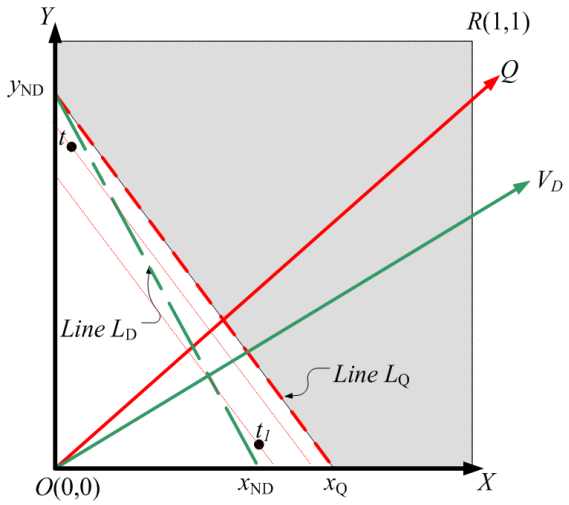


Figure 2.9. The Case where the View is "Under" the Query.

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 area 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 area above the line L_Q if $y_b \geq -a_Q \cdot x_b + y_{ND}$.

2.2.6. 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 (Figure 2.10), line L_V is of the form $y = y_n$. In addition, assume a query Q with scoring function $w_Q(a_Q \cdot x + y) = s_Q$. 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 as usual.

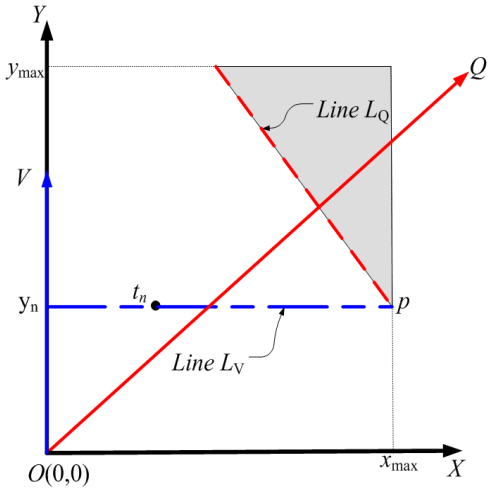


Figure 2.10. Special Case where V is of the Form $s_V = y$.

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.

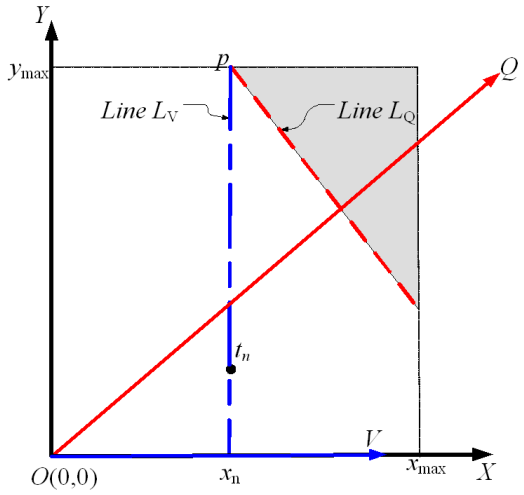


Figure 2.11. Special Case where V is of the Form $s_V = x$.

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.

(ii) Assume a view with a scoring function of the form $s_V = x_V$ (Figure 2.11). 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 $w_Q(a_Q \cdot x + y) = s_Q$. 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.

2D SafArI Algorithm

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 · x + y), k ≤ 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 (α_Q^-1 ≤ α^-1){
3.        compute point x_NU: x_NU = a^-1 (y_n + a·x_n)
4.        define line L_Q as y = -α_Q·x + α_Q·x_NU
5.  }
6.  else{
7.        compute point y_ND: y_ND = y_n + a·x_n
8.        define line L_Q as y = -α_Q·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 ≥ k ) return(true);
15. else return(false);
```

End.

Algorithm 2.1. 2D SafArI Algorithm

2.2.7. Algorithmic Results

Now, we are ready to give the 2D SafArI algorithm (**2D Saf**e **A**rea **I**llation algorithm) an algorithm for deciding view suitability in the 2D case (shown as Algorithm 2.1) that decides whether a 2D query *Q* can be answered by a 2D view *V* and populates *Q* if the test is positive. As Figure 2.12 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 Algorithm 2.1).

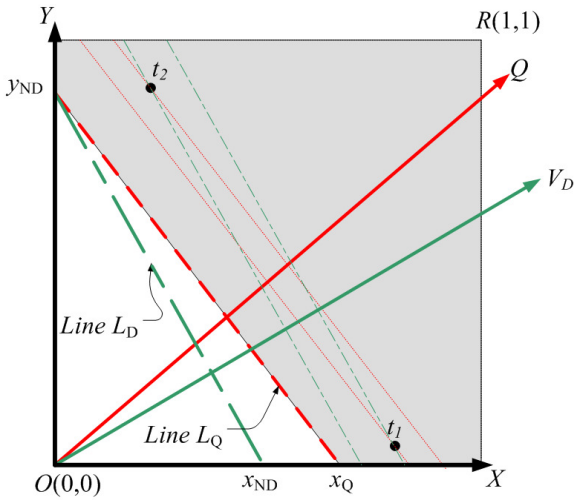


Figure 2.12. All the Safe Area Should Possibly be Exhausted for the Determination of the top-*k* Query Tuples.

2.3. Queries and Views with More than Two Scoring Attributes

The results of the previous sections 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.

2.3.1. Fundamental Results for the n-Dimensional Case

Assume a relation $R(ID, X_1, X_2,\ldots, X_N)$ where without loss of generality the attributes X_i are within the interval [0,1]. All tuples of R can be represented as points over an *N*-dimensional space.

Definition 2.7 (Active Region). The hyper-cube that contains all points of the form $(x_1, ..x_N)$ with $0 \leq x_i \leq 1$ is the *active region* and contains all tuples of a relation a relation $R(ID, X_1, X_2,\ldots, X_N)$.

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 can be found by projecting 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 $s_v(t_n)$.

Definition 2.8 (HyperPlane P_V). The hyper plane P_V, with respect to a top-n materialized view V having the scoring function $s_V = w\ (a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + x_N)$ and t_n being the n^{th} tuple of V, is the hyper plane drawn perpendicular to the line that describes the scoring function of V (L_V: $\frac{x_1}{a_1} = \frac{x_2}{a_2} = \ldots = x_N$) and passing from the point $s_V(t_n)$ (with $x_{1V}, x_{2V}, \ldots, x_{NV}$ being the points where it meets the axes $X_1, X_2, \ldots X_N$ respectively).

Then, the hyper plane 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.

Definition 2.9 (Extent of V SR_{low}). The area defined above the hyper plane P_V towards the point $R(1,\ldots,1)$ (within the active region) is the extent of the materialized view that contains the top-n tuples with respect to V, denoted as SR_{low}.

Figure 2.13 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.

Definition 2.10 (HyperPlane P_Q). The hyper plane P_Q, with respect to a materialized view V and a query Q, is the hyper plane perpendicular to the line that describes the

scoring function of the query Q (L_Q $\frac{x_1}{a_{1Q}} = \frac{x_2}{a_{2Q}} = \ldots = x_N$) and meets P_V in one point such that any point of P_Q within the active region belongs in the sub region SR_{low}.

Definition 2.11 (Safe Area). The area defined above hyper plane P_Q towards the point $R(1,\ldots,1)$ within the active region is called the safe area of the query Q with respect to the materialized view V.

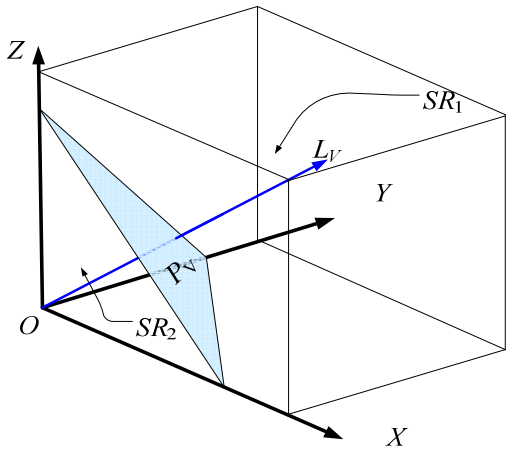


Figure 2.13. The Two Sub-Regions Defined by P_V.

Lemma 2.2. 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: $s_v(t_1) \leq s_v(t_n)$ and $s_Q(t_1) \geq s_Q(t_k)$. 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: $s_v(t_2) \geq s_v(t_n)$ 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: $s_v(t_1) \leq s_v(t_n) \leq s_v(t_2)$ 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 2.3. *V* can answer *Q* if 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. Similarly with Theorem 2.1, since *t* does not belong in *V*, it lies in the sub-region SR_{high}. 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 safe area 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*. □

Much like the case of two dimensions, it is not possible to infer the inverse of the theorem. Even if the safe area 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.

Definition 2.12 (Critical Area). The area in the active region defined by the hyper planes P_V and the hyper plane 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*, is the critical area of *Q* in regards to *V*.

Theorem 2.4. 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 critical area of *Q* in regards to *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_{Q1} is the hyperplane that is perpendicular to L_Q and passes through point t_1. If the critical area 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*. □

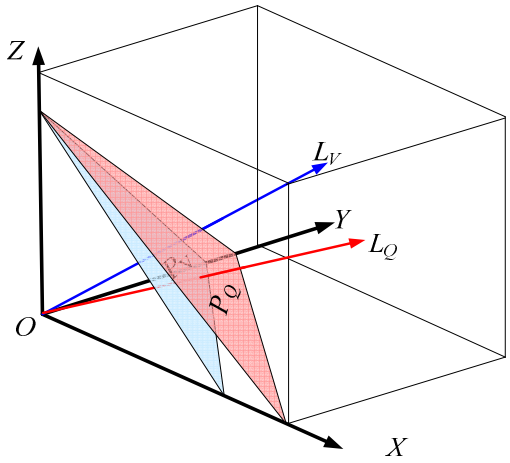


Figure 2.14. Example of Why a View V is Not Always Reliable for Answering a Query Q.

2.3.2. 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_{last}. t_{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 is 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 $w(a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + x_N) = s_v(t_n)$. 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} (0, …, x_{iV},…,0). Since x_{iV} belongs in P_V its coordinates are x_{iV} (0,…, $s_v(t_n)\cdot (w\cdot a_i)^{-1}$,…0) where all are equal to zero except the i-th coordinate. Similarly, it could be any other axis X_i. The hyperplane P_Q is defined by the equation w_Q ($a_{1Q} \cdot x_1 + a_{2Q} \cdot x_2 + \ldots + x_N$) = 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 $w_Q \cdot$

$s_v(t_n) \cdot (w \cdot a_i)^{-1}$ and the hyperplane P_Q is defined from the equation $a_{1Q} \cdot x_1 + a_{2Q} \cdot x_2 + \ldots + x_N = s_v(t_n)\ (w \cdot a_i)^{-1}$. Assume a tuple $t_b\ (x_{1b}, x_{2b}, \ldots, x_{Nb})$. Tuple t_b belongs to the safe area if $x_{Nb} \geq -a_{1Q} \cdot\ x_{1b} - a_{2Q} \cdot\ x_{2b} - \ldots + s_v(t_n) \cdot \cdot (w \cdot a_i)^{-1}$.

2.3.3. Algorithmic Results

Now, we are ready to give the SafArI algorithm (Algorithm 2.2) that decides whether Q can be answered by V and populates Q if the test is positive.

The computation of where the hyperplanes P_Q and P_V first meet on one of the n axis and thus the safe area for n-dimensions is computed by solving a linear problem with the usage of the simplex method. Therefore, the value s_Q that determines the hyperplane P_Q is computed through the solution of the following linear problem: The objective function is to maximize $w_Q\ (a_{1Q} \cdot x_1 + a_{2Q} \cdot x_2 + \ldots + x_N)$ under the constraints

$$\text{s.t. } w(a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + x_N) \leq s_v(t_n)$$

$$0 \leq x_i \leq 1, \text{ for all } i \in \{1, \ldots, N\}$$

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

2.4. Working with More Than One Views

In this section we deal with the problem of answering top-k queries through the usage of more than one materialized views. Firstly, we show that the usage of two materialized views and specifically the union of the safe areas of two views do not add better guarantees for the answering of a query. Secondly, we exploit the problem of answering a top-k query by parallelizing its process and assigning different parts of the query's answer to a different view and then uniting the results.

2.4.1. Safe Area Containment 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*?

SafArI Algorithm

Input: A materialized view V (ID, X_1, X_2,…, 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 (ID, X_1, X_2,…, 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

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, $t_n(x_1,x_2,\ldots,x_N)$=V[n].
2. Define hyper-plane P_Q
   Solve linear problem:
   max s_Q
   s.t. $s_V \leq s_V(t_n)$
3. for all tuples of V {
4. Compute s_Q(V[i])
5. if (s_Q(V[i]) belongs above hyper-plane P_Q) counter++ ;
6. }
7. if (counter $\geq$ k) return(true);
8. Else return(false);

End

Algorithm 2.2 SafArI Algorithm

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_j 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. For reasons of intuition we illustrate in Figure 2.15

a query Q encompassed by two materialized views in the 2D space. 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 2.15 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 2.5. 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 safe areas of the views.

Proof. Since the border 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. □

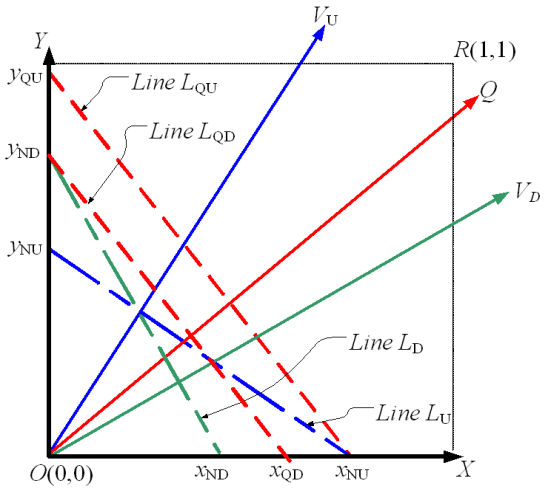


Figure 2.15. A Query Q with One View on Either of its Sides, V_U for the Upper Side and V_D for the Lower Side.

2.4.2. 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.

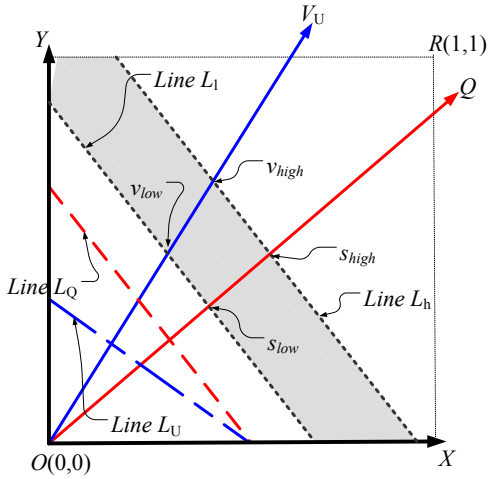


Figure 2.16. The Active Zone for the Range $s_{low,\ high}$ of Query Q within its Safe Area over View V_U.

Theorem 2.6. Assume a materialized view V (with a line $V_U : y = a_U^{-1} \cdot x$) 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 $[s_{low}, s_{high}]$ (with $s_{low} \leq s_{high}$ and s_{low} and s_{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 = $-a_Q \cdot x + s_{low} \cdot \sqrt{a_Q^2 + 1}$, L_h : y = $-a_Q \cdot x + s_{high} \cdot \sqrt{a_Q^2 + 1}$.

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_{low} and s_{high} from the beginning of the axes. For point p_h (x_h, y_h) corresponding to s_{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 $y = -a_Q \cdot x + offset$, with *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 $y_h = -a_Q \cdot x_h + offset$) and solve the system of equations that also comprises the equation of line Q. The solution gives the equations of the theorem. □

Observe that it is indifferent whether V is on the upper or lower side of Q, since we have carefully selected the scores s_{low} and s_{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 Figure 2.16. We need to identify tuples that have a score with respect to V's scoring function within the range [v_{low}, v_{high}]. Unfortunately, we cannot solely rely on the score bounds of v_{low}, v_{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_{low}, v_{high}].

Lemma 2.3. Given the values s_{low}, s_{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:

$$v_{low} = s_{low} \cdot \frac{a_Q^2 + 1}{1 + a_Q \cdot a_U} \cdot \sqrt{a_U^2 + 1}\,,\ v_{high} = s_{high} \cdot \frac{a_Q^2 + 1}{1 + a_Q \cdot a_U} \cdot \sqrt{a_U^2 + 1}$$

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

   $s_t \in [v_{low}, v_{high}]$,

   $y_t \geq -a_Q \cdot x_t + s_{low} \cdot \sqrt{a_Q^2 + 1}$,

   $y_t \geq -a_Q \cdot x_t + s_{high} \cdot \sqrt{a_Q^2 + 1}$
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.

Algorithm 2.3. Algorithm Compute Query Extent

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_{high}. We can compute v_{low} similarly. □

Theorem 2.7. 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:

$s_t \in [v_{low}, v_{high}]$ with this range computed via the above lemma,

$y_t \geq -a_Q \cdot x_t + s_{low} \cdot \sqrt{a_Q^2 + 1}$,

$y_t \geq -a_Q \cdot x_t + s_{high} \cdot \sqrt{a_Q^2 + 1}$

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.

2.5. Experiments

In this section, we report on the experimental assessment of the usage of materialized views to answer top-k queries. We have conducted two sets of experiments. The first set focuses on the algorithm for the 2 dimensional space, whereas the second set of experiments involves the n dimensional space.

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 be used for answering a newly posed top-k query through the exclusive usage of a materialized view.
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).

2.5.1. Experimental Method for 2D

In this set of experiments, all tests involve a relation $R(tid,X,Y)$. All the queries were fully answered and then used as materialized views for the subsequent queries.

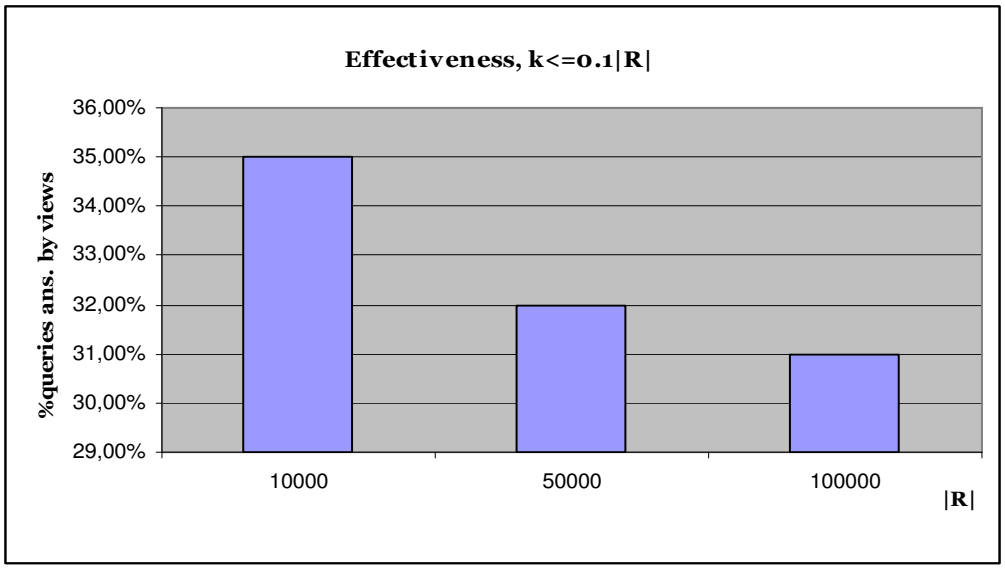


Figure 2.17. Percentage of Views Used for 100 Queries.

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 Table 2.1.

For this set of 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 2.1. Experimental Parameters for 2D.

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

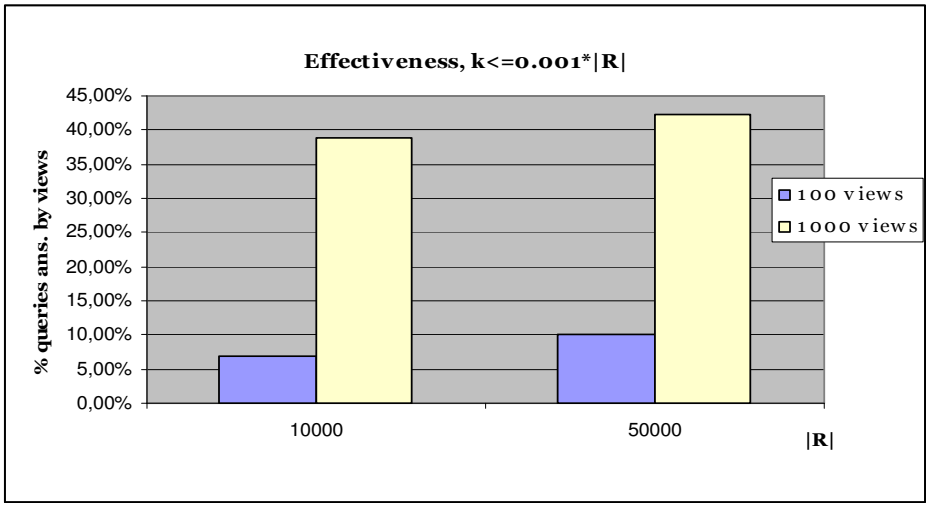


Figure 2.18. Percentage of Views Used for Different Time Spans (Numbers of Posed Queries).

Effectiveness

The effectiveness of the method is depicted in Figure 2.17 and 2.18. Figure 2.17 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 2.18, 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 2.17, 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 2.18, concerns the effectiveness of the method over time. So, in Figure 2.18, 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.

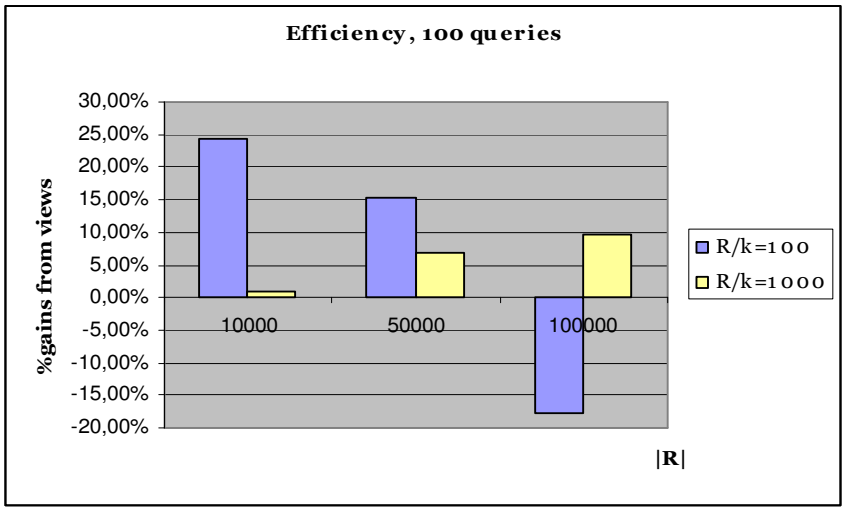


Figure 2.19. Time Savings from the Usage of Queries for Different Database Sizes and Requested Results.

R	k	D/k	% views used	Total time without views (sec's)	Total time via views	Total opponent time	% improved
10000	100	100	35	0.35	0.006	0.09	24.28
10000	10	1000	7	0.07	0.00001	0.0007	0.99
50000	500	100	32	4.32	0.39	1.06	15.39
50000	50	1000	10	1.06	0.0001	0.07	6.82
100000	1000	100	31	12.03	4.59	2.45	-17.79
100000	100	1000	11	2.68	0.003	0.26	9.66

Figure 2.20. Detailed Information for the Efficiency of the Method in Time Savings.

Efficiency

The efficiency of the method over random data is depicted in Figure 2.19. We vary two parameters, 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 shown in Figure 2.20.

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%. 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.

2.5.2. Experimental Method for n-D

The second set of experiments involves the testing of the algorithm for the *n* dimensional space. In this set of experiments we have made use of synthetic as well as real data sets. All synthetic experiments involve a relation $R(\ tid, X_1, \ldots X_n)$ of various distributions and dimensionality. For this set of experiments we have used a Core 2 CPU at 2.53 GHz with 3.12GB memory. All the implementations were made using BerkeleyDB and its C API.

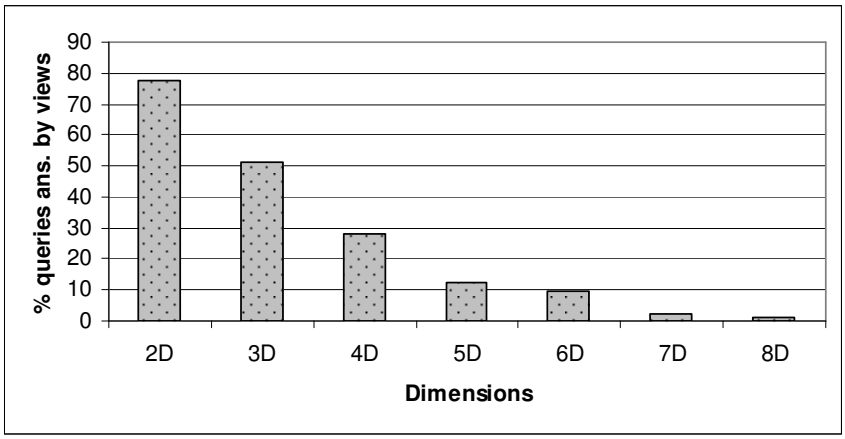
Table 2.2 Experimental Parameters for Synthetic *N*-D.

Data Distribution	*Distr*	Random, Correlated, Anticorrelated
Data dimensionality	*D*	2, 3, 4, 5, 6, 7, 8
Max size of top-k tuples	*Max_k*	25, 100

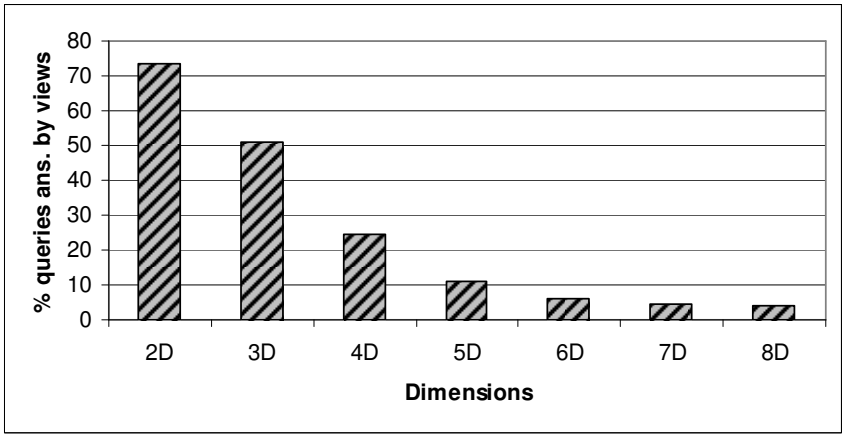
Synthetic Data Sets

The synthetic datasets are of three optional distributions: Random, Correlated and Anticorrelated. Random datasets are generated such that the attributes of the tuples are independent of each other following a uniform distribution. The Correlated and Anticorrelated datasets are generated as described in [BoKS01]. In the correlated datasets the attribute values of the tuples are positive correlated, whereas in the anticorrelated datasets, one attribute value is large and the remaining attribute values are small. The datasets are of dimensionality *d* that varies from 2 to 8. We generate views and queries with random coefficients and result size (*k*). The weights of the scoring function of the views and the queries all add to 1. The important parameters that we have experimented with are: (a) the distribution of the relation (*Distr*), (b) the dimensionality *d* of the relation and, (c) the maximum number of the requested tuples (*max_k*). The size of the relation is 1 million records, the number of views materialized is set to be 100 and the queries requested are 1000. The parameters for this set of experiments are listed in Table 2.2.

The effectiveness of the method is depicted in Figures 2.21, 2.22 and 2.23. In these figures we present the percentage of queries that were answered by our method over the set of 100 prematerialized views. For all the figures the (a) part depicts the percentage of queries answered when both views and queries request top-*k* tuples, where k is randomly generated with maximum value 100, and the (b) part depicts the percentage of queries answered by our method when views and queries request top-*k* tuples with maximum value of *k* being 25. For the random and anticorrelated dataset we observe that the percentage of queries answered decreases as the dimensionality of the dataset increases. In the correlated dataset the percentage of queries answered by our method seems rather constant and independent of the dimensionality and almost 100%. In addition, when comparing figures (a) and (b) in each distribution dataset we observe that the percentage of queries answered when the dimensionality increases is similar and rather regardless of the *max_ k* value.



(a)Percentage of queries answered with max_k 100

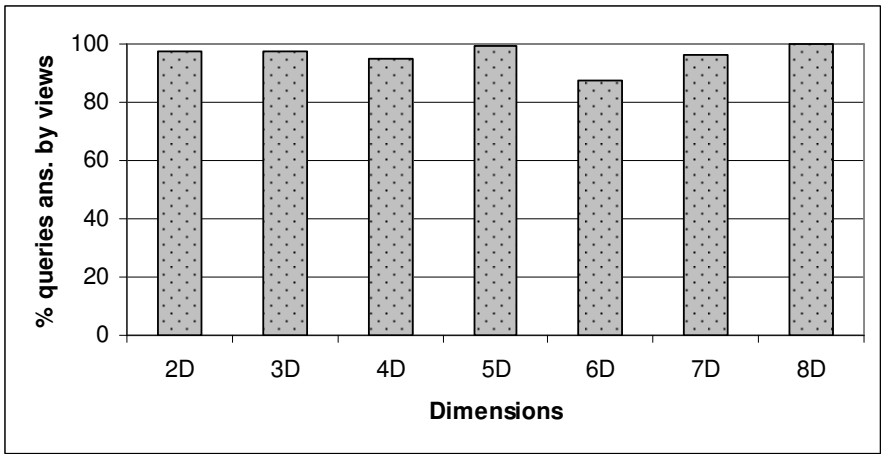


(b) Percentage of queries answered with max_k 25

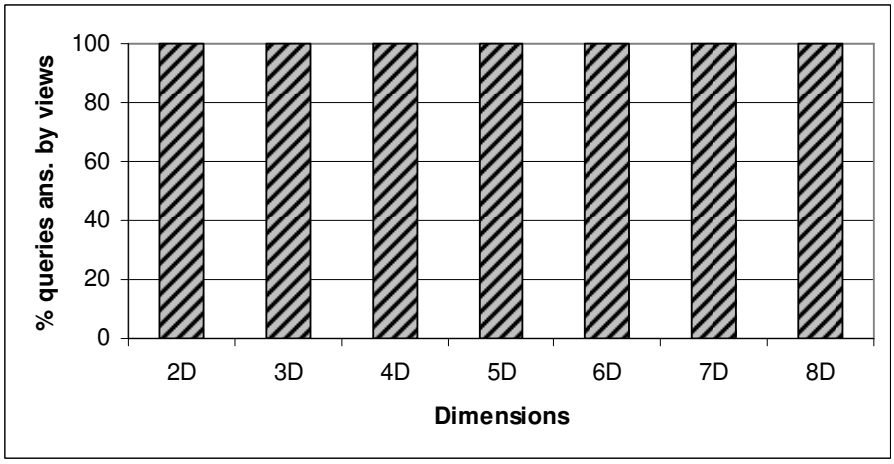
Figure 2.21. Percentage of Queries Answered for Random Data.

The efficiency of the method is depicted in Figures 2.24, 2.25, 2.26. We vary again the distribution, the dimensionality and the maximum possible number that *k* can take, and we assess the improvement in time when comparing our method with the opponent. Specifically, we measure the percentage time improvements of our method when compared to the opponent. The detailed numbers (including total query times) are depicted in Tables 2.3, 2.4 and 2.5 for the distributions Random, Correlated and Anticorrelated respectively. By observing Figure 2.24 and 2.26 we can see that the time savings for these datasets decrease while the dimensionality increases. On the contrary, in Figure 2.25 we can observe that the time savings of our method seem to increase when the dimensionality increases. However, in Figure 2.25 (b) the time savings for 2, 3, and 4 dimensions when max *k* is 25 are negative showing that the

opponent outperforms our method. In conjunction with Table 2.4 we can observe that for the correlated data the opponent as well as our method needs a small amount of time to compute the results. For the same parameters (i.e., *d*, *max_k*) but for random and anticorrelated data the time needed by the opponent (see Table 2.3 and 2.5) in comparison to the correlated data is much greater.

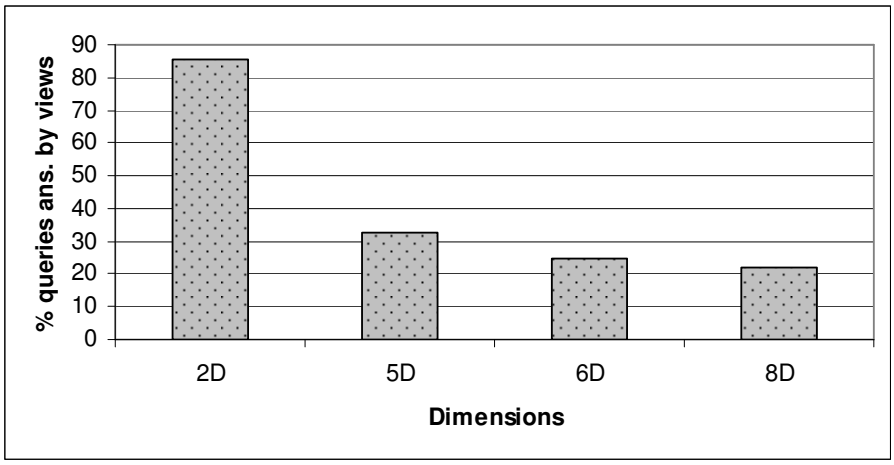


(a) Percentage of queries answered with max_k 100

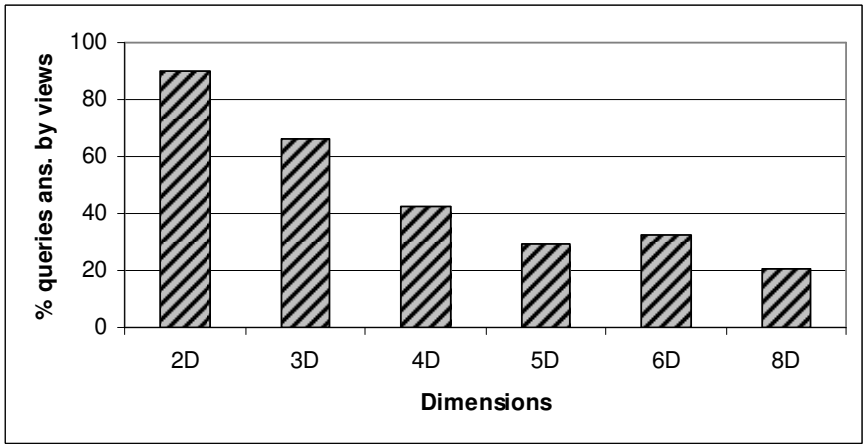


(b) Percentage of queries answered with max_k 25

Figure 2.22. Percentage of Queries Answered for Correlated Data.

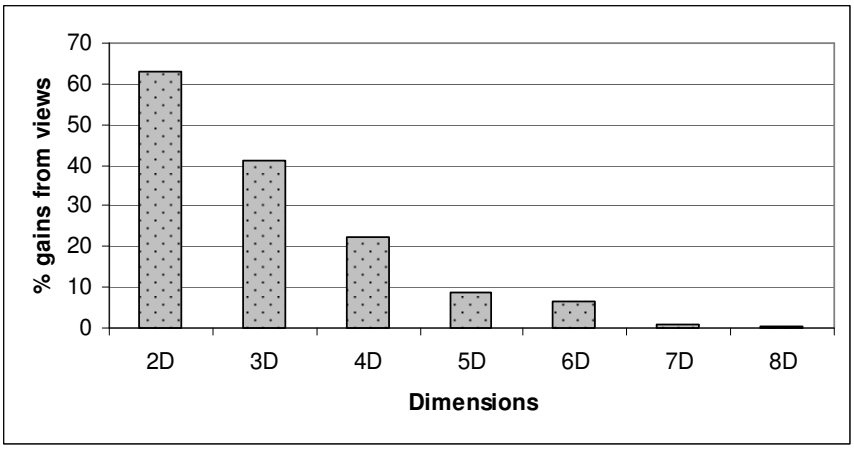


(a) Percentage of queries answered with max_k 100

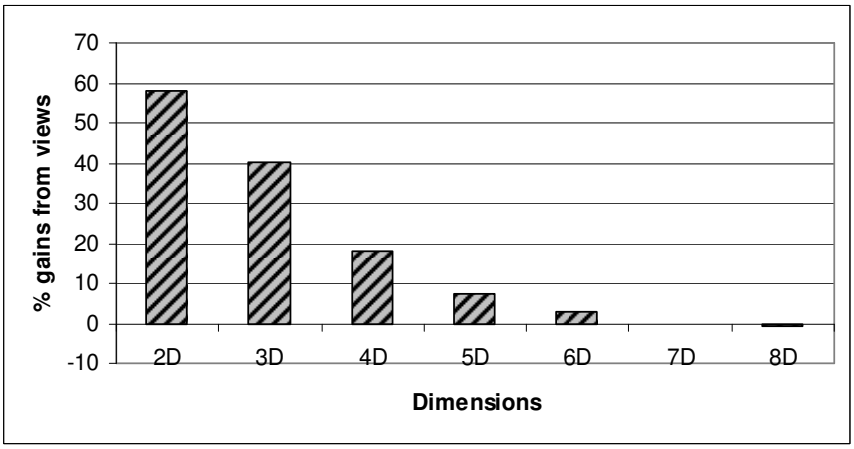


(b) Percentage of queries answered with max_k 25

Figure 2.23. Percentage of Queries Answered for Anticorrelated Data.

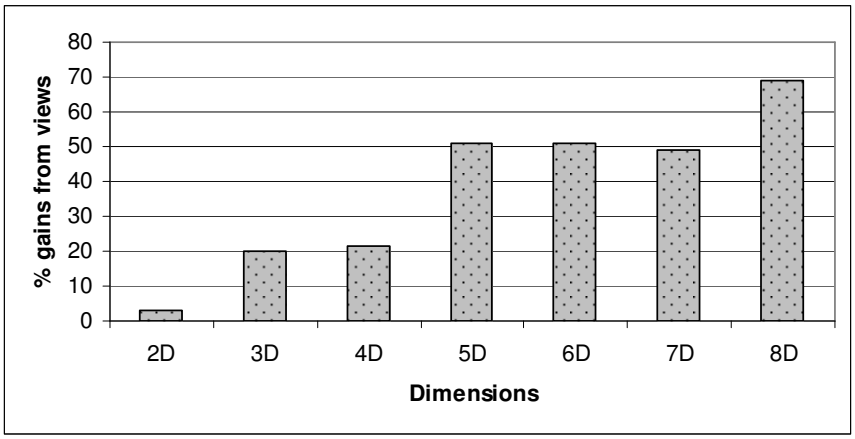


(a) Time savings from the usage of views with max_k 100

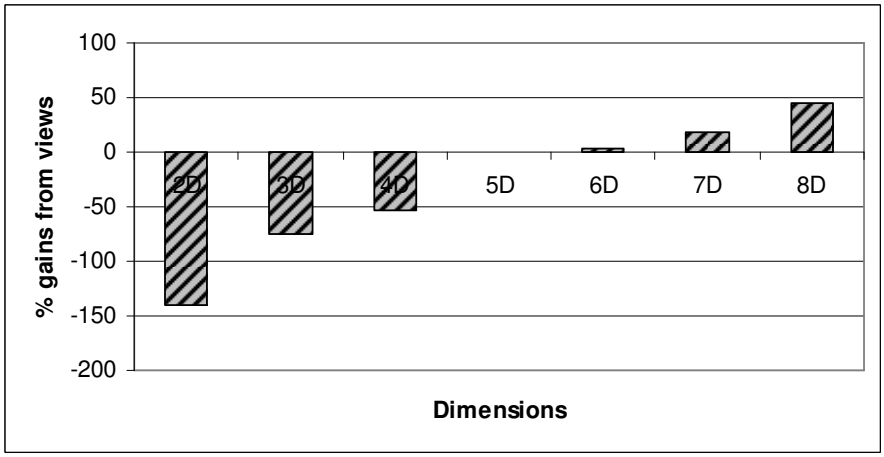


(b) Time savings from the usage of views with max_k 25

Figure 2.24. Time Savings from the Usage of Views for Random Data.

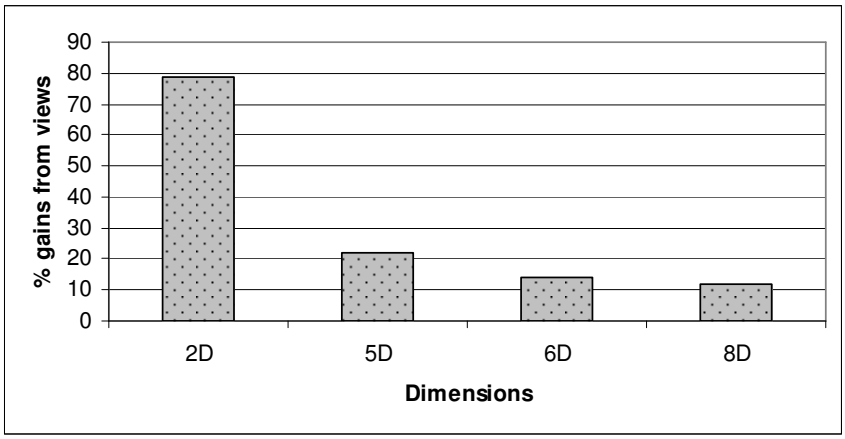


(a) Time savings from the usage of views with max_k 100

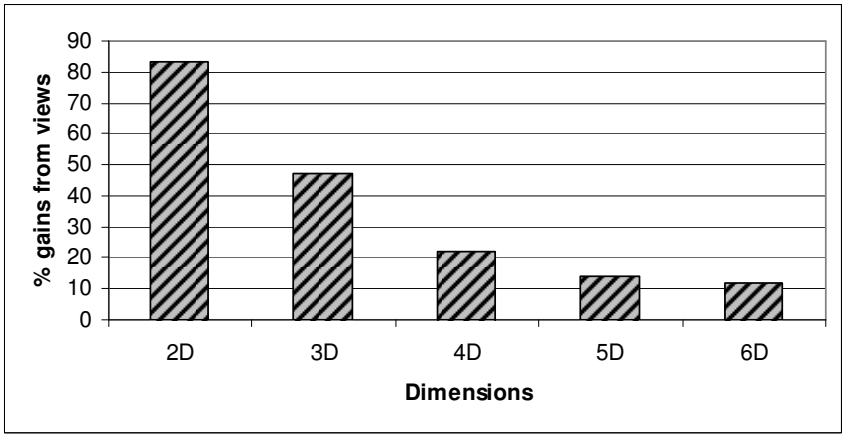


(b) Time savings from the usage of views with max_k 25

Figure 2.25. Time Savings from the Usage of Views for Correlated Data.



(a) Time savings from the usage of views with max_k 100



(b) Time savings from the usage of views with max_k 25

Figure 2.26. Time Savings from the Usage of Views for Anticorrelated Data.

Table 2.3 Absolute Times and Time Savings for Random Data.

d	Max_k	% Q answered	Total opponent time (sec's)	Total time of our method (sec's)	Total opponent time for queries answered via views	Total time of our method for queries answered via views	% improved
2	100	77.8	109.31	40.47	79.78	5.77	62.97
2	25	73.4	59.62	25.01	41.98	3.17	58.03
3	100	51.3	674.93	397.52	304.63	5.13	41.10
3	25	51.2	393.97	235.62	177.98	3.62	40.19
4	100	28	1872.74	1453.43	456.89	3.97	22.39
4	25	24.7	1337.10	1095.07	285.36	4.06	18.10
5	100	12.2	3630.21	3319.41	360.97	2.75	8.56
5	25	11	2822.67	2620.57	254.23	2.94	7.15
6	100	9.5	5364.30	5013.96	405.73	1.52	6.53
6	25	6	4108.59	3994.96	168.48	2.01	2.76
7	100	2.4	9121.94	9052.71	128.86	0.96	0.75
7	25	4.3	7444.47	7457.33	56.11	2.35	-0.17
8	100	1.4	12610.82	12569.41	101.83	0.65	0.32
8	25	4.1	10353.91	10407.03	8.93	2.68	-0.51

Table 2.4 Absolute Times and Time Savings for Correlated Data.

d	**Max_k**	**% Q answered**	**Total opponent time (sec's)**	**Total time of our method (sec's)**	**Total opponent time for queries answered via views**	**Total time of our method for queries answered via views**	**% improved**
2	100	97.8	1.73	1.68	1.65	1.60	2.96
2	25	100	0.47	1.15	0.47	1.15	-140.73
3	100	97.7	2.45	1.96	2.33	1.71	19.88
3	25	100	0.69	1.22	0.69	1.22	-74.76
4	100	94.8	3.01	2.36	2.69	1.67	21.64
4	25	100	0.81	1.25	0.81	1.25	-52.86
5	100	99.2	3.83	1.88	3.77	1.78	50.92
5	25	100	1.26	1.25	1.26	1.25	0.48
6	100	87.8	4.66	4.29	3.56	1.63	7.91
6	25	100	1.36	1.31	1.36	1.31	3.48
7	100	96.1	5.29	2.68	4.87	1.82	49.16
7	25	100	1.64	1.34	1.64	1.34	18.42
8	100	99.9	6.56	2.03	6.55	2.01	69.01
8	25	100	2.53	1.40	2.53	1.40	44.67

Table 2.5 Absolute Times and Time Savings for Anticorrelated Data.

d	max_k	% Q answered	Total opponent time (sec's)	Total time of our method (sec's)	Total opponent time for queries answered via views	Total time of our method for queries answered via views	% improved
2	100	85.5	7056.78	1518.47	5545.90	3.82	78.48
2	25	90.2	6823.17	1161.88	5672.06	7.33	82.97
3	25	66.1	12532.81	6580.70	5974.25	8.27	47.49
4	25	42.7	15939.83	11366.84	4608.92	5.84	28.68
5	100	32.5	18807.73	14632.41	4217.97	5.02	22.20
5	25	29.2	18299.33	14907.86	3437.27	6.73	18.53
6	100	24.7	22298.06	19158.12	3184.52	4.30	14.08
6	25	32.7	21914.17	17612.97	4344.64	6.08	19.62
7	100	22.2	26247.56	23073.72	3218.96	5.24	12.09
7	25	20.4	26138.95	23645.24	2545.39	5.22	9.54
8	100	85.5	7056.78	1518.47	5545.90	3.82	78.48
8	25	90.2	6823.17	1161.88	5672.06	7.33	82.97

.Table 2.6 Experimental Parameters for Synthetic *N*-D.

Number of mat. Views	|V|	100, 500
Number of queries	|Q|	100, 1000
Max size of top-k tuples	max_k	25, 50, 100

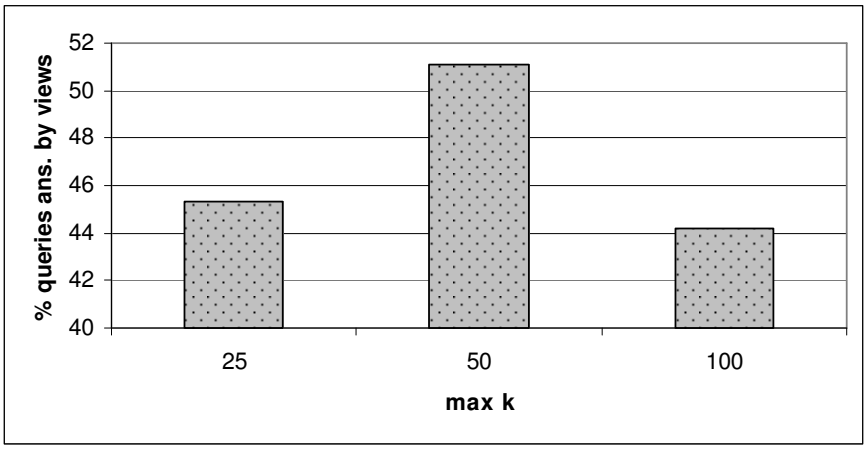
Real Data Sets

To demonstrate the usefulness of our methods, we ran our algorithm on a real data set, Household data set, which is publicly available from the ipums (http:// www.ipums.org) and has been frequently used in the related literature. This dataset contains about 4 million tuples with 5 attributes. Again, we generate views and queries with random coefficients and result size (k) where weight factors all add to 1. The important parameters that we have experimented with are: (a) the number of materialized views $|V|$, (b) the number of queries asked $|Q|$, (c) the maximum number

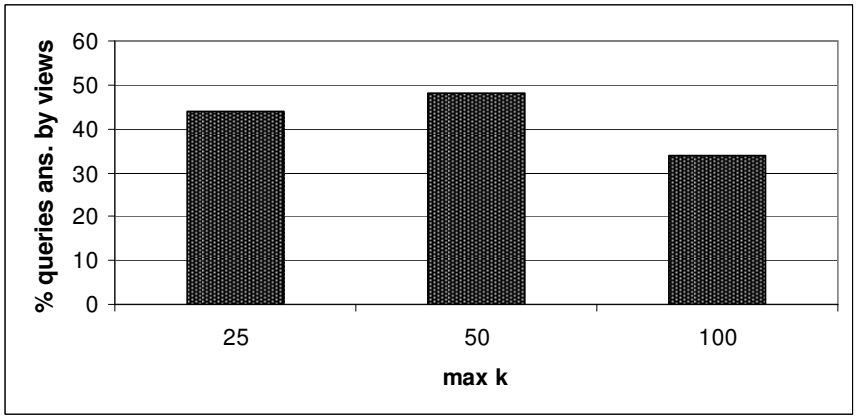
of the requested tuples *max_k*. The parameters for this set of experiments are listed in Table 2.6

In Figure 2.27 we can see the percentage of queries answered by our method for (a) 1000 queries over 100 views, (b) 100 queries over 100 views and (c) 1000 queries over 500 views. We can observe that in all three sets the percentage of queries answered are above 35%. In addition, in Figures 2.27 (a) and (b) which demonstrate the percentage of queries over 100 views, we see that the percentages of queries answered are similar for each *max_k*. In the third figure, where the number of views is greater (i.e., 500) we see that the percentage of queries answered are higher and above 60%, something reasonable due to the greater possibility of a query being answered from a greater set of possible views.

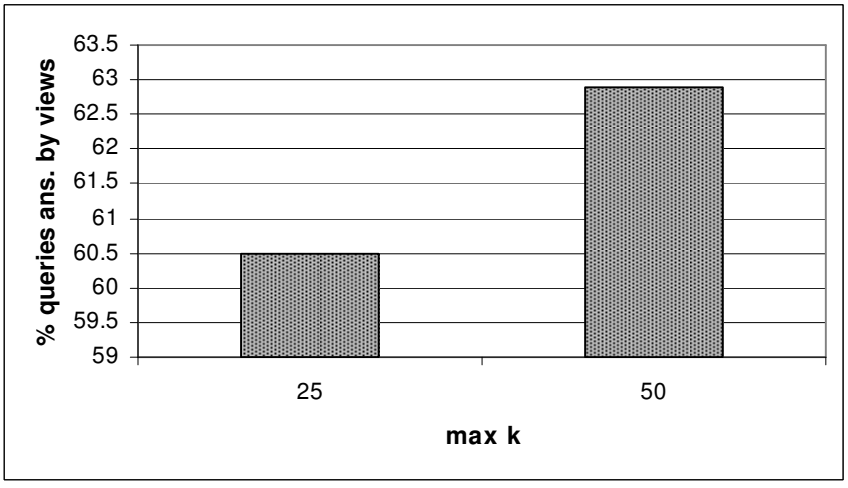
In Figure 2.28 we observe the time savings of our method over the opponent for the three sets of experiments: (a) 1000 queries over 100 views, (b) 100 queries over 100 views and (c) 1000 queries over 500 views. Again, Figure 2.28 presents the time savings for the three *max_k* values for the real dataset. We can observe that the time savings mainly are around 20%. The time savings of the third figure are quite smaller something that can be explained due to the greater number of possible views. This is reasonable since there are more views that should be checked until we can conclude whether a query can be answered by a view.



(a) Percentage of 1000 queries answered over 100 views

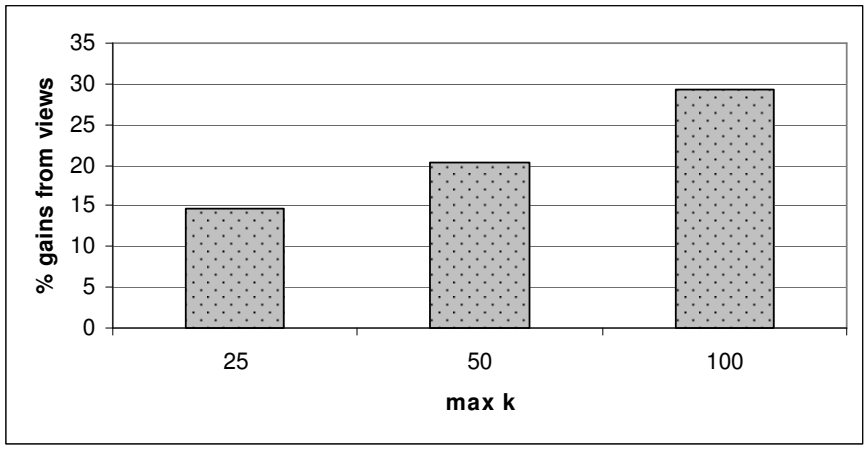


(b) Percentage of 100 queries answered over 100 views

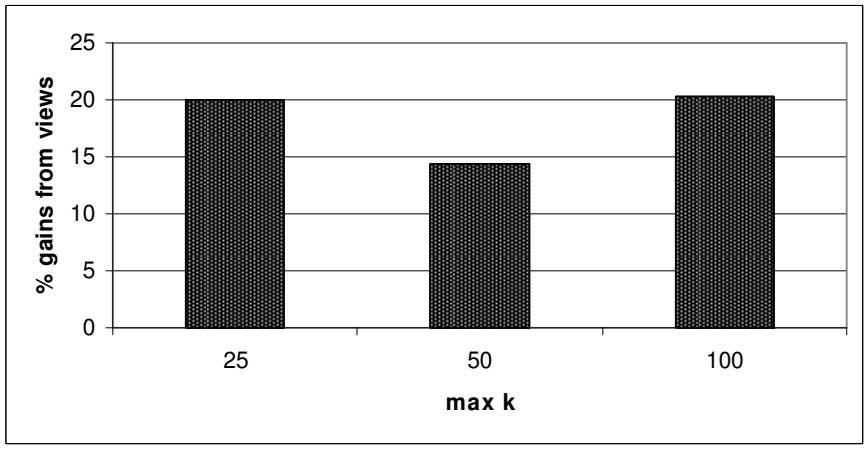


(c) Percentage of 1000 queries answered over 500 views

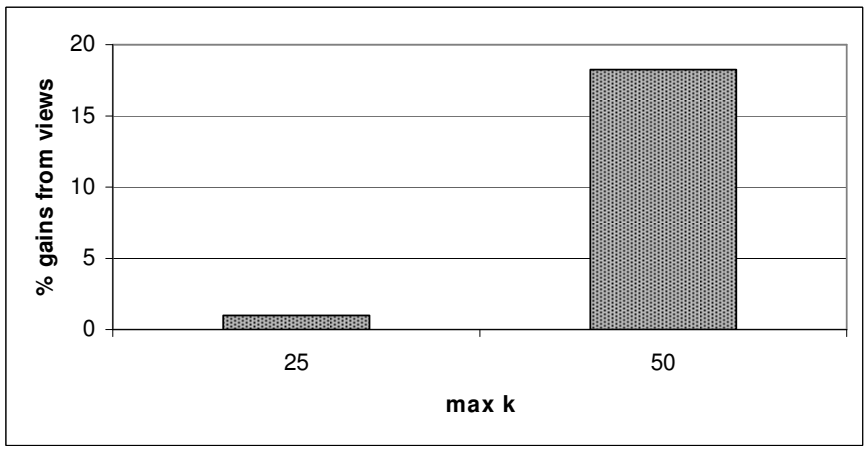
Figure 2.27 Percentage of Queries Answered for Real Dataset.



(a) Time savings for 1000 queries over 100 views



(b) Time savings for 100 queries over 100 views



(c) Time savings for 1000 queries over 500 views

Figure 2.28 Time Savings of Our Method for Real Dataset.

2.6. Chapter Summary and Findings

In this Chapter we have provided theoretical guarantees of the suitability of a materialized ranked view for the answering of a top-*k* query. To this end, we have introduced the notion of *safe area* of a query in regards to a view and provided the respective suitability theorem. In addition, we have proved that the theorem is strict in the sense that it cannot be inversed. In other words, we have proved that even if the safe area is not eligible for answering a top–*k* query, still the view may be suitable for answering a query and described this through the notion of the *critical area*. According to these theoretical establishments for the case of 2-D spaces as well as for the case of multidimensional spaces, we have provided algorithmic results for the answering of a top-*k* query through the usage of a materialized view, namely the 2D SafAri algorithm and the SafArI algorithm. Moreover, we have theoretically proved that the safe areas of a query in regards to more than one views do not offer further usefulness for answering the query compared to the safe area of a single view. We have also discussed the issue of providing partial results for a query via a materialized view by splitting the range of score into appropriate sub-ranges and provided the Compute Query Extent Algorithm. This way, different parts of the query answer can be obtained in parallel, by distributing their processing to different servers.

We have tested our methods for their efficiency and effectiveness through a set of experiments over synthetic as well as real datasets. The first set of experiments concerned the 2D SafArI Algorithm, where the effectiveness of the method proved to be quite stable and ranged around 30-35%. The efficiency of our method is shown to increase consistently for reasonable *k*'s of size 0.1% of the dataset size and rise up to 24% for large relations. The second set of experiments concerned the *N*-D case. The effectiveness of our method was counted as the percentage of queries answered from a set of materialized views. For the synthetic datasets, the effectiveness of the method seemed to be affected by the dimensionality for the random and anticorrelated datasets whereas for the correlated datasets the effectiveness was rather constant around 100%. The effectiveness of our method was also tested over a real dataset and proved to be above 35% in all scenarios and increased significantly when the number of materialized views increased. The efficiency of our method showed again an influence from the dimensionality, where for the random and anticorrelated datasets

the efficiency decreased while the dimensionality was increased. However, for the correlated datasets the efficiency increased when the dimensionality was increased. As for the efficiency of our method over the real dataset, this appeared to be around 20% in terms of time savings over the state of the art.

CHAPTER 3. MAINTENANCE OF TOP-K MATERIALIZED VIEWS

View materialization is typically used for increasing the efficiency of query answering. However, this speed-up comes at a price. Remember that in our *view maintenance* 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 materialized views resulting in better performance than making use only of the base relation from the database. As typically happens with materialized views, though, *when the source relation is updated, we need to refresh the contents of all the materialized views in order to reflect the most recent data*.

Before proceeding, we present a motivating example to contextualize our discussion. Consider a database containing data about stores, products and customers visiting a shopping center near the metro station. When a train arrives, several potential customers arrive with it, at the same time though, there is a massive departure of existing potential customers due to the train's departure. We assume a pervasive environment, where customers are equipped with wireless devices and connect to the

shopping center's server as they enter the building. Assume a relation *Customer* (*c_id*, *c_name*, *c_age*, *c_income*) as well as accompanying relations with the customer's profile, sales history, etc. For a salesman that needs to send the appropriate advertisements, it is important to know which customers are the top-*k* ones according to their characteristics. To achieve this, salesmen use queries that have scoring functions over customer data. For example, assume a salesman wants to advertise a new gadget about mobile phones. The salesman needs to create a profile for the new product, or register the product in an existing profile. The profile includes a formula that assigns a score for a potential customer according to several distance functions and matching of the gadget's and the customer's characteristics. To speed up things, it is reasonable to search for the top-*k* customers in order to send them the advertisement. When a train departs, many customers leave the shopping center; still, the top-*k* list of candidates per product must be maintained so that the remaining possibly interested clients are notified. Consequently, the top-*k* customer lists should be maintained when updates occur in the relation of customers.

The two main problems that pertain to the maintenance of materialized views are (a) the correct and efficient maintenance of a single view when updates occur to the base relation, and (b) the generalization of the maintenance problem for a large number of materialized views. Remember that, given a relation R (tid, A_1, A_2,..., A_m) 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: $f : dom(A_1) \times \ldots \times dom(A_m) \rightarrow \Re$.

Maintaining a single top-*k* materialized view. Concerning the problem of maintaining a single view, the first –and only- attempt that we are aware of is [YYY+03]. To sustain the update rate at the source relation without having to fully re-compute the materialized views, [YYY+03] maintain k_{max} tuples (instead of the necessary k) and perform *refill* queries whenever the contents of the materialized views fall below the threshold of k tuples. Yet, the approach of [YYY+03] suffers from the following problems: (a) the method is theoretically guaranteed to work well only when insertions and deletions are of the same probability (in fact, the authors deal with updates in their experiments), (b) there is no quality-of-service guarantee

when deletions are more probable than insertions. In this chapter, we compensate for these shortcomings by providing a method that is able to provide quality guarantees when the deletion rate is higher than the insertion rate. The case is not so rare if one considers that the number of persons logged in a web server or a portal presents anticipated high peaks and valleys at specific time points or dates. The first contribution of our work is to deal with these phenomena efficiently. The solution to the problem is not obvious for the following reasons. First, even if the value distributions of the attributes that participate in the computation of the score are known individually, it is not possible to compute the distribution of their linear combination, i.e., the score (unless they are stable probabilities – e.g., Normal, Cauchy). Second, even if we extend k with extra tuples to sustain the incoming stream of updates that eventually affects the top-k materialized view, the extra tuples increase the possibility that an incoming source update might affect the view, thus resulting in the need to recursively compute this extension. Finally, we need to accommodate statistical fluctuations from the expected values. To resolve all the above, we provide a principled method that operates independently of the statistical properties of the data and the characteristics of the update streams. The method comprises the following steps: (a) a computation of the rate that actually affects the materialized view, (b) a computation of the necessary extension to k in order to handle the augmented number of deletions that occur and (c) a fine tuning part that adjusts this value to take the fluctuation of the statistical properties of this value into consideration.

Maintaining a set of top-*k* materialized views. The problem of maintaining multiple materialized views is quite important. Its most prominent occurrence has to do with the situation where incoming queries are cached and treated as materialized views to efficiently support the answering of subsequent queries. The problem is hard if we assume that we need all the materialized views to be refreshed every time the source relation undergoes a change. A first workaround concerns the typical warehouse solution of collecting individual updates to larger batches that can be processed much more efficiently than treating each update one tuple at a time. Still, even in this setting, we would like to avoid visiting every view for every tuple. Two extra problems that occur are (a) it is not sufficient to simply include the appropriate tuples in the extent of a materialized view, but we need to compute their score and position

them appropriately in this extent (so, the sharing of tuples between views does not relieve us a lot from the overheads) and (b) we cannot solve the problem by sorting the tuples by their value over a single attribute, since the scoring function takes several attributes into consideration. A possibility that opens is to be able to prune data from the batch when we can infer that they need not be checked against a certain view. So, we develop mathematical guarantees that can decide whenever the current contents of a view need to be updated from a certain batch of modifications, when we know that another view has been affected by this same batch. We assume that the tuples in the extent of our views include (a) the tuple identifier of the tuple in the base relation, (b) the scoring attributes (needed for the management of updates) and (c) its score in the view. In our method, we introduce the idea of *nucleation* between views, which is quite similar to inclusion: a view V_2 nucleates another view V_1, whenever all tuples of the former belong to the extent of the latter, with the exception of their scores. The decision for this kind of inclusion is not straightforward; to avoid checking all the extents of two views we employ a geometric representation of the score function and the tuples of the two views and decide on the nucleation on the basis of this representation. Then, we structure views in a set of hierarchies, where each ancestor view nucleates its descendants. Updates can be pruned from a hierarchy, or a part of it, when a certain view in the hierarchy is unaffected from a modification; in this case, all its ancestor views avoid the test, too. At the same time, nucleation hierarchies come with a price: they are instance dependent and thus they need to be rechecked after the modifications of the view extents take place.

Chapter Roadmap. In this Chapter we address the problem of efficiently maintaining top-k materialized views. In Section 3.1 we describe the state-of-the-art work. In Section 3.2 we propose a method for the fine-tuning of a materialized view for the 2 dimensional case. In Section 3.3 we generalize the problem for the n dimensional space and for non-linear scoring functions. In Section 3.4 we describe the problem of updating multiple views and insert the notion of nucleation relationships between views. In Section 3.5 we provide an algorithm that updates multiple views by constructing a hierarchy structure based on the nucleation relationships of the views. In Section 3.6 we report on the experimental assessment of the estimation of the

essential view size in order to sustain a high rate of updates. Finally, in Section 3.7 we summarize our findings.

3.1. Efficient Maintenance of Materialized top-k Views [YYY+03]

[YYY+03] deal with the following problem: Given a base table *R (id, val)* where *val* is the score of the tuple according to a scoring function and a materialized view *V (id, val)* containing the top-*k* tuples from *R* according to their values, compute a k_{max} that is adjusted at runtime such that a refill query, that re-computes the view *V* from scratch for the missing part, is rarely needed. Assume an update of the form <*id, val*> occurs and let $val_{k'}$ the tuple with the lowest value in *V*. Then the update can be classified as *ignorable*, *neutral*, *good* or *bad*. *Ignorable* is an update when its *id* is not in *V* and $val<val_{k'}$ and thus there is no effect in *V*. A *neutral* update occurs when its *id* is in *V* and $val>val_{k'}$. Then the tuple *id* is updated with value *val*. An update is categorized as *good* update when its id is not in *V* and $val>val_{k'}$. Then this tuple is inserted in *V* and *k'* is increased by one. If *k'* exceeds k_{max} then the lowest tuple in *V* is deleted. A *bad* update describes an update whose *id* is in *V* and $val<val_{k'}$. The tuple *id* is then deleted from *V* and *k'* is decreased by one. If *k'* drops below *k*, a *refill operation* is performed. A refill operation queries the base table *R* and returns all tuples ranked between *k* and k_{max}. [YYY+03] formulated the problem through a random walk model. The values of *k'* between two refill operations are represented through a 1- dimensional random walk model. The points are represented as $\{0, \ldots, n\}$ where *0* denotes the starting point (k_{max})and *n* ($k_{max} - k + 1$) the absorbing point at which a refill operation is needed. Assume that the random walk is currently in position *i* and a bad update moves the random walk to position *i+1* with probability p_i, whereas a good update moves the random walk to position *i-1* with probability q_i. In any other case the update is ignorable or neutral with probability $1 - p_i - q_i$. The problem is focused on analyzing the number of steps needed for the random walk model to go from *0* to *n*. In other words the analysis is conducted in order to find the probabilistic properties of the refill interval *Z*.

According to the assumptions that each step is independent of all previous choices and the probabilities of bad and good updates remain constant as updates occur in the

view ($p_0=p_1=\ldots=p_{n-1}=p$ and $q_0=q_1=\ldots=q_{n-1}=q$) the following occur. When $p=q$ then if $n = N^{\frac{1}{2}+\varepsilon}$ the refill integral Z is greater than N with high probability being $Pr[\, Z > N \,] \geq 1 - 4e^{-N^{2\varepsilon}/2}$, for any positive constant ε. When $p<q$, if $n = c \ln N$ the refill integral Z is greater than N with high probability being $Pr[\, Z > N \,] > 1 - o(1)$, for constant c big enough depending only on p and q. When $p>q$, then, if $n = N$ the refill integral Z is on the order of n. An adaptive algorithm chooses k_{max} at runtime without need to know the probabilities of good and bad updates. The algorithm is trying to keep the refill interval Z around the value $Z_0=C_{refill}/\, C_{update}$ (where C_{refill} is the observed cost of a refill query and C_{update} is the observed cost of a base table update). The algorithm counts the number of base table updates occurred from the last refill operation. If the updates are less than Z_0 /a then k_{max} is increased whereas if the number of updates is greater than aZ_0 then k_{max} is decreased, where a is a constant parameter.

3.2. Fine-Tuning of Views to Sustain High Update Rates

In this section we present our method for the fine tuning of materialized views defined over a relation that goes through updates in high rates. First, we formally define the problem. Second, we sketch our method and then, we move on to further detail the individual steps of the method.

3.2.1. Formal Definition of the Problem

The formal definition of the problem is:

Given *a base relation R (ID, X, Y) that originally contains N tuples, a materialized view V that contains top-k tuples of the form (id, val) where val is the score according to a function f(x, y)= a·x + b·y and a, b are constant parameters, the update ratios Λ_{ins}, Λ_{del} and Λ_{upd} for insertions, deletions and updates respectively over the base relation R,,*

Compute k_{comp} *that is of the form* $k_{comp} = k + \Delta k$

Such that the view will contain at least k tuples, $k \leq k_{comp}$*, with probability p, after a period T.*

Assume a base relation R (ID, X, Y), that contains N tuples a materialized view V that contains top-k tuples of the form (id, val) where val is the score according to a function $f(x, y)= a \cdot x + b \cdot y$ and a, b are constant parameters. Assume that the last tuple in the view has value val_k. Given the aforementioned update rates, insertions, deletions and updates occur in the base relation R with probabilities P_{INS}, P_{DEL} and P_{UPD} respectively. These probabilities are expressed as: $P_{INS} = \frac{\Lambda_{INS}}{\Lambda_{INS} + \Lambda_{DEL} + \Lambda_{UPD}}$.

$P_{DEL} = \frac{\Lambda_{DEL}}{\Lambda_{INS} + \Lambda_{DEL} + \Lambda_{UPD}}$ and $P_{UPD} = \frac{\Lambda_{UPD}}{\Lambda_{INS} + \Lambda_{DEL} + \Lambda_{UPD}}$

In the rest of our deliberations, updates are treated as combinations of deletions and insertions. This is a quite reasonable treatment, since we are mainly interested in the statistical properties of the rates of these actions and not in their hidden semantics. A simple method for the conversion of the involved rates is given in Section 3.2.2.

Our problem is to find a k_{comp} that will guarantee that the view will be maintained when insertions and deletions will occur in R. In order to do so, we must estimate the number of insertions and deletions that might affect the view. In other words, we need to compute the probability of the view being affected by a tuple inserted in R or deleted from R.

Assume that a new tuple z (id, x, y) is inserted in R. The probability of this tuple affecting the view is $p\ (z > val_k)$. Hence, the probability of a new tuple to be inserted in R and affect the view V is p_{ins}^{aff} which is expressed as: $p_{ins}^{aff} = p\ (z > val_k) \cdot p_{ins}$. The probability of a tuple to be deleted from R and affect the view V is p_{del}^{aff} which occurs as $p_{del}^{aff} = p\ (z > val_k) \cdot p_{del}$.

A problem that occurs with the maintenance of k_{comp} tuples at the view side is that k_{comp} incurs extra maintenance overheads, since the tuples of Δk can suffer updates too. Thus, we need to compute p_{ins}^{aff} and p_{del}^{aff} for the case where k_{comp} tuples are maintained. Therefore, the view V will contain k_{comp} tuples instead of k. Assume that

the last tuple of the view containing k_{comp} tuples is val_{kcomp}. Consequently, the probability of a new tuple z to affect the view V is $p(z>val_{\text{kcomp}})$ whereas the probability of a new tuple to be inserted in R and affect the view occurs as: $p_{ins}^{aff} = p(z>val_{\text{kcomp}}) \cdot p_{\text{ins}}$. Respectively the probability of a tuple z to be deleted from R and affect the view V can be expressed as: $p_{del}^{aff} = p(z>val_{\text{kcomp}}) \cdot p_{\text{del}}$.

3.2.2. Sketch of the Method

The proposed method is focused around three main steps: first, we compute the percentage of the incoming source updates that affect a top-k materialized view; second, we compute a safe value for the additional view tuples that we need in order to sustain high deletion rates; third, we fine tune this value with a safety range of values. Specifically, the three main steps are:

1. Given Λ_{INS}, Λ_{DEL} and Λ_{UPD}, we can compute λ_{ins} and λ_{del}, p_{ins} and p_{del}, and finally, p_{ins}^{aff} and p_{del}^{aff} as well as λ_{ins}^{aff} and λ_{del}^{aff}.

   Λ_{INS}, Λ_{DEL} and Λ_{UPD} denote the ratios of insertions deletions and updates that occur in the base table R. p_{ins} and p_{del} denote the probabilities of an insertion and deletion occurring on the base table R respectively. p_{ins}^{aff} and p_{del}^{aff} denote the probabilities of insertions and deletions that affect the view V respectively. These probabilities are expressed as a function of k_{comp}. λ_{ins}^{aff} and λ_{del}^{aff} denote the ratios of insertions and deletions occurring in the view V in the period of operations T. Updates are treated as a combination of deletions and insertions thus λ_{ins} and λ_{del} denote the ratios of insertions and deletions including those occurring from updates.
2. Compute k_{comp} as a function of λ_{ins}^{aff}, λ_{del}^{aff}.

   k_{comp} denotes the number of tuples that the view V should initially contain, such that after a period of operations T, V will contain at least k tuples.
3. Fine-tune k_{comp} by using the variance of the probability that a deletion and insertion action affects the materialized view.

3.2.3. Handling of Updates

Given Λ_{INS}, Λ_{DEL} and Λ_{UPD} and treating updates as a combination of deletions and insertions, the ratios λ_{ins} and λ_{del} can be computed through the following equations:

λ_{ins} = number of insertions including those from updates / T

λ_{del} = number of deletions including those from updates / T

Λ_{INS} = number of insertions / T

Λ_{DEL} = number of deletions /T

Λ_{UPD} = number of updates /T

Therefore, $\lambda_{ins}=\Lambda_{INS}+\Lambda_{UPD}$, $\lambda_{del}=\Lambda_{DEL}+\Lambda_{UPD}$. In addition, p_{ins} and p_{del} can be expressed through the usage of ratios as $p_{ins}=\frac{\lambda_{ins}}{\lambda_{ins}+\lambda_{del}}$ and $p_{del}=\frac{\lambda_{del}}{\lambda_{ins}+\lambda_{del}}$ respectively.

3.2.4. Computation of the Actual Rates that Affect V

The problem now is to compute the probabilities p_{ins}^{aff} and p_{del}^{aff} that affect the view V. These probabilities can be computed as $p_{ins}^{aff} = p_{ins} \cdot p\ (z > val_{kcomp})$ and $p_{del}^{aff} = p_{del} \cdot p(z > val_{kcomp})$ respectively. Actually, p_{ins}^{aff} is the number of insertions affecting the view V divided by the number of insertions and deletions occurring on the base table R and p_{del}^{aff} is the number of deletions affecting the view V divided by the number of insertions and deletions occurring on the base table R. Now the problem is focused upon finding the probability $p(z>val_k)$.

In order to compute the above probability we will use the *Empirical Cumulative Distribution Function* $F_n(x)$ (*ECDF*). Instead of using of a particular parametric cumulative distribution function, we will use ECDF which is a non parametric cumulative distribution function that adapts itself to the data. ECDF returns the values of a function $F(x)$ such that $F_n(x)$ represents the proportion of observations in a sample less than or equal to x. $F_n(x)$ assigns the probability $1/n$ to each of n

observations in the sample. In other words $F_n(x)$ estimates the true population proportion $F(x)$. ECDF is formally defined as follows [Triv02]:

Let X_1, X_2, …, X_n be independent, identically distributed random variables and let $x_1<x_2<\ldots<x_n$ denote the values of the order statistics of the sample. Then the empirical distribution function $F_n(x)$ is defined by the following formula:

$$F_n(x) = \begin{cases} 0, & x<x_1 \\ \dfrac{i}{n}, & x_i \le x<x_{i+1} \\ 1, & x_n \le x. \end{cases}$$

The alternative definition of $F_n(x)$ is:

$$F_n(x) = \frac{number\_of\_values\_in\_the\_sample\_that\_are\_\le x}{n}$$

Assume that the base relation R contains N tuples and the view V should contain k_{comp} tuples. If we order these tuples according to their values then there are N-k_{comp} tuples in R with value less than the value of k_{comp}. The following theorem implies that when the sample size n is large, $F_n(x)$ is quite likely to be close to $F(x)$ over the entire real line.

Theorem 3.1 Glivenko-Cantelli Theorem [DeSc02]:

Let $F(x)$ denote the density function of the distribution from which the random sample X_1, X_2,…, X_n was drawn. For each given number x ($-\infty<x<\infty$) the probability that any particular observation X_i will be less than or equal to x is $F(x)$. Therefore, it follows from the law of large numbers that as $n \to \infty$, the proportion $F_n(x)$ of observations in the sample that are less than or equal to x will converge to $F(x)$ uniformly over all values of x. Let $D_n = \sup_{-\infty<x<\infty} |F_n(x) - F(x)|$, the Glivenko-Cantelli theorem states that $D_n \xrightarrow{p} 0$. □

Therefore, the probability of a tuple z affecting the view V can be expressed as:

$$p(z>val_{kcomp}) = 1 - p(z \leq val_{kcomp}) = 1 - F_N(k_{comp})$$

$$p(z>val_{kcomp}) = 1 - \frac{N - k_{comp}}{N} = \frac{k_{comp}}{N} \qquad \text{Eq 3.1}$$

As a more general example, consider a base relation *R* where the score of its tuples according to a function follow an exponential distribution in the interval [0, 2] and that a view *V* requires the top-*k* tuples of *R* according to their score value. In Figure 3.1 the probability distribution function of an exponential distribution is illustrated. In addition, assume that the top-*k* tuples are the 20% of the relation *R* and thus the vertical line top-k shown in Figure 3.1 denotes the values of the tuples that participate in the top-*k* view. Thus, the values in the view are greater or equal to 0.3. Assume a new tuple *t* following the same exponential distribution being inserted in *R*. For the new tuple *t* the probability of its value participating in the top-*k* ones is again 20%.

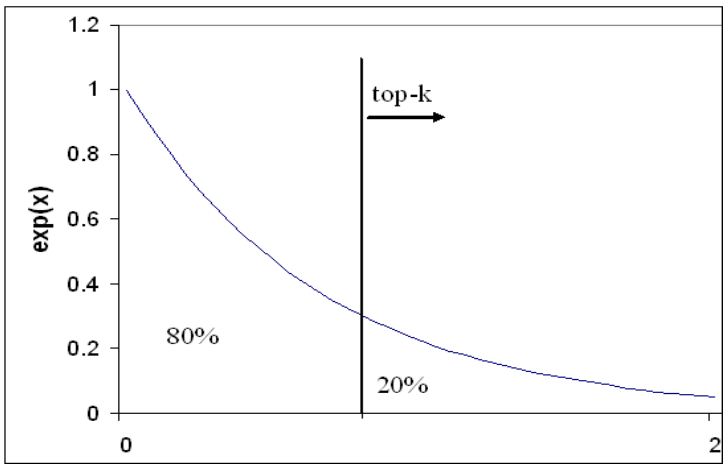


Figure 3.1. Exponential Probability Distribution.

Again, consider a similar situation where a view contains the top-*k* tuples from a base relation *R* according to a scoring function. Assume that the score values of *R* this time follow a beta distribution in the interval [0, 1] with parameters given as 5 and 2. Figure 3.2 illustrates the probability distribution function of such a distribution. Similar to the previous example, the vertical line illustrated as top-*k* in Figure 3.2 denotes that the view contains 20% of *R*'s tuples where the values participating in the view are greater or equal to 1.7. Assume a new tuple denoted as *t* being inserted in *R*.

The new tuple t will again follow the same beta distribution and the probability of t having a value greater than 0.8 is 20%.

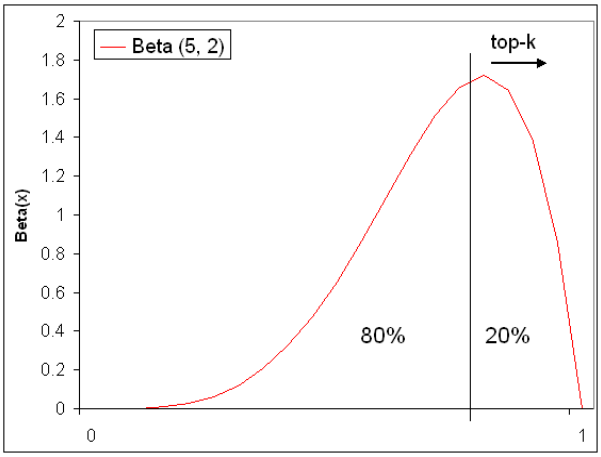


Figure 3.2. Beta Probability Distribution.

Therefore, λ_{ins}^{aff} and λ_{del}^{aff} are computed through the following equations:

$\lambda_{ins}^{aff} = p_{ins}^{aff} \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}})$ and $\lambda_{del}^{aff} = p_{del}^{aff} \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}})$.

According to equation 3.1, λ_{ins}^{aff} and λ_{del}^{aff} can be expressed as:

$$\lambda_{ins}^{aff} = p_{\text{ins}} \cdot p(z > val_{\text{kcomp}}) \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}})$$

$$\lambda_{ins}^{aff} = p_{\text{ins}} \cdot \frac{k_{comp}}{N} \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}}) \qquad \text{Eq 3.2}$$

and

$$\lambda_{del}^{aff} = p_{\text{del}} \cdot p(z > val_{\text{kcomp}}) \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}})$$

$$\lambda_{del}^{aff} = p_{\text{del}} \cdot \frac{k_{comp}}{N} \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}}) \qquad \text{Eq 3.3}$$

3.2.5. Computation of k_{comp}

The last step of the method is to compute k_{comp}, such that it will guarantee that the view will contain at least k tuples, $k \leq k_{\text{comp}}$, with probability p, after a period of operation T. In other words compute a k_{comp} that is of the form $k_{\text{comp}} = k + \Delta k$. In general, when the ratio of insertions λ_{ins} is greater than that of deletions λ_{del} it is clear that V will be maintained. The problem arises when the opposite occurs, i.e., when the

ratio of deletions is greater than that of insertions. In such a case it is vital to compute a value for k_{comp} that can guarantee that V will contain at least k tuples after a period of operations.

Let us denote the frequency of deletions that affect the view V as λ_{del}^{aff}. In a period of time T, in order to keep the view maintained the following inequality should hold: $k_{comp}^{T} - \lambda_{del}^{aff} \cdot \mathrm{T} \geq k$.

Thus, in case both insertions and deletions occur in a period of time T, in order to keep the view maintained for k_{comp} the following inequality should hold $k_{\text{comp}} \geq k + (\lambda_{del}^{aff} - \lambda_{ins}^{aff}) \cdot T$. Clearly, to minimize memory consumption, we need to take the minimum possible k_{comp} and thus treat the above inequality as the equation $k_{\text{comp}} = k + (\lambda_{del}^{aff} - \lambda_{ins}^{aff}) \cdot T$.

Therefore, by replacing λ_{ins}^{aff} and λ_{del}^{aff} from equations 3.2 and 3.3 the following equality occurs:

$$k_{\text{comp}} = k + (p_{\text{del}} - p_{\text{ins}}) \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}}) \cdot \frac{k_{comp}}{N} \cdot T \Rightarrow$$

$$k_{\text{comp}} = k + (\lambda_{\text{del}} - \lambda_{\text{ins}}) \cdot \frac{k_{comp}}{N} \cdot T \qquad \text{Eq 3.4}$$

Thus, by solving the above equation according to k_{comp} we obtain:

$$k_{\text{comp}} = k \cdot \frac{N}{N + (\lambda_{ins} - \lambda_{del}) * T} \qquad \text{Eq 3.5}$$

Equation 3.5 has a meaning when $N + (\lambda_{\text{ins}} - \lambda_{\text{del}}) \cdot T > 0$. This states that the size of the base relation R will not fall below 0, after updates occur in a period of operations T. At the same time, when $\lambda_{\text{ins}} - \lambda_{\text{del}} < 0$ (i.e., the case we are particularly interested in), then the fraction is greater than 1 and thus, $k_{\text{comp}} > k$.

3.2.6. Fine-Tuning of k_{comp}

Although we now have a formula to compute the value of k_{comp}, we have expressed the probability of a new tuple $z(id, x, y)$ affecting the top-k_{comp} tuples of the view as $p(z>val_{\text{kcomp}})$. Assume that a new tuple z is inserted in R. The probability of this tuple to affect the view is $p(z>val_{\text{kcomp}})$ whereas, the probability of this tuple not to affect the view is 1- $p(z>val_{\text{kcomp}})$. The above can be expressed as a Bernoulli experiment with two possible events. These are (a) the new tuple being inserted in V with probability of success $p(z>val_{\text{kcomp}})$ and, (b) the exact opposite where the new tuple is not inserted in V with probability 1- $p(z>val_{\text{kcomp}})$. When the ratio of insertions occurring in the base relation R are λ_{ins}, a Bernoulli experiment is occurring λ_{ins} times where the probability of success is $p(z>val_{\text{kcomp}})$ and the number of successes follow a Binomial distribution. The probability of having Y_{ins} affected insertions in the view follow a Binomial distribution of the form *Binomial* (λ_{ins}, $p(z>val_{\text{kcomp}})$). The variance of the above distribution can be expressed as:

$Var(Y_{\text{ins}}) = \lambda_{\text{ins}} \cdot p(z>val_{\text{kcomp}}) \cdot (1- p(z>val_{\text{kcomp}}))$.

The above formula indicates that insertions expected to affect the view may vary by $Var(Y_{\text{ins}})$. Correspondingly, if there are λ_{del} deletions occurring in the base relation R, then the variance of these deletions expected to affect the view is

$Var(Y_{\text{del}}) = \lambda_{\text{del}} \cdot p(z>val_{\text{kcomp}}) \cdot (1- p(z>val_{\text{kcomp}}))$. This occurs as the variance of the Binomial distribution $B(\lambda_{\text{del}}, p(z>val_{\text{kcomp}}))$, which is similar to the one used for insertions.

Therefore in the worst case, in order to guarantee that the view will contain at least k tuples with confidence 95%, where $k \leq k_{\text{comp}}$, equation 3.4 becomes as stated below:

$$k_{\text{comp}} = k + (\lambda_{\text{del}} - \lambda_{\text{ins}}) \cdot \frac{k_{comp}}{N} \cdot T + 2 \cdot Var(Y_{\text{del}}) + 2\, Var(Y_{\text{ins}}) \qquad \text{Eq 3.6}$$

The confidence rate of 95% occurs from statistical properties concerning the variance factor appearing in equation 3.6. In case another confidence percentage is needed, equation 3.6 can be adjusted according to typical statistical methods [DeSc02].

3.2.7. Discussion

The problem of maintaining a view when updates occur in a base relation R, mainly lies in the problem of estimating the number of updates that will affect the view. Statisticians have contributed in this by providing equations that compute the value of a probability of the form $p(z>val_{kcomp})$. However, the equation of such a probability depends on the distribution that the variable z follows. In our context, the variable z is a linear combination of the form $a \cdot x + b \cdot y$ where x and y are values from the attributes X and Y of the base relation. Even if the distributions that X and Y follow are known, the distribution of the score Z cannot be computed unless X and Y follow a *stable* distribution. A stable distribution (e.g., Normal, Cauchy) has the property of stability. This property states that if a number of independent identically distributed (iid) random variables have a stable distribution, then a linear combination of these variables will have the same distribution. Therefore, the distribution of the variable Z can only be known in few cases. However, even if the distribution of the score was known, the probability $p(z>val_{kcomp})$ could be computed only with respect to the val_k instead of the value val_{kcomp}. This is because the val_{kcomp} could not been know in advance. Therefore, an iterative procedure would be needed. This occurs from the fact that we could compute the effect top-k tuples could have but not the effect the extra tuples would arise. Thus, a recursive procedure would be required.

3.2.8. Example

As an example, consider the base relation R (ID, X, Y) initially containing N tuples with N=20 where attributes X and Y follow a uniform distribution over the interval [0, 100]. In addition, consider a materialized view V that contains the top-3 tuples (k=3) of the form (id, val) where $val=3 \cdot x+7 \cdot y$ is the score according to a function $f(x, y)=a \cdot x + b \cdot y$ and a=3, b=7. The base relation R and the initial state of V are shown in Figure 3.3. Finally, the update ratios are Λ_{ins}=5, Λ_{del}=15 and Λ_{upd}=0. We will compute k_{comp} such that the view would contain k_{comp} tuples instead of k in order to be kept maintained when insertions, deletions and updates will occur in the base relation R. Moreover, let the period of operations occurring set as T=1.

According to the method of Section 3.2.3, the ratios λ_{ins} and λ_{del} are 5 and 15 respectively. Therefore, p_{ins}=0.25 and p_{del}=0.75. The probability $p(z \geq val_{kcomp})$ can be calculated according to the following:

$p(z \leq val_{kcomp}) = F_N\ (val_{kcomp})$

$p(z \leq val_{kcomp}) = (number\ of\ elements\ in\ score\ sample \leq val_{kcomp})\ /\ N$

$p(z > val_{kcomp}) = k_{comp}\ /\ 20$

In consequence, the probabilities p_{ins}^{aff} and p_{del}^{aff} can be calculated as:

$$p_{ins}^{aff} = p_{ins} \cdot p(z \geq val_{kcomp}) = 0.25 \cdot \frac{k_{comp}}{20} \text{ and } p_{del}^{aff} = p_{del} \cdot p(z \geq val_{kcomp}) = 0.75 \cdot \frac{k_{comp}}{20}.$$

R		
id	**X**	**Y**
1	56	41
2	58	62
3	15	97
4	78	86
5	69	10
6	96	60
7	12	43
8	74	76
9	26	71
10	95	92
11	34	51
12	27	36
13	19	25
14	68	81
15	91	82
16	84	65
17	41	59
18	37	37
19	23	17
20	47	27

V	
id	**Z**
10	929
15	847
4	836

Figure 3.3. Base Relation *R*.

Given the previous probabilities, the effective update ratios for the view *V* are then:

$$\lambda_{ins}^{aff} = p_{ins}^{aff} \cdot (\lambda_{ins} + \lambda_{del}) = 0.25 \cdot \frac{k_{comp}}{20} \cdot (5 + 15)$$

$$\lambda_{del}^{aff} = p_{del}^{aff} \cdot (\lambda_{\text{ins}} + \lambda_{\text{del}}) = 0.75 \cdot \frac{k_{comp}}{20} \cdot (5 + 15)$$

The above equations state that if 5 insertions will occur in the base relation R, λ_{ins}^{aff} will affect the view and if 15 deletions occur then λ_{del}^{aff} will affect the view respectively. To be more specific the ceiling function is applied on λ_{ins}^{aff} and λ_{del}^{aff}. Therefore, for k_{comp} the following inequality holds:

$k_{\text{comp}} \geq k + \lceil \lambda_{del}^{aff} - \lambda_{ins}^{aff} \rceil \cdot T \Rightarrow k_{\text{comp}} \geq 6$

where actually $k_{\text{comp}} = 6$. Thus, k_{comp} should be 6 in order to keep the view maintained after insertions, deletions and updates will occur in the base relation R. Suppose that insertions and deletions, shown in Figure 3.4, occur in the base relation R. The view V contains initially top-6 tuples and after updates the view will contain top-3 tuples. These are shown in Figure 3.5 where the dark shading denotes the initial top-3 tuples of V whereas the light shading denotes the extra top-3 tuples in order to have top-k_{comp} tuples.

insertions			**deletions**		
Id	**X**	**Y**	**id**	**X**	**Y**
21	25	33	1	56	41
22	18	64	2	58	62
23	97	83	3	15	97
24	31	50	4	78	86
25	53	82	5	69	10
			7	12	43
			8	74	76
			10	95	92
			11	34	51
			12	27	36
			13	19	25
			15	91	82
			16	84	65
			17	41	59
			20	47	27

Figure 3.4. Insertions and Deletions Occurring in Base Relation R.

V		
id	**Z**	
10	929	Deleted
23	872	Inserted
15	847	Deleted
4	836	Deleted
14	771	
8	754	Deleted
25	733	Inserted
3	724	Deleted

V	
id	**Z**
23	872
14	771
25	733

Figure 3.5. The View *V* Prior and Subsequent to Updates.

3.3. Generalization of the Problem

In this section we provide two generalization of the above problem. The first generalization concerns a relation *R* that contains more than two attributes and the scoring function is of linear form whereas the second generalization concerns the problem when the scoring function is not obligatory linear but is a monotone function. Assume that the relation is of the form R (ID, X_1, X_2, …, X_n) and the scoring function of the view includes all the attributes X_i or a number of them. The problem then can be generalized as:

3.3.1. Formal Definition of the Problem Generalized for More than Two Attributes

Given *a base relation R (ID, X_1, X_2, …, X_n) that originally contains N tuples, a materialized view V that contains top-k tuples of the form (id, val) where val is the score according to a function $f(x_1, x_2, \ldots, x_n)=a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n$ and $a_1, a_2, \ldots a_n$ are constant parameters, the update ratios Λ_{ins}, Λ_{del} and Λ_{upd} for insertions, deletions and updates respectively over the base relation R,*

Compute k_{comp} *that is of the form* $k_{comp} = k + \Delta k$

Such that the view will contain at least k tuples, $k \leq k_{comp}$*, with probability p, after a period T.*

The solution to the problem is similar to the previous three-step method which leads to the computation of equation 3.5. This is because the computation of k_{comp} from

equation 3.5 is independent of the attributes that participate in the scoring function of *V*.

3.3.2. Formal Definition of the Problem Generalized for Non-Linear Monotonic Functions

Given *a base relation R (ID, X_1, X_2, …, X_n) that originally contains N tuples, a materialized view V that contains top-k tuples of the form (id, val) where val is the score according to a monotone function f(x_1, x_2, …,x_n), the update ratios Λ_{ins}, Λ_{del} and Λ_{upd} for insertions, deletions and updates respectively over the base relation R,*

Compute k_{comp} *that is of the form* $k_{comp} = k + \Delta k$

Such that the view will contain at least k tuples, $k \leq k_{comp}$*, with probability p, after a period T.*

In general, the scoring function of the view can be any monotonic function and not compulsory a linear function. The monotonic property is important in order to make use of the ECDF distribution function. Remember that ECDF returns the values of a function $F(x)$ such that $F_n(x)$ represents the proportion of observations in a sample less than or equal to x. Therefore, it is necessary that the values among a sample have an order. In other words, for the setting of our problem, the values of the sample are the tuples and their score according to the scoring function of *V*.

3.4. Multiple View Updates

So far, our deliberations have been focused on the fine tuning of the size of a materialized view in order to sustain high update rates. The next step in our investigation of the field of top-*k* materialized view refreshment is to consider the case where more than one views need to be materialized. We will split the overall problem in two parts:

The first problem that we consider concerns the *dominance* of a view over another and how this reflects to the view refreshment problem. In other words, we investigate

whether we can efficiently infer when the updates over a view directly affect the materialized contents of another view. Formally, assume a relation $R(ID, X, Y,\ldots)$ and two materialized views $V_1(ID, X, Y, s_1)$ and $V_2(ID, X, Y, s_2)$ that contain k_1 and k_2 tuples respectively. The score s_1 of V_1 is defined as $s_1 = a_1 \cdot x + b_1 \cdot y$ and the score s_2 of V_2 is defined as $s_2 = a_2 \cdot x + b_2 \cdot y$ and a_1, a_2, b_1, b_2 are positive parameters. Assume that updates occur at the relation R, and one of the views is affected by them (i.e., its extent has to be updated). Then, the question that arises is whether it is possible to know a-priori if the impact of these updates deterministically results in the necessity to update the other view too. We provide guarantees for this case via a geometrical representation of the views and their scoring equations and we can safely determine the effect of an update on a view on the basis of its effect on another view.

The second problem that we consider involves the design of an efficient structure for a large set of top-k materialized views in order to speed up their maintenance. The constructed structure is based on the abovementioned dominance relationship among the views. We introduce hierarchies for the views and test batches of updates over the bottom of the hierarchies. If the updates affect the bottom view, its immediate ancestors are candidates for being affected by the updates; otherwise, we can surely alleviate them from the burden of being tested against the update under examination. Obviously, the same pattern recursively propagates throughout all the hierarchy as long as a member of the hierarchy is affected.

The structure of this section is as follows. First, we start with preliminary ideas coming from the related literature and subsequently, we expand these results to discuss the case of view dominance. The third part of the section involves the discussion of view maintenance for large sets of views.

3.4.1. View Nucleation

Assume a relation $R(ID, X, Y,\ldots)$ and a materialized view $V(ID, X, Y, s_1)$ that contains k tuples, scored via s which is defined as $s = w_x \cdot x + w_y \cdot y = w \cdot (a \cdot x + y)$. Both a_1, and b_1 are positive numbers). To simplify notation, we will often denote the view as $V(a, k)$. Assume now a relation $R(ID, X, Y,\ldots)$ and two materialized views $V_1(ID, X, Y, s_1)$ and

$V_2(ID, X, Y, s_2)$ that contain k_1 and k_2 tuples respectively, with the score s_1 of V_1 defined as $s_1= a_1 \cdot x + b_1 \cdot y$ and the score s_2 of V_2 defined as $s_2=a_2 \cdot x + b_2 \cdot y$. All a_1, a_2, b_1, b_2 are positive numbers. Assume now that updates occur to the base relation and they must be propagated to the views. In a typical relational situation with SPJ queries, we would say that a view V_1 is contained within view V_2, if the extent (i.e., the materialized tuples) of view V_1 is always a subset of the extent of view V_2. In our case, due to the fact that the scores of the materialized tuples are different, we slightly tweak the terminology and instead of the 'containment' terms we employ a terminology around the notion of '*nucleus*'.

Definition 3.1 (Nucleation Relationship of two Views). Assume a relation $R(ID, X, Y,\ldots)$ and two materialized views $V_1(a_1, k_1)$ and $V_2(a_2, k_2)$. A view V_2 *nucleates* a view V_1 if for each tuple $t(t.id,t.x,t.y,\ldots) \in R$ that belongs to the extent of V_2 as a tuple $t_2(t.id,t.x,t.y,s_2(t)) \in V_2$ (i.e., with a score $s_2(t)$), a respective tuple $t_1(t.id,t.x,t.y,s_1(t))$ obligatorily belongs to the extent of V_1. We denote this nucleation as $V_2 \subseteq V_1$.

Definition 3.2 (Nucleus equivalent Views). Two views $V_1(a_1, k_1)$ and $V_2(a_2, k_2)$ are *nucleus equivalent* if both V_2 nucleates V_1 and V_1 nucleates V_2.

Clearly, the main idea behind nucleation is that despite the difference in scores, the 'nucleus' of a tuple (i.e., the tuple identifier and the scoring attributes) are the same in the respective materialized tuples.

3.4.2. Updates for Nucleated Views

Can we efficiently decide when a view V_1 is nucleated by another view V_2? In this subsection, we will deal with this problem based on an analysis conducted via a geometric representation. Specifically, assume the views V_1 and V_2 defined as $V_1(ID, X, Y, s_1)$ and $V_2(ID, X, Y, s_2)$ that contain k_1 and k_2 tuples respectively, with the score s_1 of V_1 defined as $s_1= a_1 \cdot x + b_1 \cdot y$ and the score s_2 of V_2 defined as $s_2=a_2 \cdot x + b_2 \cdot y$. These two views are characterized by the lines $y=b_1 \cdot a_1^{-1} \cdot x$ and $y= b_2 \cdot a_2^{-1} \cdot x$ respectively. There are two cases depending on the scoring functions of V_1 and V_2 and, consequently, on the slopes of their characteristic lines. The first case is trivial in

the sense that the two views are practically characterized by the same line. The second case concerns the typical situation when the lines of the two views are different. In the sequel, we discuss these cases in more detail.

Case 1: $\frac{a_1}{b_1} = \frac{a_2}{b_2}$

In this situation, the equation of V_1 is proportional to the equation of V_2. Without loss of generality assume that the equation of V_1 is $s_1 = a_1 \cdot x + b_1 \cdot y$ and the equation of V_2 is $s_2 = \lambda\ (a_1 \cdot x + b_1 \cdot y)$ where $\lambda \in \Re^+$. Then, the line that characterizes both views is $y = b_1 \cdot a_1^{-1} \cdot x$. There are two sub-cases in this situation.

Case 1.1: $k_1 = k_2$. In addition, assume that both views contain the same number of tuples, i.e., $k_1 = k_2$. In this case, any update affecting V_1 will definitely affect V_2 and vice versa. The only difference between the results of the two views will be the score of their tuples. Obviously, if V_1 contains a tuple t with score $s_1(t)$ then the same tuple will belong in V_2 but with score $s_2(t) = \lambda \cdot s_1(t)$.

Lemma 3.1. If the equation of a view V_1 is proportional to the equation of a view V_2 with the same extent size k of materialized tuples, then they both contain the exact same tuples (i.e., they are nucleus equivalent) with the same ordering.

Proof. Assume that the equation of V_1 is $s_1 = a_1 \cdot x + b_1 \cdot y$ and the equation of V_2 is $s_2 = \lambda\ (a_1 \cdot x + b_1 \cdot y)$ where $\lambda \in \Re^+$. In addition, assume $t_k(x_k, y_k)$ is the last tuple in V_1. Then for any tuple $t(x_t, y_t)$ from V_1, obviously by definition $s_1(t) \geq s_1(t_k)$. In other words, $a_1 \cdot x_t + b_1 \cdot y_t \geq a_1 \cdot x_{tk} + b_1 \cdot y_{tk}$. Multiplying this inequality with the proportion λ, we get $\lambda\ (a_1 \cdot x_t + b_1 \cdot y_t) \geq \lambda\ (a_1 \cdot x_{tk} + b_1 \cdot y_{tk})$. This states that $s_2(t) \geq s_2(t_k)$ for every tuple t from V_1. However, the last inequality is the definition of the top-k tuples of V_2. Therefore, any tuple in V_1 will be in V_2 as well. In addition, if for two tuples t_1 and t_2 from V_1 we know that $s_1(t_1) \geq s_1(t_2)$ then by multiplying the inequality with the parameter λ we get $s_2(t_1) \geq s_2(t_2)$. This proves that tuples t_1 and t_2 appear with the same ordering in V_2 as well. □

Corollary 3.1. If the equation of a view V_1 is proportional to the equation of a view V_2 with the same extent size k of materialized tuples, whenever V_1 is affected by an update, V_2 will be affected as well and vice versa.

Proof. Assume a tuple $t(x_t, y_t)$ being updated (inserted or deleted) in R and t affects V_1 with score $s_1(t)$. This means that $s_1(t) = a_1 \cdot x_t + b_1 \cdot y_t$ and $s_1(t) \geq s_1(t_k)$, where t_k is the last tuple materialized in V_1. Multiplying the above inequality by the parameter λ we get $\lambda \cdot s_1(t) \geq \lambda \cdot s_1(t_k)$ which can be written as $s_2(t) \geq s_2(t_k)$. From the above lemma t_k is also the last materialized tuple in V_2. Therefore, tuple t has a higher score than t_k for V_2 as well. Therefore, tuple will also affect V_2. □

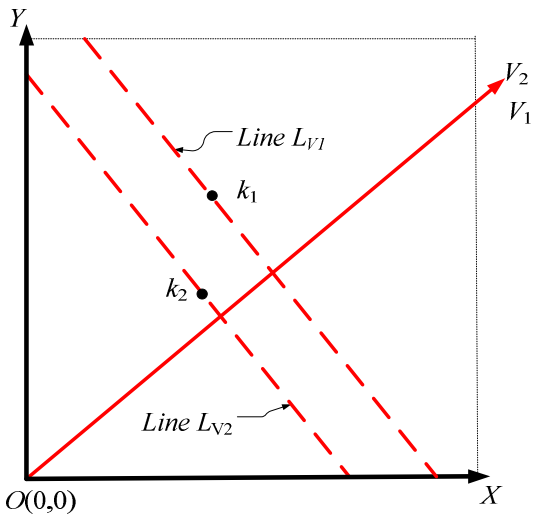


Figure 3.6. Both Views Are of Proportional Equations.

Case 1.2: $k_1 < k_2$. Consider now the case where the equations of the two views V_1 and V_2 are still proportional, but $k_1 < k_2$ (which means that V_1 contains less tuples than V_2). In this case, V_1 nucleates V_2 and any update affecting V_1 will definitely affect V_2 as well.

Corollary 3.2. If the equation of a view $V_1(a, k_1)$ is proportional to the equation of a view $V_2(a, k_2)$ and $k_1 < k_2$, V_1 nucleates V_2.

Proof. According to the above lemma as shown in Figure 3.6, the top-k_1 tuples are exactly the same for both views. The inverse however, does not always hold. This is because an update occurring in V_2 might be affecting the tuples that are ranked below k_1 and thus, the k_1 tuples of V_1 will not suffer any change. Obviously, if an update occurring in V_2 affects the top-k_1 tuples then it will affect V_1 as well. □

Case 2: $\frac{a_1}{b_1} \neq \frac{a_2}{b_2}$

In this situation, the equations of the two views are completely different. In this case, since the equations of the two views are not proportional, the only piece of information that can be used in order to conduct a conclusion with respect to the nucleation of the two views is the position of the last tuple of each view. Again, assume two views $V_1(ID, X, Y, s_1)$ and $V_2(ID, X, Y, s_2)$ with k_1 and k_2 tuples respectively where score s_1 is defined as $s_1= a_1 \cdot x + b_1 \cdot y$ and s_2 is defined as $s_2=a_2 \cdot x + b_2 \cdot y$. The lines that characterize the two views are V_1: $y=b_1 \cdot a_1^{-1} \cdot x$ and V_2: $y= b_2 \cdot a_2^{-1} \cdot x$ respectively (see Figure 3.7 or Figure 3.8). Let t_{k1} be the last tuple materialized in V_1 with score $s_1(t_{k1})$ and L_1 be the line which is vertical to the line of V_1 and passes from point t_{k1}. The area above the line L_1 contains the top-k_1 tuples with respect to V_1. Now, take the line L_2, which is vertical to V_2 and passes through the point t_{k2}, where t_{k2} is the last tuple materialized in V_2. The area above line L_2 contains points that belong to V_2. In addition, let I denote the point where L_1 and L_2 intersect.

The position of the intersection point I is critical in regards to the knowledge of whether updates affecting one view will affect the other view or not. Assume that the active domains of attributes X and Y are $X \in [x_{min}, x_{max}]$ and $Y \in [y_{min}, y_{max}]$. We will employ the term *active area* to refer to the region in which any tuple from relation R belongs. This is constrained within a rectangle defined by the points (x_{min}, y_{min}) and (x_{max}, y_{max}). Checking whether point I lies inside the active area or not can be easily done when the last tuple of each view is known. Line L_1 is expressed as: $a_1 \cdot x + b_1 \cdot y = s_1(t_{k1})$. Similarly, line L_2 is expressed as: $a_2 \cdot x + b_2 \cdot y = s_2(t_{k2})$. Therefore, the coordinates of point $I(x_I, y_I)$ can be found by solving the linear system of L_1 and L_2. Specifically,

$x_I = (a_1 \cdot b_2 - a_2 \cdot b_1)^{-1} \cdot (b_2 \cdot s_1(t_{k1}) - b_1 \cdot s_2(t_{k2}))$ and

$y_I = (a_1 \cdot b_2 - a_2 \cdot b_1)^{-1} \cdot (a_1 \cdot s_2(t_{k2}) - a_2 \cdot s_1(t_{k1}))$.

Depending on the position of where point *I* lies we have the following cases:

Case 2.1: point *I* intersects outside of the active area. Point *I* lies outside of the active area if at least one of its coordinates x_I, y_I does not belong in the active domains of *X* and *Y* respectively. In fact, in case point *I* lies outside the active area (see Figure 3.7), then all tuples materialized in one view are also materialized in the other view as well. This situation indicates that whenever an update occurs in V_2, this will definitely affect V_1 as well. The inverse however is not always true.

In Figure 3.7, tuples of V_2 also belong in V_1 and V_2 nucleates V_1. In other words, V_2 is a subset of V_1 in the sense that any tuple in V_2 will be part of V_1 but with a different ranking and score.

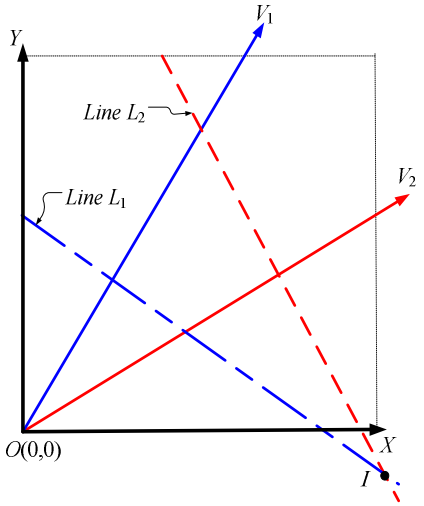


Figure 3.7. Intersection of Two Views Outside the Active Area.

Case 2.2: **Point *I* intersects inside the active area**. Point *I* lies inside the active area if both of its coordinates x_I, y_I belong in the active domains of *X* and *Y* respectively. In case point *I* lies within the active area, there is no clear guarantee of the way the views are affected when updates occur. However, there is a sub-area which we refer to as *safe area*, where both views will be affected in the same way. Observe Figure 3.8, where the safe area is the convex defined by the points y_2, *I*, x_1, *R*. This area contains

points that both belong in V_1 and V_2. If an update occurs within this safe area then if one view is affected then obviously the other view will be affected.

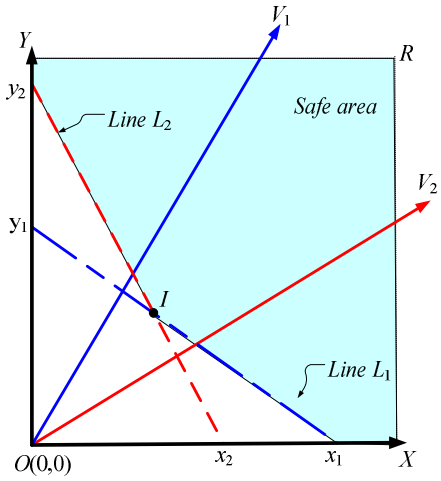


Figure 3.8. Intersection of Two Views Inside the Active Area.

On the other hand, there are two critical areas where an update might occur and affect one view but not the other. These two critical areas are the two triangles tr_1: y_1y_2I and tr_2: x_1x_2I. Assume the relation R is updated with a tuple t that falls within the triangle tr_1. This means that either t is inserted in R and its representation lies within tr_1, or t belonging in tr_1 is deleted from R. Then, t will affect V_1, but will leave V_2 unaffected. Similarly, if tuple t falls within the triangle tr_2, then V_2 will suffer changes whereas V_1 will remain unchanged.

Case 2.3: **Special Case.** Assume two views $V_1(a_1, k_1)$ and $V_2(a_2, k_2)$ as the ones depicted in Figure 3.8, where point I is within the active area. The safe area of these two views is the convex defined by the points: y_2, I, x_1, R. The main observation that can be made is that *the tuples in the safe area are common* and therefore, the two views share the same set of top-k tuples, $k \leq k_1,k_2$ (although, possibly with different ordering for each view, since each point in the safe area has a different score for each of the two views). The areas outside the safe area contain k_1-k and k_2-k tuples for each view, respectively.

In addition, assume now that both (i) k_1=k_2 and (ii) the two critical regions tr_1: y_1y_2I and tr_2: x_1x_2I are void of tuples. In such a case when an update occurs, a conclusion can be conducted depending on the type of the update (i.e., insertion or deletion):
If the update is a deletion and affects one of the views, then it will definitely affect the other view.

However, if an insertion occurs and affects one of the views, then depending on the position of the insertion the other view might be or not affected. This depends on whether the insertion lies within the safe area or in one of the non-common triangles.

3.4.3. Discussion & Summary

It is important to stress that the nucleation relationship of the two views is typically dependent on the specific instances (expect for special cases) and has to be re-evaluated each time that updates occur.

Whenever an update occurs that affects at least one of the views, the position of its respective line (L_1 and/or L_2) is altered. In fact, when an insertion occurs in at least one of the views, the position of its respective line is moved towards the upper right part of the active area (or, infinity, if one chooses to think without active areas). Similarly, when a deletion occurs in a view, its respective line is moved towards the beginning of the axes. Therefore, lines L_1 and/or L_2 should be recomputed after every update affects at least one of the views. Consequently, point I should be recomputed. This might also cause the change from the situation where I is outside the active area to the situation where I is inside the active area and vice versa.

Combining the above cases the following theorem occurs (the proof is obvious by referring to the lemmas and discussions of this section).

Theorem3.2. Assume two views V_1 (ID, X, Y, s_1) and V_2 (ID, X, Y, s_2) that contain k_1 and k_2 tuples and have their scores defined as $s_1 = a_1 \cdot x + b_1 \cdot y$ and $s_2 = a_2 \cdot x + b_2 \cdot y$, respectively. In addition, without loss of generality, assume for the slopes of the lines

L_1 and L_2 that $\frac{a_1}{b_1} \leq \frac{a_2}{b_2}$. When updates occur in the relation R and the view V_1 is affected, then, the view V_2 will be affected if one of the following holds:

The scoring function of V_1 is proportional to the scoring function of V_2 and $k_1 \leq k_2$

The intersection point I of L_1 and L_2 lies outside the active area, and L_2 is above L_1

The intersection point I lies inside the active area, critical areas tr_1 and tr_2 are void of tuples and updates are only deletions.

The intersection point I lies inside the active area, critical areas tr_1 and tr_2 are void of tuples and insertions occur only within the safe area. □

3.5. Updating Multiple Nucleated Views

Assume a relation $R(ID, X, Y, \ldots)$ containing initially n tuples. In addition, assume that our user requirements allow us to structure the updates that occur in R in a batch way, with ΔR^+, ΔR^- denoting the insertions and deletions of a batch respectively. Assume a set of m materialized views $\mathbb{V} = \{V_i(ID, X, Y, s_i) \mid 1 \leq i \leq m\}$ where each view V_i contains k_i tuples with score s_i defined as $s_i = a_i \cdot x + b_i \cdot y$. When updates occur in R, the set of views $\mathbb{V}$ should be maintained appropriately. In a naïve manner, ΔR^+ and ΔR^- would be checked over each view of the set $\mathbb{V}$. However, if there are *nucleation* relationships among them, the update process can be done more efficiently. In this section we describe an algorithm that updates a set of views by taking advantage of the *nucleation* relationships among them.

3.5.1. Representation of Nucleation Relationships as Hierarchy Paths

Assume that there exist several *nucleation* relationships among the set of views $\mathbb{V}$. Taking into consideration the nucleation between views, we can construct a number of hierarchy paths among them. Each hierarchy path will contain the views that are related-connected by nucleation relationships. As a simple example, assume that V_1 nucleates V_2 and V_2 nucleates V_3. This can be depicted as a hierarchy shown in Figure 3.9 where the nucleation relationship is represented as an ancestor-descendant relationship (i.e., the fact that V_1 nucleates V_2 is depicted as V_1 being the ancestor of V_2). In other words, *when a view V_i is an ancestor of a view V_j in a hierarchy path, all*

tuple ids of V_i are also contained in the materialized tuples of V_j at this specific point in time (i.e., for the current extents of the two views). Following the same example, the hierarchy path H_1 from Figure 3.9 indicates that all the tuples materialized in V_1 are also materialized in V_2 and all tuples materialized in V_2 are materialized in V_3. Since, tuples materialized in V_1 are also in V_2 and all tuples from V_2 are materialized in V_3, by induction, all tuples in V_1 are also part of the materialized tuples in V_3 as well. Therefore, when an update affects a view that is part of a hierarchy path, then all its descendants will be affected by this update. On the other hand, if an update is not affecting the lowest view from a hierarchy path, then it will definitely not affect any of its ancestors. According to this, we propose a procedure for updating a number of views based on their nucleation. *We need to stress that the relationships are instance-dependent, i.e., they depend on the contents of the views at any time point and they need to be re-evaluated after each update occurs*. Also, this explains why we structure our discussion around *batches* of updates (as opposed to individual modifications). From the theoretical point of view, individual modifications are a special case of batch updates; at the same, tuple-at-a-time updates can be an overkill when compared to the processing of batches.

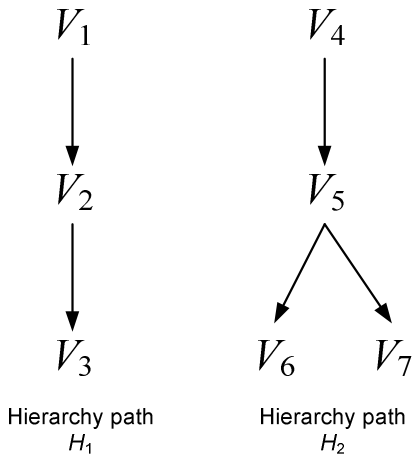


Figure 3.9. Hierarchies for Efficient View Updates.

Before proceeding to the algorithms that update the views of a set $\mathbb{V}$ we need to construct the algorithm that creates the hierarchy paths. Firstly, we describe the algorithm that constructs the hierarchy paths among the views from set $\mathbb{V}$.

Algorithm Create Hierarchy Paths

Input: A set of views V= {V_i(ID, X, Y, s_i) | 1 ≤ i ≤ m},

Output: a set of hierarchy paths **H** = {H_j | 1 ≤ j ≤ l}

Begin

```
1.   H = {Hj | Hj=Vi} //every view forms a hierarchy path
2.   For every Hj of H {
3.       Begin from root Vj of Hj {
4.           For every Hl ≠ Hj of H {
5.               Begin from root Vl of Hl {
6.                   CI=CheckInerstectionPointI(Vj,Vl)
7.                   if (CI = true){
8.                       Remove Hj, Hl from H
9.                       Hj = Merge { Hj, Hl }
10.                      Add Hj in H
11.                  }
12.                  Vl = Vl-1 //move a level down the path Hl
13.              } until CI = true
14.          }
15.          Vj = Vj-1 //move a level down the path Hj
16.      } until CI = true
17.  }
18.  Return(H)
```

End.

Algorithm 3.1 Algorithm Create Hierarchy Paths

Algorithms. How can we create a number of hierarchy paths according to the nucleation relationships for a set of *m* views V? Let the set of hierarchy paths be denoted as $H = \{H_j \mid 1 \le j \le l\}$ where $l \le m$. Each hierarchy path H_j is a partial order (denoted as $\prec$) among the views. Consider the hierarchy path H_1 denoted in Figure 3.9. Then, for views V_1, V_2, and V_3, partial orders are defined as: $V_1 \prec V_2 \prec V_3$. The algorithm *Create Hierarchy paths* initially treats each view of the set V as a hierarchy path of its own. Then, in an iterative manner it checks among views of hierarchy paths nucleation relationships exist. In case there is a partial order between a view of a hierarchy path and a view of another hierarchy path, the two hierarchy paths are merged into a new hierarchy path. The algorithm proceeds until all nucleation

relationships are considered. For each two hierarchy paths, the algorithm iteratively checks the views from one hierarchy path with the views of the other hierarchy path starting from the root and proceeding top-down until it finds a nucleation relationship.

Now, once the hierarchy paths have been constructed, we can update the views by taking into consideration the fact that any update not affecting a lower view in a hierarchy path will not affect any of its ancestors. In fact, the algorithm works in a bottom up way for every hierarchy path constructed. Initially, we check if the updated tuples ΔR^+ and ΔR^- of R, affect the lowest views from each hierarchy path. Then, the set of ΔR^+ tuples are split into two sets: (i) *Ignorable* set and (ii) *Affected* set. The *Ignorable* set contains all the tuples from ΔR^+ that do not affect the view, where the *Affected* set contains all the rest. In the next step, the algorithm proceeds by checking which updates affect the immediate ancestor of the previous view. However, there is no need to check every update from set ΔR^+. Instead, only updates contained in the *Affected* set are checked. Similarly to the previous step, the *Affected* set is now split into two new sets (i) *Ignorable* and (ii) *Affected*. The same procedure is conducted for the set ΔR^-, where in each step the set of possible updates are split into (i) *Ignorable* set and (ii) *Affected* set. This procedure is repeated for every hierarchy path, until the root of the path is reached. In addition, every time a view V is checked and the sets of *Ignorable* and *Affected* tuples are created, V is updated in regards to the *Affected* set of tuples.

Notice that in the *Create Hierarchy Paths* algorithm, if a view does not participate in any hierarchy path, then it creates a hierarchy path of its own. Therefore, there will not be any views not updated. In other words, in the worst case where no view nucleates another one, the algorithm is simplified to the naïve algorithm where every view is checked and maintained.

Algorithm Maintain View Updates

Input: Hierarchy paths H ,ΔR+ tuples inserted in R and ΔR- tuples deleted from R,

Output: maintain views

Begin

```
1.  Let V1 be the lowest view in a hierarchy path
2.  For all hierarchy paths Hj {
3.     For all V1 in Hj{
4.        V = V1
5.        Aff+ = ΔR+
6.        Aff- = ΔR-
7.        Ign+ = {}
8.        Ign- = {}
9.        do{
10.          For all tuples t+ in Aff+ {
11.             if (t+ not affects V){
12.                Ign+ = Ign+ ∪ {t+}
13.             }
14.          }
15.          For all tuples t- in Aff- {
16.             if (t- not affects V){
17.                Ign- = Ign- ∪ {t-}
18.             }
19.          }
20.          Aff+ = Aff+ - Ign+
21.          Aff- = Aff- - Ign-
22.          Update V with tuples in Aff+ and Aff-
23.          V = parent(V) //set the view V to be its immediate ancestor from hierarchy
                                path
24.       } until the root of the hierarchy path is reached
25.    }
26. }
```

End.

Algorithm 3.2. Algorithm Maintain View Updates

After each batch of updates has been checked and performed over the views, the hierarchy paths must be reconstructed. This is due to the fact that when updates occur in views then their relative positions and therefore, nucleation relationships are altered. In other words, before a new batch of updates is processed, the hierarchy paths should be appropriately reconstructed. To this end, we execute algorithm *Create Hierarchy Paths*.

Algorithm Check Intersection Point I

Input: Two materialized views $V_1(ID, X, Y, s_1)^{k1}$, with $s_1 = a_1 \cdot x + b_1 \cdot y$ and $V_2(ID, X, Y, s_2)^{k2}$, with $s_2 = a_2 \cdot x + b_2 \cdot y$ and maximum and minimum values of attributes *X* and *Y* in *R*,

Output: the position of the intersection point of V_1 and V_2

Begin

1. Let t_{k1} be the last tuple of V_1, $t_{k1}(x_{k1},y_{k1})=s_1(t_{k1})$
2. Let t_{k2} be the last tuple of V_2, $t_{k2}(x_{k2},y_{k2})=s_2(t_{k1})$
3. $x_I=(a_1 \cdot b_2 - a_2 \cdot b_1)^{-1} \cdot (b_2 \cdot s_1(t_{k1}) - b_1 \cdot s_2(t_{k2}))$
4. $y_I=(a_1 \cdot b_2 - a_2 \cdot b_1)^{-1} \cdot (a_1 \cdot s_2(t_{k2}) - a_2 \cdot s_1(t_{k1}))$ //compute coordinates for point I
5. if ($X_{min} \leq X_I \leq X_{max}$ and $Y_{min} \leq Y_I \leq Y_{max}$){
6. return(false);
7. }
8. else {
9. d_1 = dist(O(0,0), V_1);
10. d_2 = dist(O(0,0), V_2);
11. if ($d_1 < d_2$){
12. return(V_1 nucleates V_2);
13. }
14. else {
15. return(V_2 nucleates V_1);
16. }
17. return(true);
18. }

End

Algorithm 3.3. Algorithm Check Intersection Point *I*

3.6. Experiments

In this section, we report on the experimental assessment of (a) the estimation of the essential view size in order to sustain a high rate of updates and (b) updating multiple views by making use of the nucleation relationship among them. We start with presenting the experimental methodology and discuss our findings over the first set of experiments and then continue by describing the experimental methodology and results over the second set of experiments

3.6.1. Experimental Study of Sustaining High Rate of Deletions

Throughout this section we describe the experimental methodology and conclusions over the proposed method of sustaining a materialized view in the presence of high deletion rates. Our experimental study has been conducted towards assuring the following two goals:

- *Effectiveness*. The first desideratum of the experimental study has been the verification of the fact that the proposed method can accurately sustain intervals with high deletion activity in the workload. In other words, the experimental goal was to verify that a top-k materialized view contains at least k items, in at least 95% of the cases.
- *Efficiency*. The second desideratum of the experimental study has been the establishment of the fact that the computation of the exact number of auxiliary view tuples is faster than the computation of refill queries as proposed in the related literature. As well as the number of auxiliary view tuples is less than the number proposed in [YYY+03].

To achieve the first goal we have estimated k_{comp} via two methods: (a) without the fine tuning that uses the rates' variances (i.e., through equation 3.5) and (b) with this fine tuning (i.e., through equation 3.6). For both methods, we have computed the number of tuples that were deleted from the view, below the threshold of k.

In the context of the second goal, we have measured three metrics: (a) the memory overhead for k_{comp} and k_{comp} with tuning, measured as the number of extra tuples that we need to keep in the view, (b) the time overhead for computing k_{comp} and k_{comp} with

tuning as compared to the necessary time to compute the refill queries of [YYY+03] and (c) the time needed to compute the equation for k_{comp}. Again, we have evaluated these metrics using both the aforementioned methods.

In all our experiments we have used one relation $R(RID, X, Y)$ and one view $V(RID, score)$ with a formula $score = 3X+7Y$. The parameters that we have tested for their effect over the aforementioned measures are: (a) the number of relation tuples, (b) the number of materialized top-k results, (c) the fraction of the delete rate, over the insertion rate and (d) the percentage of the update stream over the relation size. We have not altered the time window T in our experiments; nevertheless, this is equivalent to varying the last parameter (denoted as λ), which measures the amount of modifications that take place as a percentage of the size of R. In other words, it is equivalent to increase the modifications number instead of reducing the window size.
We have tested the method over data whose attributes X and Y followed the Gaussian (with mean μ=50 and variance σ=10 for both X, Y), negative exponential (with a=1.5 for X and a=2.0 for Y) and Zipf distributions (with a=2.1 for both X, Y). The notation for the parameters and the specific values that we have used are listed in Table 3.1. All the experiments were conducted on a 2.8 GHz Pentium4 PC with 1 GB of memory

Table 3.1. Experimental Parameters.

Size of source table R (tuples)	$\|R\|$	$1x10^5$, $5x10^5$, $1x10^6$, $2x10^6$
Size of mat. view (tuples)	k	5, 10, 100, 1000
Size of update stream (pct over $\|R\|$)	λ	1/1000, 1/100
Deletion rate over insertion rate (ratio)	D/I	1.0, 1.5, 2.0

Effectiveness of the Method

The effectiveness of the method is demonstrated in Figure 3.10 and Figure 3.11. We present results organized by the data distribution of the attributes and compare two methods for computing k_{comp}, (a) the method including the fine-tuning part and (b) the

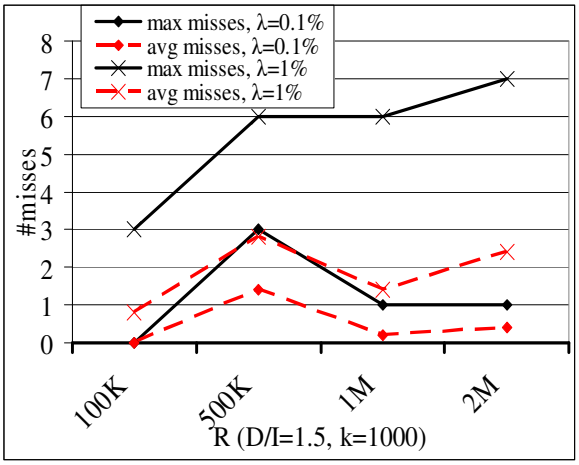
method simply based on equation 3.5. We have conducted the full range of combinations of the values listed in Table 3.1.

In Figure 3.10, we fix *D*/*I* to 1.5 and *k* to 1000 (the largest possible value) and vary the size of *R* (in the *X*-axis) and the update stream size (in different lines in the Figure). Each experiment has been conducted 5 times. We measure both the average and the maximum number of misses. In Figure 3.11 we report only the maximum number of misses, as it appears to be in analogy with the average in almost all the cases, and we vary *k* and *D*/*I*, while keeping *R* fixed to 1M rows and λ to 1%. The findings are as follows:

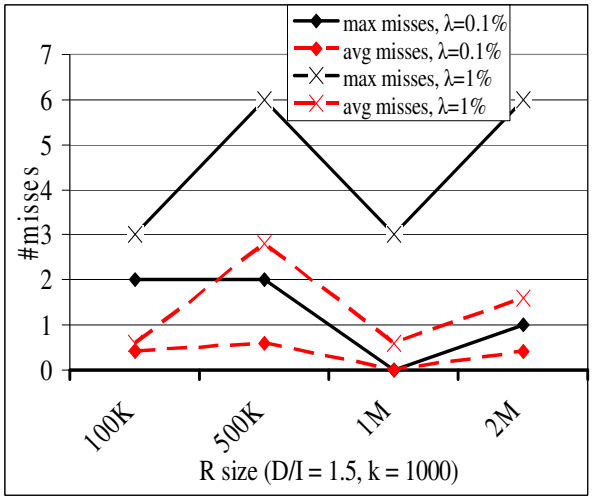
The fine tuning method gives 0 losses, and thus describes the bold line lying on top of the *X*-axis in Figure 3.10 and 3.11.

If the fine tuning was not included, misses would have been encountered. In cases where insertions are close to deletions, the underestimation of the value of k_{comp} would lead to potentially important errors (in the Zipf case, errors have come up to 9 misses which is almost 1% of the top-*k* view size).

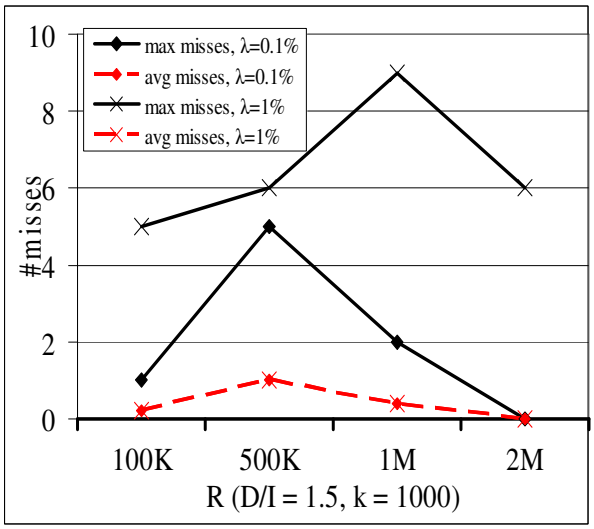
It is also interesting how the distribution of data affects the stability of the error (Gaussian seems to converge, as expected, whereas the Zipf drops when the percentage of *k* is small over *R*, as the hot values are rather fixed).



Gaussian

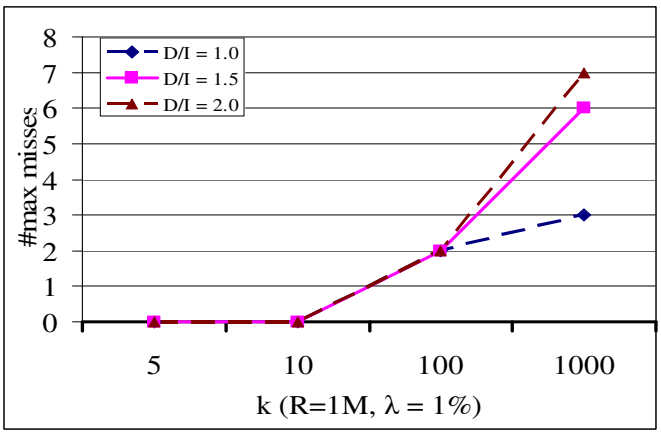


Negative Exponential

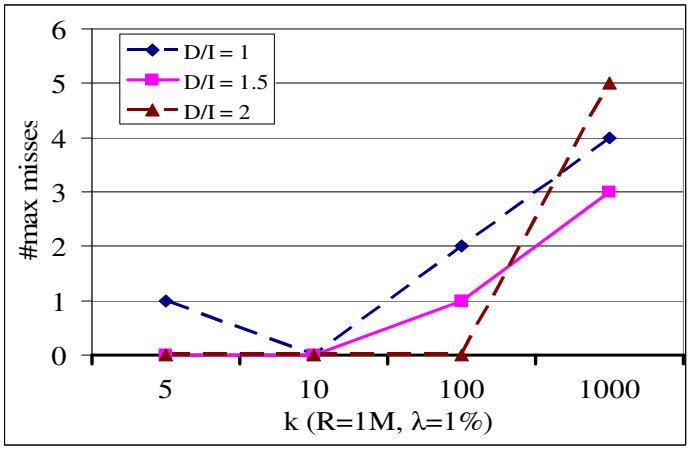


Zipf

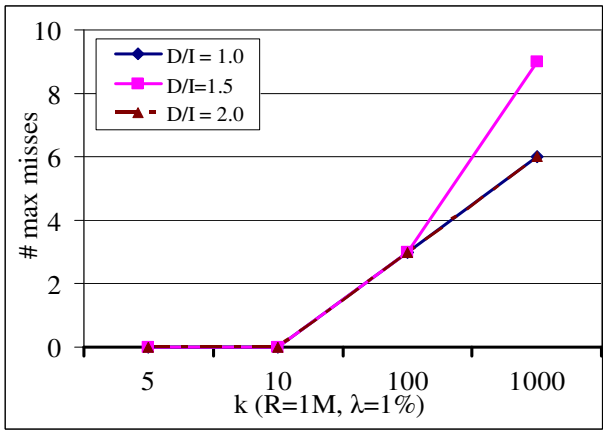
Figure 3.10. Maximum and Average Misses as a Function of $|R|$ and λ.



Gaussian



Negative Exponential



Zipf

Figure 3.11. Maximum Misses as a Function of *k* and *D/I*.

Our experimental study has also explored the case of larger workloads of updates that may occur in the base relation. Specifically, the experiments were conducted by

making use of three different scenarios of possible update workloads. All the scenarios were applied over a database of 1 million records with attributes x and y following the Gaussian distribution (in any case, the distribution of data does not have an effect to the effectiveness of the method as our aforementioned experiments have demonstrated). Every experiment was conducted 100 times in order to eliminate cases where the actual values of the tuples inserted or the tuples deleted contribute significantly to the experimental results. All three workloads contain 91 thousand updates occurring in the base relation and in all three of them the insertions and deletions do not occur uniformly. There are peaks and valleys of high insertion and deletion rates throughout all three scenarios. The first workload (denoted as W_1), depicted in Figure 3.12, contains updates where insertions and deletions are of the same size (specifically, 45500 insertions and 45500 deletions). The two other workloads are constructed in order to test the method to extreme cases. In the second workload (denoted as W_2), shown in Figure 3.13, deletions are approximately twice as many as the insertions (specifically, 60700 deletions and 30300 insertions). The third workload (denoted as W_3), shown in Figure 3.14, is the inverse of workload W_2. Specifically, W_3 occurred by replacing in workload W_2 deletions with insertions and vice versa. Thus, W_3 constitutes of 60700 insertions and 30300 deletions, having a ratio of deletion rate over insertion rate approximately equal to 0.5.

In order to assure that a large number of updates will affect the top-k view results, we have set the parameter k to 1000 tuples. The resulting numbers of tuples that are either inserted or deleted in the extent of the top-k view are depicted in Figure 3.15 for all the workloads.

For all these three workloads, we have counted the number of misses that occurred (as a measure of how often we would have to run refill queries) as well as the memory overhead for k_{comp} and k_{comp} with tuning, measured as the number of extra tuples that we need to keep in the view. Our findings are as follows:

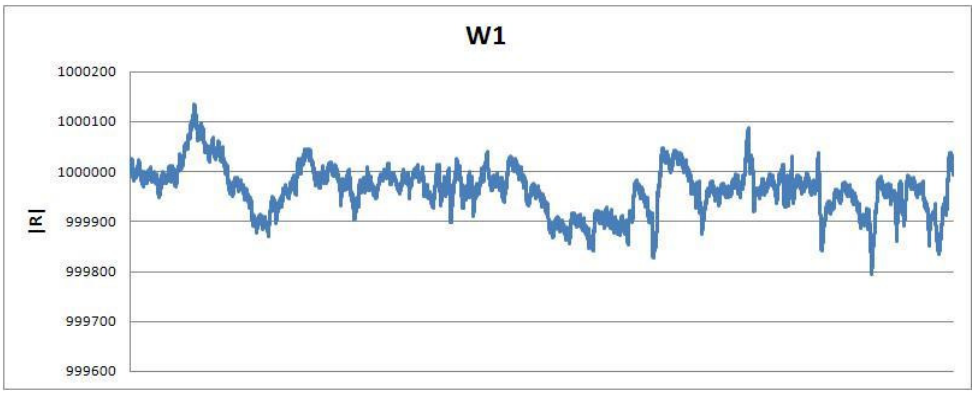


Figure 3.12. Size of Relation R ($|R|$) over Time as Insertions and Deletions Take Place for Workload W_1 Having a Ratio of Deletion Rate over Insertion Rate D/I =1.0.

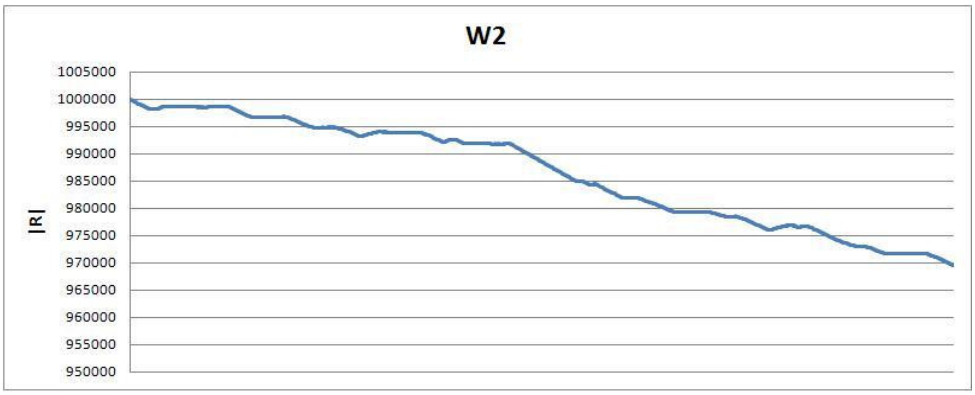


Figure 3.13. Size of Relation R ($|R|$) over Time as Insertions and Deletions Take Place for Workload W_2 Having a Ratio of Deletion Rate over Insertion Rate $D/I \approx 2.0$.

Figure 3.14. Size of Relation R ($|R|$) over Time as Insertions and Deletions Take Place for Workload W_3 Having a Ratio of Deletion Rate over Insertion Rate $D/I \approx 0.5$.

- Concerning the number of misses, the number of missed tuples was exactly zero for all the three workloads and in each one of the 100 runs of every workload.
- Concerning the memory overheads, the extra tuples that we had to store for the top-1000 view of our experiments was quite low. The results for k_{comp} and k_{comp} with tuning are shown in Figure 3.16 for all three workloads. Observe that in all three scenarios the number of extra tuples materialized over 1000 tuples, due to the extra tuning (i.e., the difference of k and k_{comp} with tuning) does not exceed 188 tuples. Specifically, the mixed workload W_1 requires 137 extra tuples (i.e., a 13.7% increase over k). Workload W_2 that is heavy on deletions (and therefore requires a provision for a larger k_{comp}, in order to sustain the high deletion rate) requires an increase of 18.8% (although the deletion rate is twice as high as the insertion rate). Workload W_3 which is heavy on insertions only requires an increase of 0.89% over k. In particular, in workload W_3, equation 3.5 gives for k_{comp} the value of 971 tuples instead of 1000 tuples, due to the high insertion rate in regards to the deletion rate. However, in the experimental setup we have used as k_{comp} the maximum value between k and the computed value of k_{comp} from equation 3.5.

Efficiency of the Method

We compared the values of k_{comp} without the fine tuning (i.e., through equation 3.5) and k_{comp} *tuning* with this fine tuning. The comparison of the above values was conducted for all three distributions as well as for all parameters listed in Table 3.1. Due to the fact that our equation is independent of the distribution the tuples follow we only present some indicative results. In Figure 3.17 we compare k_{comp} and k_{comp} *tuning* (a) as a function of k, where the size of R is 100000 tuples and (b) as a function of the size of R where we have fixed k=1000. For both of them and for all possible values of D/I the size of the update stream λ is 1% and the distribution is the Negative exponential. In Figure 3.17 (a) the Y-axis denotes the percentage of extra tuples. From both graphs in Figure 3.17 we observe that k_{comp} is slightly greater than k and k_{comp} *tuning* is slightly greater than k_{comp} in all cases. The number of the auxiliary tuples in the view (i.e., k_{comp} and k_{comp} *tuning*) in the maximum case is approximately 1% and

6% respectively. Thus, the number of the auxiliary tuples does not cause a great extra memory cost.

Figure 3.15. Average Number of Insertions and Deletions that Affect the Top-*k* Tuples in the View.

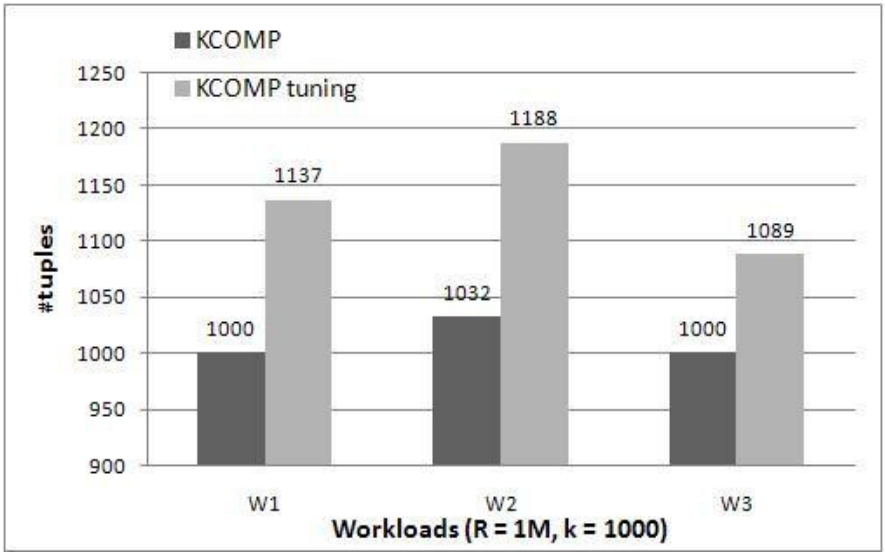
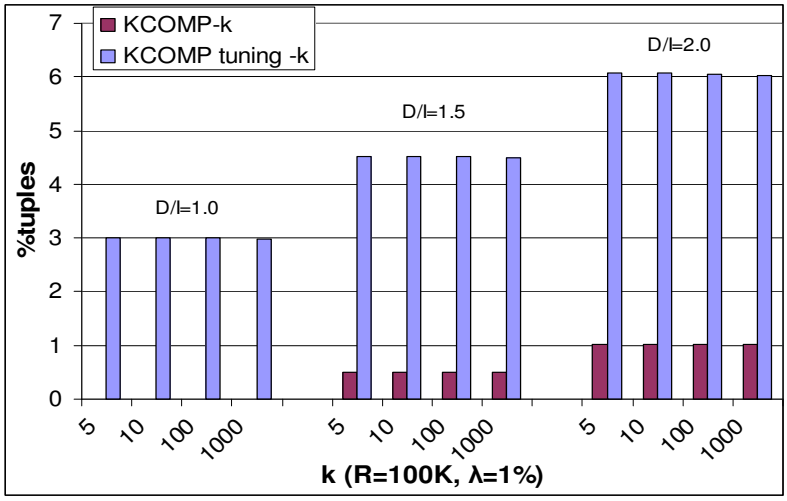
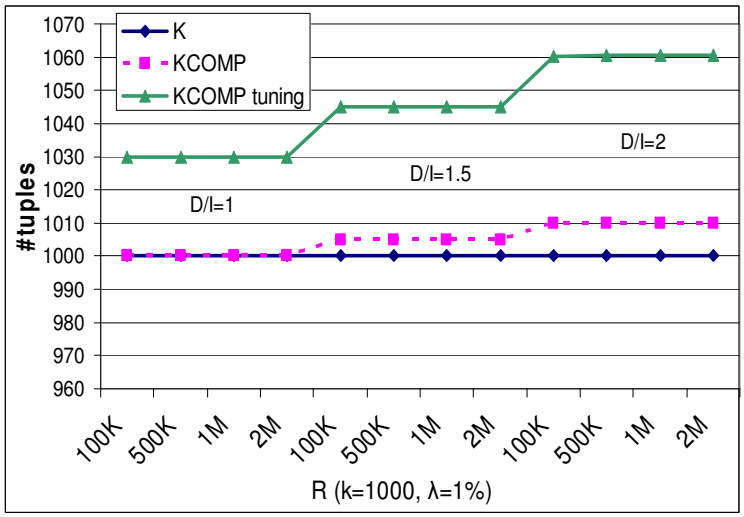


Figure 3.16. Memory Overhead Expressed as the Number of Tuples Stored in the View.



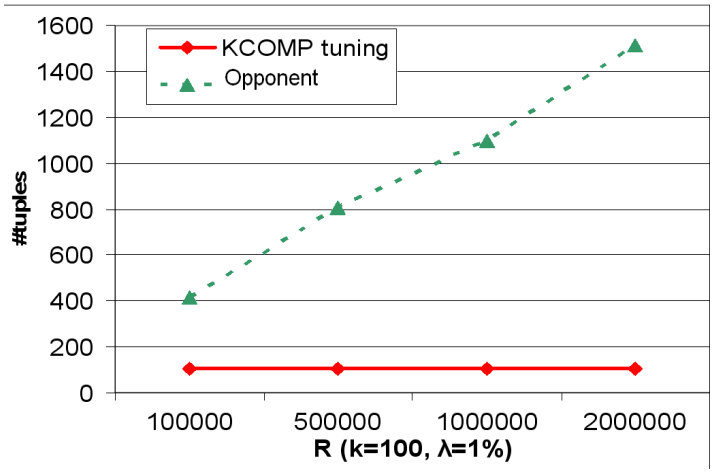
(a) Percentage of extra tuples as a function of k and D/I



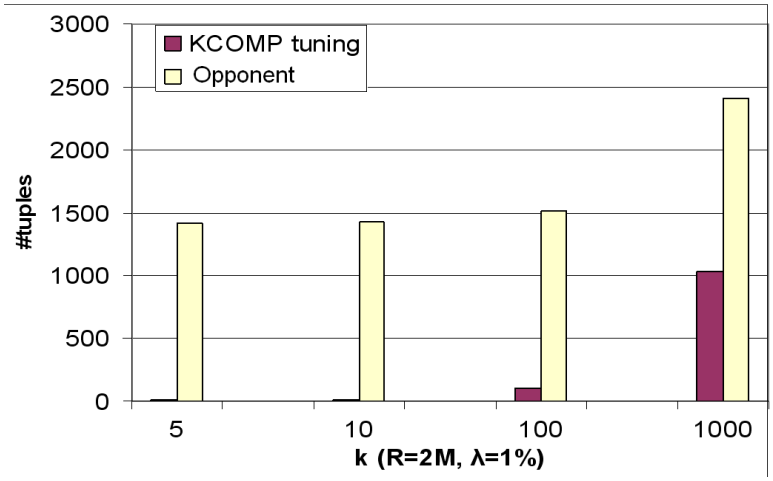
(b) Number of extra tuples as a function of R and D/I

Figure 3.17. Comparison of k, k_{comp}, and k_{comp} with Tuning.

In Figure 3.18, we compare the value of k_{comp} *tuning* with the one proposed by [YYY+03]. Again, we compare the above (a) as a function of k where the size of R is set to 2M (the largest possible value) and (b) as a function of R where k is fixed to 100. In both graphs the distribution is the negative exponential. The parameter D/I=1, since it is the only value that can be compared with the proposed method in [YYY+03]. We notice that the number of tuples proposed by [YYY+03] is significantly larger than the one proposed in our method. Thus the memory cost in our method is considerably less.



(a) Number of extra tuples as a function of R



(b) Number of extra tuples as a function of k

Figure 3.18. Comparison of k_{comp} with Tuning and [YYY+03].

The second part of our experimental results had to do with the comparison of the time needed to compute the value of k_{comp} as compared to the time needed to re-compute the view as part of a refill query. Figure 3.19 measures the computation time needed for the view computation for a value of k in microseconds. On the contrary, the time necessary to perform the computation of k_{comp} has consistently been negligible (practically 0 in all occasions).

N	K	Gauss	Negative exponential	Zipf
100K	5	328000	348500	242000
100K	10	333000	345667	239667
100K	100	335500	343000	239667
100K	1000	395333	406000	299500
500K	5	1650667	1715500	1216333
500K	10	1650667	1713000	1208333
500K	100	1653167	1710500	1205667
500K	1000	1736667	1796167	1291833
1M	5	3298667	3429000	2427167
1M	10	3301333	3426667	2429667
1M	100	3304000	3439500	2422167
1M	1000	3403167	3520500	2606667
2M	5	6650667	6900500	5406333
2M	10	6653167	6900833	4909000
2M	100	6747167	6906000	4906500
2M	1000	6895500	7082833	4992167

Figure 3.19. Time to Build the Top-k View (microseconds).

3.6.2. Experimental Study for Multiple Views Updates

In this section we describe the experimental study and findings of maintaining multiple views by making use of the nucleation relationship among them. The experimental study has focused on proving the correctness and efficiency of the proposed method. We have implemented the algorithms described in section 5.4 and compared them with a base method which we refer to as naïve method. The naïve method checks a batch of updates over each view independently and applies them appropriately. In order to test the correctness of the proposed nucleation method we have compared the results of the updated views with the results of applying the updates over each view independently and the outcome has been absolute identical. Having secured the correctness of our algorithms' implementation the rest of the experimental study focused on proving the efficiency of the proposed method in terms of the time needed to apply updates over multiple views. Our experiments have

demonstrated that, indeed when batches of updates are applied to a multitude of top-k views, using the nucleation relationships is faster than the naïve method. Under the context of proving the efficiency of the nucleation method, we have measured the time needed to maintain multiple views in the presence of updates over the base relation, for both the nucleation and naïve method.

In all our experiments we have used a relation $R(RID, X, Y)$ where the attribute values of X and Y were generated randomly from the interval [0, 10000]. All the views needed to be maintained are of the form $V(RID, X, Y, score)$ where *score* is a weighted sum over the attributes X and Y. Particularly, the scoring function of the views is of the form $score = w_x \cdot X + w_y \cdot Y$, with the parameters w_x and w_y being randomly generated from the interval [0, 1]. The parameters that we have tested for their effect on the efficiency of the view refreshment are: (a) the number of relation tuples, (b) the maximum number of materialized top-k results within a set of views expressed as a percentage over the relation size, (c) the number of materialized views needed to be maintained and (d) the percentage of the insertion stream over the relation size. We have kept the fraction of the delete rate, over the insertion rate constant and equal to 0.5.

The notation for the parameters and the specific values that we have used are listed in Table 3.2. All of the experiments were conducted on a 2.53GHz Core Duo PC with 3.12 GB of memory.

Table 3.2. Experimental Parameters.

Size of source table R (tuples)	$\|R\|$	$2x10^5$, $3x10^5$, $4x10^5$
Max size of mat. tuples (pct over |R|)	max_k	1/100, 1/1000
Number of views	M	100, 1000
Size of insertion stream (pct over $\|R\|$)	λ	1/10, 1/100, 1/1000

In all the experiments the measure for time is expressed as number of seconds. The comparison of the time needed for the two methods has been conducted for all

possible combinations of the above parameters listed in Table 3.2. We run every experiment five times and the results presented here are the average time. In all charts of Figure 3.20 the *Y*-axis indicates the time needed for the two methods to apply the updates. The *X*-axis shows (a) the size of the source table *R* and (b) the size of the insertion stream. Specifically, for each possible value of |*R*| (i.e., 200, 300 and 400 thousand tuples) *X*-axis also indicates the stream of insertions for all three possible values (i.e., 1/10, 1/100 and 1/1000 percentage of |*R*|). Since, the fraction of the deletion rate over the insertion rate is set to be 0.5 the number of updates occurring can be calculated as 1.5 times the value of parameter λ, times the value of parameter |*R*|. The naïve method is denoted with the darker grey color, whereas the nucleation method is presented with the lighter grey color. In all charts we can notice that the nucleation method is faster than the naïve. The title of each chart also clarifies the fixed value of the parameters *M* and max_*k*.

Graphs (a) and (b) in Figure 3.20 demonstrate the time needed for applying updates over a set of 100 views. In these two graphs the maximum number of tuples materialized in each view expressed as a rate over |*R*| is 0.1 % and 1% respectively. In graph (a) of Figure 3.20 the ratio time between the two methods is not that significant but still the nucleation method is faster than the naïve method. In graph (b) of Figure 3.20 we observe that time needed for nucleation method is approximately half the time needed for the naïve method. This is due to the fact that the number of views is 100 and in each view the maximum number of tuples materialized is only 200, 300 and 400 respectively for each size of *R*. In other words, the larger the extent of the views (due to the size of *k*), the larger the benefits from the nucleation method are.

In graphs (c) and (d) of Figure 3.20 we see the time needed for the two methods over a set of 1000 views (as opposed to 100 views for the cases of (a) and (b)). The maximum value of tuples materialized in each view is set to 0.1 % and 1% respectively. In graph (c) the ratio time between the two methods ranges approximately between 2 and 4. In graph (d), the time needed for the nucleation method is approximately 4 times faster than the naïve method. Again, nucleation scales up much better than the naïve method. Moreover, if one reads Figure 3.20 vertically, one can observe that the scaling capabilities involve both the extend of the

view and the number of materialized views; in fact, the improvements in cases (c) and (d) where a larger number of views is maintained are significantly higher than the improvements of cases (a) and (b) where a smaller number of views is maintained.

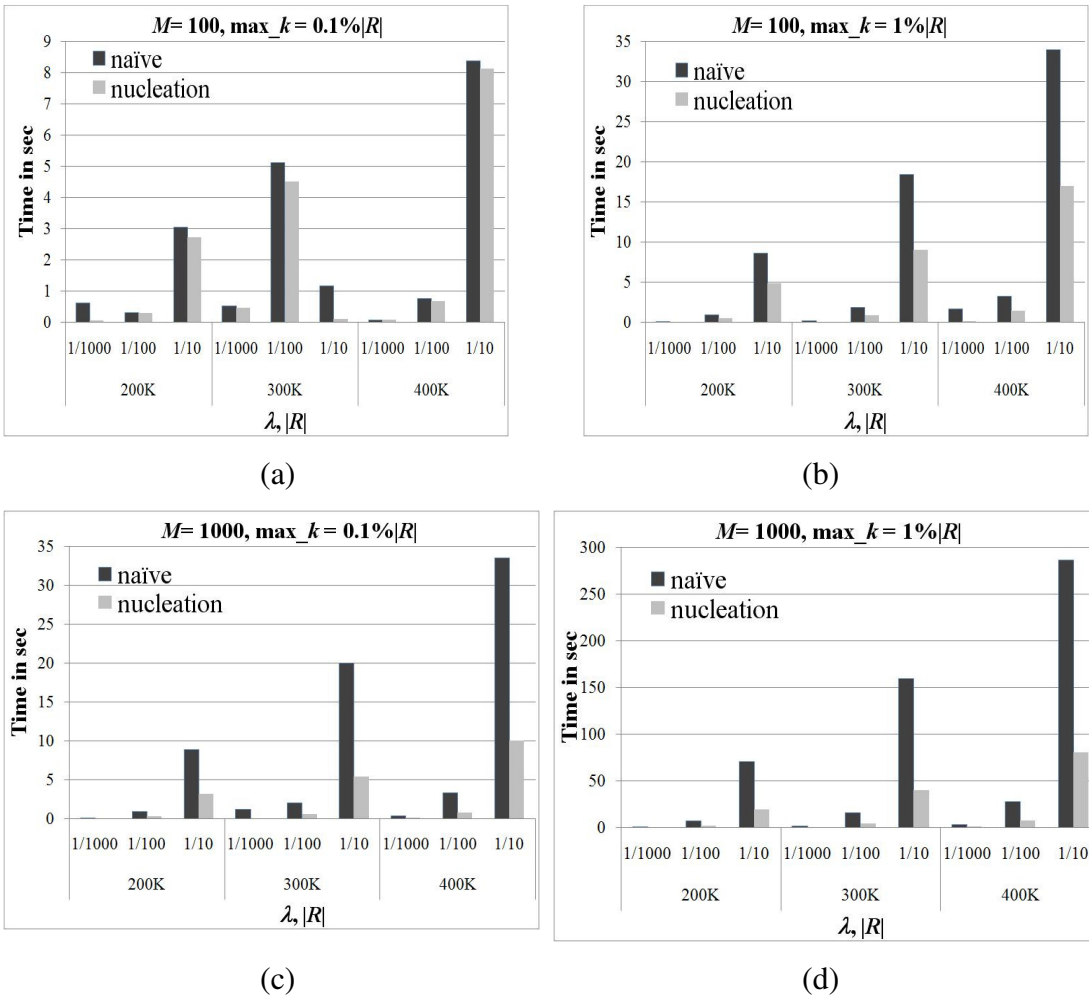


Figure 3.20. Comparison between Naive and Nucleation Method. All Graphs Show the Time of Applying Updates as a Function of Insertion Size and |*R*|.

In all the graphs of Figure 3.20 we can observe that the time needed for the naïve method scales up linearly with respect to the number of updates occurring in the base relation. Considering the nucleation method the time scales up almost linearly as well.

3.7. Chapter Summary and Findings

In this Chapter we have handled the problem of maintaining materialized top-*k* views and provided results in two directions. The first problem we have been concerned with has to do with the maintenance of top-*k* views in the presence of high deletion rates. We have provided a principled method that complements the inefficiency of the state of the art independently of the statistical properties of the data and the characteristics of the update streams. The method comprises the following steps: (a) a computation of the rate that actually affects the materialized view, (b) a computation of the necessary extension to *k* in order to handle the augmented number of deletions that occur and (c) a fine tuning part that adjusts this value to take the fluctuation of the statistical properties of this value into consideration. The second problem we have been concerned with concerns the case of multiple top-*k* views and their efficient maintenance in the presence of updates to their base relation. We have provided theoretical guarantees for the establishment of the effect of updates to a certain view, whenever we know that another view has been updated. We have also provided algorithmic results towards the maintenance of a large number of views, via their appropriate structuring in a hierarchy of views. Our experiments have shown that our method accurately sustains intervals with high deletion activity in the workload and specifically in at least 95% of the cases there were top-*k* materialized views that contained at least *k* items. The experiments indicate that our method outperforms the state-of-the-art in terms of efficiency as the computation of the exact number of auxiliary view tuples has shown to be faster than the computation of refill queries as proposed in the related literature. At the same time, the number of auxiliary view tuples has been less than the number proposed in [YYY+03]. Moreover, the fine tuning method we proposed, gave zero losses. The experiments for updating multiple views revealed that the time needed through the nucleation method outperforms the naïve method.

CHAPTER 4. SIMILARITY MEASURES FOR MULTIDIMENSIONAL DATA

In our deliberations so far, we have dealt with our data as points in the multidimensional space. Each top-*k* view or query is a collection of such points, ranked according to a scoring function. So far, we have been interested on the suitability of a view to answer a query as well as the refreshment of such views. Still, inherent to the problem of view management is the answer to the question "How similar are two data collections?". If a query is given to us and we have to suggest similar views to the user to explore, or we have to decide the most similar views in order to answer a query, which ones would we use? To answer the question we need a fundamental insight on the question "which is the best distance function for two data collections?" We are interested in discovering what users prefer and not which function is more efficiently computed or has the nicest properties.

In order to achieve an answer to this question we resort to the simplest framework that can be given to a user to work with and that is OLAP Cubes and hierarchical multidimensional spaces. OLAP is preferred for simplicity as it organizes data in dimensions and measures that are most intuitive to users. We model a collection of data in the form of a multi-dimensional array called *Cube*. Each cell of the cube contains data that are called *measures* of the cell. The cell is uniquely defined by its

coordinates as values of the dimensions of the cube. A dimension D is a lattice of a finite subset of levels and a partial order defined among the levels. Formally, the notions of dimension and Cube are defined as follows.

Definition 4.1 (dimension) [VaSk00]. A dimension D is a lattice $(\mathcal{L}, \prec)$ such that: $\mathcal{L}= (L_1, ..., L_n, ALL)$ is a finite subset of levels and $\prec$ is a partial order defined among the levels of $\mathcal{L}$, such that $L_1 \prec L_i \prec ALL$ for every $1<i\leq n$. We require that the upper bound of the lattice is always the level *ALL*, so that we can group all the values of the dimension into the single value 'all'. The lower bound of the lattice is called the detailed level of the dimension.

Definition 4.2 (Cube) [VaSk00]. A cube c over the schema $[L_1, \ldots L_n, M_1, \ldots, M_m]$, is an expression of the form: $c= (DS^0, \varphi, [L_1, \ldots L_n, M_1, \ldots M_m], [agg_1(M_1^0, \ldots, agg_m(M_m^0)])$, where DS^0 is a detailed data set over the schema $S= [L_1^0, \ldots L_n^0, M_1^0, \ldots M_m^0]$, $m\leq k$, φ is a detailed selection condition, $M_1^0, \ldots M_m^0$ are detailed measures, $M_1, \ldots, M_m$ are aggregated measures, L_i^0 and L_i are levels such that $L_i^0 \prec L_i$, $1<i\leq n$ and agg_i, $1<i\leq m$ are aggregated functions from the set $\{sum, min, max, count\}$.

Then the research question is rephrased: *given two sets of points in a multidimensional hierarchical space, what is the distance between these two collections?* The above research problem is generic and has several applications in domains such as multimedia information retrieval, statistical data analysis, scientific databases and digital libraries [ZADB06]. In such applications, where contemporary data lead to huge repositories of heterogeneous data stored in data warehouses, there is a need of similarity search that complements the traditional exact match search. For example, one might easily envision a context where a user of an OLAP tool is proactively informed on reports that are similar to the one she is currently browsing.

In this chapter, we address the problem by (a) organizing alternative distance functions in a taxonomy of functions and (b) experimentally assessing the effectiveness of each distance function via a user study. The novelty of our work is not in the suggestion of new distance functions, but rather, it lies (a) in the adjustment of existing distance functions in the OLAP setting and (b) in their evaluation –via two

user studies- in order to discover which distance function is mostly preferred by the users.

In the related literature there are a number of papers that have pointed out the necessity of having appropriate similarity measures in order to discover objects that are similar to each other and measure in a quantitative way the distance among them. Most of them examine similarity measures used between objects that are described from various features such as in image retrieval or data that are stored in a hierarchical taxonomy. Notably, [SaJa95] and [SaJa99] describe how similarity measures used by human perception and computer science follow different properties. The authors provide a collection of references where the metric axioms have been refuted. Computer scientists in the areas of data mining and information retrieval have also considered the problem of introducing appropriate similarity measures. Few papers have associated the areas of mathematics and computer science and have introduced similarity measures for lattices by mapping them with semantic hierarchies [Josl04].

So far, related work have dealt with similar problems in different ways; however, this particular problem has not been dealt per se. Specifically, Sarawagi in [Sara99] and [Sara01] has dealt with the problem of discovering interesting patterns and differences within two instances of an OLAP cube. The DIFF and RELAX operators summarize the difference between two sub-cubes in order to discover the reason of abnormalities within the measures of two given cells. The only common element of this work with ours is the usage of the Manhattan distance in the process of discovering abnormalities. Our work addresses the problem of finding the appropriate distance function among a great variety of functions in order to compute the similarity between two given OLAP cubes. Giacometti et al. [GMNS09] propose a recommendation system for OLAP queries by evaluating distances between multidimensional queries. This work involves the distance between queries whereas our work involves distance functions between the data of multidimensional queries. Li et al. in [LiBM03] describe the semantic similarity between ontologies. In contrast to our work, they consider a limited set of functions whereas we have a wider range of distance functions and our work focuses on distances between data of an OLAP cube.

The main findings of our approach are due to two user studies that have been conducted to assess which distance functions appear to work better for the users (Section 4.3). The first experiment involved 15 users of various backgrounds and the *Adult* real dataset [FuWY05]. Each user was given 14 scenarios that contained a reference cube as well as a set o variant cubes, each associated with a distance function. The task of the user was to select a cube from the set of variant cubes that seemed more similar to the reference cube. The diversity of users and data types contained in the experiment was taken into consideration in order to discover which distance function between two values of a dimension is preferred depending on the user group or the type of data. The first user study showed that all distance functions under test were used at least once, but there were a couple of distance functions that were most preferred among the others. In particular, the users seemed to prefer distance functions that express the similarity between two cubes based either on the hierarchical shortest path, or with regard to ancestor values.

The second user study involved 39 users and the results of the first user study were taken into account. Each user was given 14 scenarios that contained a reference cube and three variant cubes. The purpose of this second user study concerns the most preferred distance function between two data cubes. Two distance functions have been in the center of attention in this study: the Hausdorff distance function and the *closest relative* function that sums the individual distances of cells of the two cubes. The latter has been selected by users at a remarkably higher percentage of occasions than the former (57% vs. 38%); however, if one considers the winner per scenario the result is only 6 vs. 5 in favor of closest relatives. Thus, we conclude that although the closest relative has an advantage over Hausdorff, this cannot be overemphasized.

Roadmap. We start by (Section 4.1) providing a taxonomy of distance functions for cubes based on a detailed study of the characteristics of dimension hierarchies, levels and members. At first, we organize our families of functions as follows: Initially we describe functions that can be applied between two specific values that belong to the same dimension (Section 4.1.1). Following, we describe distance functions that are applied between two cells of a cube (Section 4.1.2) and then distance functions

between two OLAP cubes (Section 4.1.3). In Section 4.2 we introduce the method that is used in order to map the cells of one cube to the cells of another cube. We refer to this method as *Cell Mapping*. Section 4.3 presents the user study experiments along with the results of the most preferred distance functions. All the results and the user study experiments can be found in the web page [Baik11]: http://www.cs.uoi.gr/~ebaikou/publications/2011_ICDE/ that includes questionnaires and findings, too. Finally, in Section 4.4 we summarize our findings.

4.1. Distance Families

In this section, we organize the distance functions that can be used to measure the distance between two cubes in a taxonomy. The formal foundations of modeling multidimensional spaces and cubes are based on an existing model in the related literature [VaSk00]. We build our taxonomy of distances progressively: In Section 4.1.1, we describe the functions that can be applied between two values for a given dimension. In Section 4.1.2 we provide a taxonomy for distance functions between two cells of cubes and in Section 4.1.3 a taxonomy for distance functions between two OLAP cubes. The distance functions described are all normalized within the interval [0, 1] and in many cases, such as in the *weighted sum* distance function, weight factors may be used. The normalization and usage of weight factors in the distance functions is not obligatory. Throughout all our deliberations we will refer to two reference dimensions, *Time* and *Location*. The hierarchies of these dimensions are shown in Figure 4.1. In more detail, the *Time* dimension hierarchy consists of 5 levels. The levels of *Time* are *Day*(L_1), *Week*(L_2) and *Month*(L_2), *Year*(L_3) and *All*(L_4). The dimension *Location* consists of four levels of hierarchy which are *City*(L_1), *Country*(L_2), *Continent*(L_3) and *All*(L_4). Figure 4.2 illustrates the lattice of the dimension *Location* at the instance level.

4.1.1. Distance Functions between two Values

In this section, we specify the distance functions that can be applied over two specific values of a dimension. In order to clarify things, distance functions described in this

section apply only between two dimension values and not between measure values of a cube.

Assume a dimension D, its lattice of level hierarchies $L_1 \prec L_2 \prec \ldots \prec ALL$, and two specific values x and y from levels of hierarchy L_x and L_y respectively. We classify the distance functions in the following categories: (*1*) *locally computable* and (*2*) *hierarchical computable* distance functions.

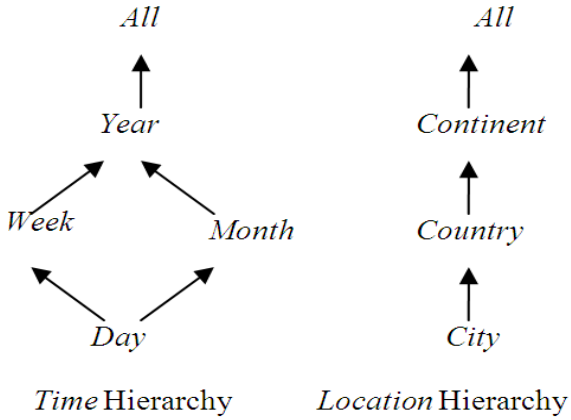


Figure 4.1. The hierarchy of levels for dimensions *Time* and *Location*

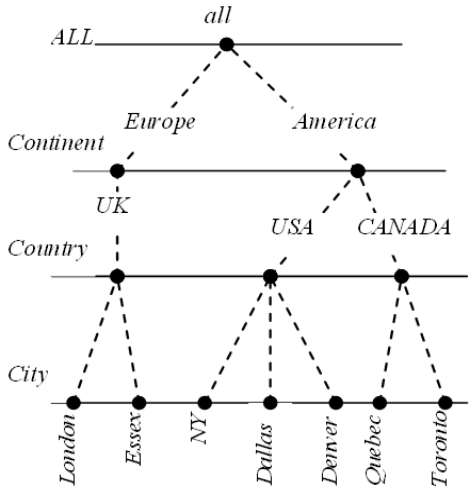


Figure 4.2. Values of the *Location* Dimension.

Locally Computable Distance Functions. The first category of locally computable distance functions can be divided into three subcategories: (a) Distance functions with explicit assignment of values, (b) Distance functions based on attribute values and (c) Distance functions based on the values of x and y.

Distance Functions with Explicit Assignment of Values. The functions of this category explicitly define n^2 distances for the n values of the dom (L_i) (the compared values must belong to the same level of the hierarchy). This requires $dom(L_i)$ to be a finite set. For example, the distance between two cities can be explicitly defined via a distance table.

Distance Functions based on Attribute Values. Assume a level whose instances are accompanied with a set of attributes. Then, every level instance can be described as a tuple of attribute values. In this case, the distance between the two values x and y can possibly be expressed with respect to their attribute values via simple distance function applicable to the attributes' domains (e.g., simple subtraction for arithmetic values). For instance, assume a dimension *Products* accompanied with an attribute *Weight* which describes the weight of the products and assume a level of hierarchy of the dimension named *Drinks*. In addition, assume two specific values x = 'milk' and y = 'orange juice' where their weight attributes are $x.weight$ = 500 and $y.weight$ = 330 respectively. Then, the distance between these two values can be expressed according to their weight attribute by making use, for instance, of the Minkowski distance function which is described in the following subsection. Thus, the distance between the values x and y can be defined as $|x.weight - y.weight| = 170$.

Distance Functions based on the Values x and y. In this subcategory, the distance between two values may be expressed through a function of their actual values whenever this is possible. This function is applicable for all type values even for nominal values. A first option is to use of the simple identity function, resulting in a value from the set {0, 1}, where $dist(x, y) = 0$ if $x=y$, or $dist(x, y) = 1$ if $x \neq y$.

Another option is to make use of the Minkowski family distance functions especially when the values are of interval type. Minkowski family distance functions can be

applied between two ordinal type values under the condition that the ordinal values have been mapped to the set of integer numbers. In this section, since the distance function is applied for two specific values, all types of Minkowski distances reduce to the Manhattan distance which is $|x-y|$. In order to normalize this function within the interval [0, 1], we can divide the distance value by the difference between the maximum and minimum values of the level where x and y belong to.

Hierarchical Computable Distance Functions

The second category of hierarchical computable distance functions can be divided into four subcategories: (a) Distance functions with respect to an aggregation function, (b) Distance functions with respect to hierarchy path, (c) Percentage distance functions and (d) Highway distance functions.

Distance functions with respect to an Aggregation Function. The distance for two values that do not belong to the detailed level L_1 can be expressed with respect to an aggregation function (e.g., *count*, *max*) applied over the descendants of the two values in a lower level of hierarchy.

Assume an instance x from level L_i and $desc_{L_L}^{L_i}(x)$ the set of its descendants, where L_L is any lower level of L_i. The result of applying an aggregation function over the set $desc_{L_L}^{L_i}(x)$ is denoted as $x_{aggr} = f_{aggr}(desc_{L_L}^{L_i}(x))$. Assume two values x and y with $x_{aggr} = f_{aggr}(desc_{L_L}^{L_i}(x))$ and $y_{aggr} = f_{aggr}(desc_{L_L}^{L_j}(y))$, where L_L could be any lower level of L_i and L_j, $x \in L_i$, $y \in L_j$ and f_{aggr} denotes an aggregation function such as *count*, *min*, *max*, *avg* or *sum*. The distance between the values x and y can now be expressed according to the following formula: $dist(x, y) = g(x_{aggr}, y_{aggr})$, where the function g can be computed from the locally computable functions. The normalized form of this function, within the interval [0, 1], can be expressed as $dist(x, y) = \frac{g(x_{aggr}, y_{aggr})}{max\{g(a_{aggr}, b_{aggr})\}}$, where a and b are any possible values from the same level of hierarchy as x and y, i.e., $a, b \in L_i$.

Distance Functions with respect to Hierarchy Path. The distance between two values x and y can be expressed according to the length of the path in the hierarchy that connects them. Several distance functions and combinations falling into this subcategory were described by Li, Bandar and McLean in [LiBM03]. Here, we describe the distance functions that can be applied between two values x and y from a hierarchy, (a) with respect to the length of the path in the hierarchy, and, (b) with respect to the depth in the hierarchy path. Assume two values x and y such that $x \in L_x$ and $y \in L_y$. We denote the *Lowest Common Ancestor* of x and y as $lca(x,y)$.

The lowest common ancestor $lca(x,y)$, of two values x and y --where $x \in L_x$ and $y \in L_y$, $lca(x,y) \in L_z$ and L_z is any non lower level of L_x and L_y, $L_z \succ L_x$, L_y -- is a value such that:

$$lca(x,y)=\{z | z = anc_{L_x}^{L_z}(x) \wedge z = anc_{L_y}^{L_z}(y) \wedge (\nexists\ z' \mid z' = anc_{L_x}^{L_z}(x) \wedge z' = anc_{L_y}^{L_z}(y) \wedge L_{z'} \prec L_z\}$$

The distance between the values x and y can be expressed with one of the following formulas:

$$d_{path}(x,y) = \frac{w_x * | path(x, lca) | + w_y * | path(y, lca) |}{(w_x + w_y) * | path(ALL, L_1) |}$$

$$d_{depth}(x, y) = \frac{| path(lca, L_1) |}{| path(ALL, L_1) |}$$

The first formula indicates that the distance is expressed as the weighted sum of the length of the path from the values x and y to their lowest common ancestor lca. The second formula indicates that the distance of the values is expressed as the length of the path of the lowest common ancestor lca from the detailed level L_1 of the hierarchy. These two functions are normalized in the interval [0, 1] by making use of the height of the hierarchy. Specifically, the first formula is divided by $(w_x + w_y) * | path(ALL, L_1) |$ whereas the second formula is divided by $| path(ALL, L_1) |$. As an example, assume two values x='NY' and y='Canada' from the hierarchy *Location* denoted in Figure 4.2 where their lowest common ancestor is the value lca = 'America' from the level *Continent*. For simplicity, assume the

weighted factors w_x and w_y are set to 1. Therefore, the functions become: d_{path}= (|*path* (*x*, *lca*)| + |*path* (*y*, *lca*)|)/ 2*|*path*(*ALL*, L_1)| and d_{depth}= |*path* (*lca*, L_1)|/ |*path*(*ALL*, L_1)|. The distance between *x* and *y* occurs to be d_{path}= (2+1)/2*3 =0.5 and d_{depth}=2/3.

Percentage Distance Functions. According to this subcategory, the distance between two values *x* and *y*, where *y* is an ancestor of *x*, may be expressed according to a percentage of occurrences over the values of the hierarchy. In other words, the similarity of two values is expressed as the similarity of the number of descendants this two values have. Assume the lattice of level hierarchies be denoted as $L_1 \prec \ldots \prec L_L \prec L_x \prec L_y \prec All$ where L_1 denotes the most detailed level. The distance of a value *x* in a level L_x with regard to its ancestor *y* in level L_y may be calculated according to the function:

$$dist(x, y) = \frac{|\, desc_{L_i}^{L_x}(x) \,|}{|\, desc_{L_i}^{L_y}(y) \,|}$$, where L_i is one of L_x, L_L and L_1.

The above formula expresses the distance between a value *x* and one of its ancestors *y* as a percentage via three ways. In case L_i is L_x, then the distance is expressed as a percentage with regard to the occurrences of all the other values from L_x whose ancestor is *y*. In case L_i is L_L(or L_1), the distance is expressed as a percentage of occurrences of the descendants of *x* in a lower level of hierarchy L_L(or L_1) with regard to the descendants of *y* in the same lower level L_L(or L_1). As an example, assume the dimension *Location* where its lattice can be visualized in Figure 4.1 and the values of this dimension are visualized in Figure 4.2. Assume the values *x*='USA' and *y*='America'. Then, with regard to the above formula the distance between these two values can be computed as:

$$dist('\text{USA}','\text{America}') = \frac{1}{|\, desc_{Country}^{Continent}('\text{America}') \,|} = \frac{1}{2}$$ where L_i is chosen to be the level L_x, i.e., $L_{country}$

$$dist('\text{USA}','\text{America}') = \frac{|\, desc_{City}^{Country}('\text{USA}') \,|}{|\, desc_{City}^{Continent}('\text{America}') \,|} = \frac{3}{5}$$ where L_i is chosen to be the detailed level L_1, i.e., L_{city}

In this example the third case coincides with the second since the lower and detailed level, i.e. *City*, are identical.

Highway Distance Functions. Assume that every level of hierarchy L is grouped into k groups and every group has its own representative r_k. Then, the distance between two representatives can be thought of as a highway [SaSc05]. We denote with $r(x)$ and $r(y)$ the representatives of the groups where x and y belong to respectively. Therefore, the distance between the values x and y can be expressed with the following formula:

$dist\ (x, y) = dist\ (x, r(x)) + dist\ (r(x), r(y)) + dist\ (y, r(y))$

The partial distances between a value and its representative and the distance between the two representatives, $r(x)$ and $r(y)$, depends on the way the representative is selected. In most cases, the representatives are selected so that they belong to the same level of hierarchy and thus their distance can be computed from the locally computable functions, the path functions or the aggregated functions (in case the two representatives belong to different levels their distance may be computed by applying any distance function from the path section or the aggregated distance function section). The main categories of selecting the representative apart from an explicit assignment are with regard to (a) an ancestor and (b) a descendant. For the following, $dist(a, b)$ denotes the distance of any two values a, b. Without loss of generality assume $L_x \prec L_y$ (see Figure 4.3). In addition, assume the ancestor of x in level L_y is $x_y = anc_{L_x}^{L_y}(x)$ and a representative of y in the level of hierarchy L_x denoted as $y_x = f(desc_{L_x}^{L_y}(y))$. The function f applied over the descendants of y can result either to an explicitly assigned descendant or to the result of an aggregation function (e.g., *min*, *max*) over the set of descendants. In the following, we describe the partial distances of the previous formula depending on the way the representative is selected.

a) The representative of a group is an ancestor. The representative of each value x and y could be $r(x) = anc_{L_x}^{L_U}(x)$ and $r(y) = anc_{L_y}^{L_V}(y)$ where L_U and L_V is any upper level of L_x and L_y respectively. L_U and L_V are not obligatory different. In general, the distance

between a value x and its representative may be computed through any distance function from the path, the percentage or the aggregated functions. For example, assume two values x='UK' and y='USA' from the level *Country* of the hierarchy *Location* denoted in Figure 4.2. Assume the representative $r(x)$='Europe' and the representative $r(y)$='America'. The distance of the values x and y is by summing the distances *dist*('UK', 'Europe'), *dist*('Europe', 'America') and *dist*('America', 'USA'). In this category there are two special cases:

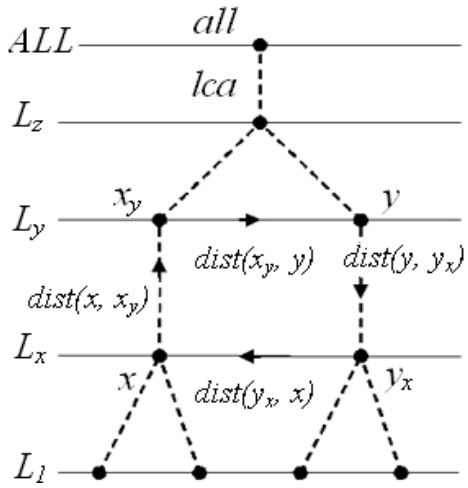


Figure 4.3. Partial Distances Between Two Values in Different Levels of Hierarchy.

The representatives $r(x)$ and $r(y)$ coincide in being the lowest common ancestor *lca*, where the formula is simplified as: $dist(x, y) = dist(x, lca) + dist(y, lca)$.

The representative $r(y)$ is identical to the actual value of y. In this case the distance is expressed as a summation of $dist(x, x_y)$ and $dist(x_y, y)$, as shown in Figure 4.3, where x_y is the representative of x from the level L_y. Therefore, the distance $dist(y, r(y)) = 0$. Formally this is expressed as: $dist(x, y) =$

$$dist(x, x_y) + dist(x_y, y) = dist(x, anc_{L_x}^{L_y}(x)) + dist(anc_{L_x}^{L_y}(x), y)$$

In case the representative x_y of x and y coincides, the distance is simplified as $dist(x, x) = dist(x, x_y)$. Since $dist(x, x_y)$ and $dist(x_y, y)$ are within the interval [0, 1], the normalized form of $dist(x, y)$ occurs by dividing it by 2. For example, assume two values x = 'USA' and y = 'Europe' from the dimension *Location* as seen in Figure 4.2.

The ancestor x_y of x is $anc_{Country}^{Continent}(x) = \text{'America'}$. Assume $dist(x, x_y)$ is computed from the percentage family functions. $dist(x_y, y)$ is computed through the first formula from the path family functions where the weighted factors w_x and w_y are set to 1. The distance between x and y becomes $dist$('USA', 'Europe')= ($dist(x, x_y)$ + $dist(x_y, y)$)/2 = ($dist$('USA', 'America') + $dist$('America', 'Europe'))/2 = (1/2 + 2/3)/2 = 7/12.

b) The representative of a group is a descendant. The representative of a group can be selected with respect to the descendants of the group where x belongs. For example, consider countries whose representatives can be selected among their cities, based for instance on the major airport or the highest population. In case the representative $r(x)$ is a value from the domain of L_L (i.e., $r(x)$ picked explicitly by applying a min or max aggregation over the set $desc_{L_x}^{L_L}(x)$), the distance between x and $r(x)$ can be any function from the families of path, percentage or aggregated functions. In case $r(x)$ is an arithmetic type value (i.e., a sum or count aggregation function over the set $desc_{L_x}^{L_L}(x)$), the distance between x and $r(x)$ can be any simple arithmetic function such as the Minkowski. There is a special case where the representative $r(x)$ is identical to the actual value of x. Thus, the distance is expressed as a summation of $dist(y, y_x)$ and $dist(y_x, x)$, where y_x is the representative of y from the level L_x as shown in Figure 4.3. Therefore, the distance $dist(x, r(x))=0$. Formally this is expressed as:

$$dist(x, y) = \frac{dist(y, y_x) + dist(y_x, x)}{2} = \frac{dist(y, f(desc_{L_x}^{L_y}(y))) + dist(f(desc_{L_x}^{L_y}(y)), x)}{2},$$

where the denominator is set to 2 for normalization reasons. For example, assume two values from the hierarchy *Location*, x='USA' and y='Europe', where the descendant of y is selected as $f(desc_{L_x}^{L_y}(y)) = \text{'UK'}$. Assume the distance between y and its descendant y_x is computed through the formula $dist(y_x, y) = \frac{|desc_{L_x}^{L_x}(y_x)|}{|desc_{L_x}^{L_y}(y)|}$ from the percentage family functions. The distance between x and y_x is computed through the first formula from the path family functions with w_x and w_y set to 1. Then, the distance between x and y becomes

$$dist(\text{'USA','Europe'}) = \frac{dist(y, y_x) + dist(y_x, x)}{2} = \frac{dist(\text{'Europe','UK'}) + dist(\text{'UK','USA'})}{2} = \frac{1/1 + 4/6}{2} = \frac{5}{6}.$$

In the special case where x is a descendant of y the above formula is simplified as: $dist(x,y) = dist(y, y_x)$.

4.1.2. Distance Functions between two Cells of Cubes

In this section, we describe the distance functions that can possibly be applied in order to measure the distance between two cells from a cube. Assume an OLAP cube C defined over the detailed schema C= [L_1^0, L_2^0, …, L_n^0, M_1^0, M_2^0, …,M_m^0], where L_i^0 is a detailed level and M_j^0 is a detailed measure. In addition, assume two cells from this cube, c_1 = (l_1^1, l_2^1, …, l_n^1, m_1^1, m_2^1, …, m_m^1) and c_2 = (l_1^2, l_2^2, …, l_n^2, m_1^2, m_2^2, …, m_m^2), where l_i^1, $l_i^2 \in dom(L_i^0)$ and m_j^1, m_j^2 denote the values of the corresponding measure M_j^0. The distance between two cells c_1 and c_2 can be expressed with regard to (a) their level coordinates $d_i(L_i^1, L_i^2)$ and (b) their measure values $d_j(M_j^1, M_j^2)$. In other words, $dist(c_1, c_2) = f(d_i(L_i^1, L_i^2), d_j(M_i^1, M_i^2))$. The function f can possibly be (a) a weighted sum, (b) Minkowski, (c) min or (d) proportion of common coordinates.

Distance functions between two Cells of a Cube Expressed as a Weighted Sum.

In this category the distance between two cells c_1, c_2 where c_1, $c_2 \in C$ can be expressed through the formula $\frac{\sum_{i=1}^{n} w_i d_i(l_i^1, l_i^2)}{\sum_{i=1}^{n} w_i} + \frac{\sum_{j=1}^{m} w'_j d_j(m_j^1, m_j^2)}{\sum_{j=1}^{m} w'_j}$, where w_i and w'_j are parameters that assign a weight for the level L_i and the measure M_j respectively, $d_i(l_i^1, l_i^2)$ denotes the partial distance between two values from dimension D_i and $d_j(m_j^1, m_j^2)$ denotes the partial distance between two instances of the measure M_j^0. Regarding the distance $d_i(l_i^1, l_i^2)$, this can be expressed through the various distance functions (Section 4.1.1) between two values from the same dimension. The distance $d_j(m_j^1, m_j^2)$ between two instances of a measure can be calculated through the Minkowski family distance when m_j^1, m_j^2 are of arithmetic

type, or through the simple identity function in case m_j^1, m_j^2 are of character type. The above formula is a general expression of the distance between two cells. Simplifications of this can be applied. For instance, the distance of two cells can be calculated only with respect to the coordinates that define each cell and without taking into consideration the measure values of each cell, i.e., by omitting from the above formula the second fraction. Moreover, in case the partial distances are normalized in the interval [0, 1] then, the distance between two cells is normalized in the same interval [0, 1]. For example, assume we want to compute the distance between cells c_1, c_2 as shown in Figure 4.4. Both cells consist of two dimensions (*Time, Location*), with the hierarchy levels of Figure 4.1, and contain one measure (*Sales*). In the above formula we set all the weight factors to 0.5 --both for dimensions (w) and measures (w'). The distance between dimensions is computed according to the function d_{path} that takes into account the length of the path of the hierarchy. The distance between the measures is computed through the normalized Manhattan distance function. In addition, assume that the overall maximum and minimum values of the measure sales are 10 and 1 respectively. Then, $d(c_1,c_2)=$

$$\frac{w * d(Month_{c_1}, Month_{c_2}) + w * d(Country_{c_1}, Country_{c_2})}{w + w} +$$

$$\frac{w' * d(Sales_{c_1}, Sales_{c_2})}{w'} = \frac{0.5*1/3 + 0.5*1/3}{0.5+0.5} + \frac{0.5*(|4-3|/|10-1|)}{0.5} = 4/9$$

To compute the distances $d(Month_{c_1}, Month_{c_2})$ and $d(Country_{c_1}, Country_{c_2})$ we refer the reader to Figure 4.5 and 4.6.

	Month	*Country*	*Sales*
C_1	*May/2000*	*USA*	*4*
C_2	*Apr/2000*	*canada*	*3*

Figure 4.4. Instances of Cells c_1 and c_2.

In Figure 4.5 we see that the length of the path between the nodes *a* and *lca* is 1, and the length of the path between the nodes *b* and *lca* is 1 again. According to the

function d_{path}, $d(Month_{c_1}, Month_{c_2}) = \frac{1+1}{6} = \frac{1}{3}$. In a similar manner, by using the information that derives from Figure 4.6 $d(Country_{c_1}, Country_{c_2}) = \frac{1+1}{6} = \frac{1}{3}$.

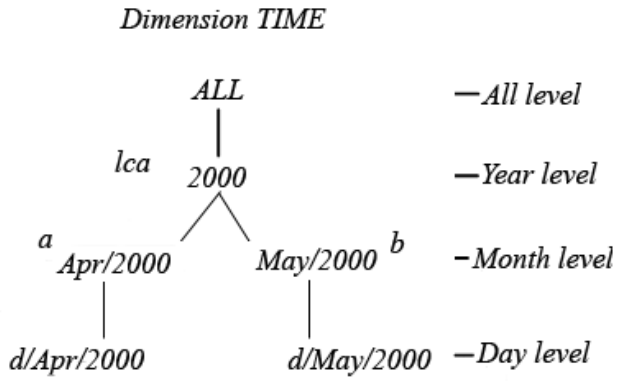


Figure 4.5. Lattice of the Dimension *TIME* for the Values of Cells of Figure 4.4.

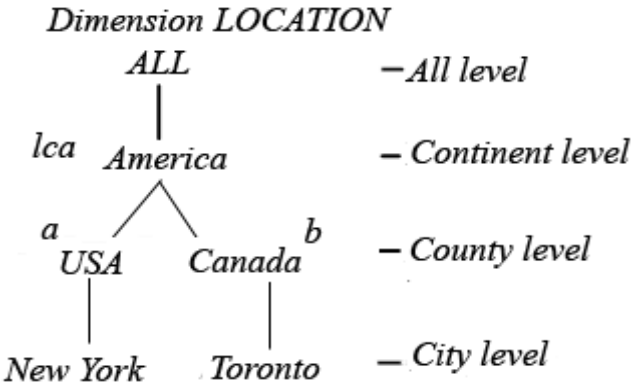


Figure 4.6. Lattice of the Dimension *LOCATION* for the Values of Cells of Figure 4.4.

Distance functions between two Cells of a Cube Expressed with regard to the Minkowski Family Distances.

In this section, we describe the possible distance functions between two cells of a cube by using the Minkowski family distances. In general, the Minkowski distance is defined via the formula $L_p[(x_1,...,x_n),(y_1,...,y_n)] = \sqrt[p]{\sum_{i=1}^{n} d_i(x_i, y_i)^p}$, where $d_i(x_i, y_i)$ denotes the distance between the two coordinates x_i and y_i of two given points x and y.

Assume two cells $c_1 = (l_1^1, l_2^1, \ldots, l_n^1, m_1^1, m_2^1, \ldots, m_m^1)$ and $c_2 = (l_1^2, l_2^2, \ldots, l_n^2, m_1^2, m_2^2, \ldots, m_m^2)$, where $l_i^1, l_i^2 \in dom(L_i)$ and m_j^1, m_j^2 denote the values of the corresponding measure M_j. The Minkowski distance can be applied in this category, by substituting point coordinates x_i and y_i with cell coordinates, thus l_i^1 and l_i^2. In general, in the Minkowski family distances the partial distances are defined as $d_i(x_i, y_i)=|x_i - y_i|$. When applying the Minkowski distance over cell coordinates, then the partial distances $d_i(l_i^1, l_i^2)$ can be expressed as the distance between two values from the same dimension (Section 4.1.1).

So far, the distance between two cells is described only with regard to their level coordinates. However, the distance between two cells can also be expressed by taking into consideration their measure values, too. The Minkowski family distances can be applied, as well, with regard to the partial distances $d_j(m_j^1, m_j^2)$. Therefore, the distance between two cells can be expressed by adding the equivalent two formulas. Depending on the value of p (1, 2, .., ∞) the Minkowski distance is defined as:

$$L_p = \sqrt[p]{\sum_{i=1}^{n} (d_i(l_i^1, l_i^2))^p} + \sqrt[p]{\sum_{j=1}^{m} (d_j(m_j^1, m_j^2))^p} .$$

Distance Functions between two Cells of a Cube Expressed as the Minimum Partial Distance.

In this category, the distance between two cells $c_1 = (l_1^1, l_2^1, \ldots, l_n^1, m_1^1, m_2^1, \ldots, m_m^1)$ and $c_2 = (l_1^2, l_2^2, \ldots, l_n^2, m_1^2, m_2^2, \ldots, m_m^2)$ can be expressed as:

$$\min_{d_i}\{d_i(l_i^1, l_i^2)\} + \min_{d_j}\{d_j(m_j^1, m_j^2)\} =$$

$$\min\{d_1(l_1^1, l_1^2), d_2(l_2^1, l_2^2), \ldots, d_n(l_n^1, l_n^2)\}$$

$$+ min\{d_1(m_1^1, m_1^2), d_2(m_2^1, m_2^2), \ldots, d_m(m_m^1, m_m^2)\}.$$

Therefore, the distance between two points is expressed as the minimum distance of their level coordinates plus the minimum distance of their measure values.

Distance Functions between two Cells of a Cube Expressed as a Proportion of Common Coordinates.

In this category the distance between two cells can be expressed as a proportion of their common values of their level coordinates and their measure values. Therefore, the distance between two cells $c_1 = (l_1^1, l_2^1, \ldots, l_n^1, m_1^1, m_2^1, \ldots, m_m^1)$ and $c_2 = (l_1^2, l_2^2, \ldots, l_n^2, m_1^2, m_2^2, \ldots, m_m^2)$ can be expressed through the formula *f*:

$$\frac{count(l_i^1 = l_i^2 \forall i \in \{1,2,...,n\})}{n} + \frac{count(m_j^1 = m_j^2 \forall j \in \{1,2,...,m\})}{m}$$

The above formula defines the distance between two cells as a summation of two fractions. The first fraction is the number of level values that are same for both cells, divided by the number of all level values that describe a cell. The second fraction expresses the number of measures that have the same value for both cells divided by the number of all possible measures in a cell.

4.1.3. Distance Functions between two OLAP Cubes

Assume two OLAP cubes *C* and C' defined over the same detailed schema $[L_1^0, L_2^0, \ldots, L_n^0, M_1^0, M_2^0, \ldots, M_m^0]$, where L_i^0 is a detailed level and M_j^0 is a detailed measure. In addition, assume that cube *C* consists of *l* cells of the form $c = (l_1, l_2, \ldots, l_n, m_1, m_2, \ldots, m_m)$ and cube *C*' consists of *k* cells of the form $c' = (l_1', l_2', \ldots, l_n', m_1', m_2', \ldots, m_m')$, where $l_i, l_i' \in dom(L_i^0)$ and m_j, m_j' denote the values of the corresponding measure M_j^0. In general, the two cubes can be of different cardinality, i.e., $l \neq k$. Assume $dist(c, c')$ where $c \in C$ and $c' \in C'$ denotes the distance between two specific cells according to the various categories of Section 4.1.2. The distance between the two cubes can be expressed as a synthesis of the partial distances $dist(c, c')$. In other words, $dist(C, C') = f(dist(c, c'))$ is a function of the partial distances $dist(c, c')$. The function *f* can possibly belong to one of the following families: (a) *closest relative*, (b) *Hausdorff* distance, (c) a *weighted sum*, (d) *Minkowski* distance, and (e) *Jaccard's coefficient*. For example, assume we want to compute the distance between the two cubes $CUBE_1$ and $CUBE_2$ as shown in Figure 4.7. $CUBE_1$ consists of three cells whereas $CUBE_2$ consists of 5 cells. Each cell in both cubes consists of two dimensions in different levels of hierarchy and the measure *Sales*. Specifically, each cell of $CUBE_1$ is of the form $c = (Day, City, Sales)$ and each cell of $CUBE_2$ is of the

form *c*' = (*Year*, *Country*, *Sales*). The distance between the two cubes can be expressed by applying a function *f* over the partial distances *dist*(*c*, *c*') of the cells of the two cubes.

4.2. Cell Mapping and Categories of Distance Functions according to it

The aforementioned function *f* can be computed either (i) over the full space of cell combinations of cells from the two cubes (families (a), (b) and (e)), or, (ii) over a specific subset of this space that is defined via a specific mapping of the cubes' cells (families (c) and (d)). In this section, we introduce the method that is used in order to map the cells of one cube to the cells of another cube. We refer to this method as *Cell Mapping*. For two cubes C_1 and C_2, the simple mapping of their cells includes the connection of every cell of the cube C_1 with one cell of the cube C_2. Intuitively, the mapping of a cell in cube C_1 tries to capture the discovery of the "closest possible representative" of this cell in cube C_2. The "closest representative" is the cell of the cube C_2 with the less distance among the dimension values with the cell of the cube C_1. In principle, the Cell Mapping method can be thought of as a relation that connects the cells of a cube to the cells of another cube (i.e., one can consider several candidate "representatives" of a cell). However, in our setting, this relation is reduced to a function, since we are interested in mapping each cell from the first cube to only one cell from the second cube. This is done for reasons of simplicity and allows the elegant definition of cube distances (see next). We impose the restriction that the function is total, i.e., each and every cell from the first cube is mapped to a cell of the second cube. We do not require that the mapping is 1:1 and onto; thus, in the second cube there might be a cell in which more than one cells from the first cube, or, no cells at all, are mapped to it.

As an example assume the cubes that are presented in the Figure 4.7. The cells c_1, c_2, c_3 of $CUBE_1$ are mapped to the cells c_7, c_5, c_5 of $CUBE_2$ respectively. Moreover, in the same figure the cells c_4, c_6, c_8 of $CUBE_2$ are not mapped with any cell of $CUBE_1$. We can also observe that the cell c_5 of $CUBE_2$ is mapped with two cells of $CUBE_1$.

The cell mapping method needs to compute the distances between the dimensions of each cell of the first cube with the dimensions of every cell of the second cube and ignores the distance between the measures. So, if the distance between two cells c_1, c_2 is expressed as $f\ (d_i(L_i^1, L_i^2), d_i(M_j^1, M_j^2))$, then the mapping method considers only the $d_i(L_i^1, L_i^2)$. Thus, each cell of the first cube is mapped to the cell of the second cube with the lowest $d_i(L_i^1, L_i^2)$ distance.

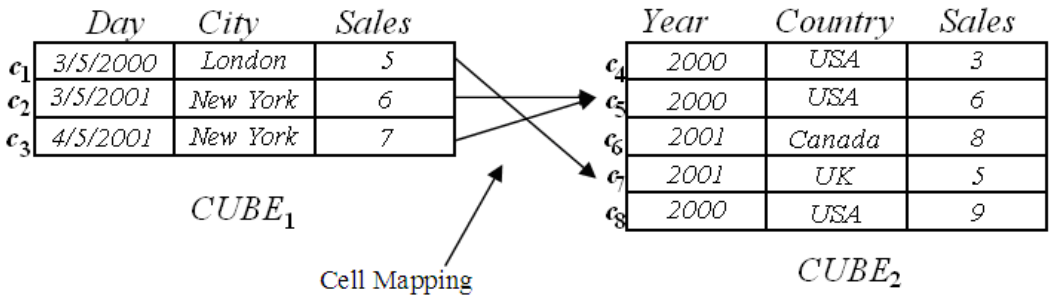


Figure 4.7. Instances of Two Cubes and the Mapping of their Cells.

In our taxonomy, two distance functions between cubes use the cell mapping method. These are (a) distance functions expressed with regard to the *Closest Relative* and (b) the distance function expressed by *Hausdorff* distance. After the mapping has been accomplished, the distances between the mapped cells are computed. Finally, the computation of the distance between the two cubes is performed on the basis of the distances among the mapped cells.

The distance functions that can be used in order to compute the distance between two OLAP cubes can be divided into two categories. The first category involves distance functions that include the cell mapping method. The second category contains distance functions that do not include the cell mapping method. Following, we describe each distance function and formally define it. The distance functions of the first category are the *Closest Relative* and the *Hausdorff Distance* (Section 4.1.3) that include the cell mapping method. Then, the category of families that do not consider the cell mapping method in their definition, include the *Weighted Sum* function, the *Minkowski family* of distance functions, the *Jaccard's Coefficient* and the *minimum of distances* function.

4.2.1. Distance Functions that Include Mappings

This subsection contains the description of the distance functions that involve the Cell Mapping method. These distance functions are the *Closest Relative* and the *Hausdorff* and are described as follows.

Distance Function between Two Cubes Expressed with regard to the Closest Relative. In this category the distance between two cubes *C* and *C*' is expressed as the summation of distances between every cell of a cube with the most similar cell of another cube through the formula:

$$\frac{\sum_{i=1}^{k}(dist(c_i, c'))}{k} \forall c' \mid dist_{dim}(c_i, c') = \min\{dist_{dim}(c_i, c')\}$$

where $dist_{\dim}$ denotes the distance of two cells excluding the distance of their measures. In the above formula, $\forall c' \mid dist_{dim}(c_i, c') = \min\{dist_{dim}(c_i, c')\}$ reveals the cell mapping. Each one of the *k* cells from cube *C* is mapped to the cell of the cube C' that has the minimum $dist_{\dim}$ from it.

As an example, we will detail the computation of the distance between the cubes $CUBE_1$ and $CUBE_2$ shown in Figure 4.7. The first step is to map the cells of the cube $CUBE_1$ to the appropriate cells of the cube $CUBE_2$. In order to simplify the example, the computational part of the cell mapping method is not described here, but the cell mapping is denoted in Figure 4.7 through arrows between the cells of the two cubes. The distance function used in this example for the purpose of computing the distance between the cells of the two cubes is the weighted sum. The weight that was used is 0.5, equal for both the dimensions and measures. In addition, the distance function used to measure the distance between the dimensions is the d_{path} function. The cells c_1, c_2, c_3, are mapped to the cells c_7, c_5, and c_5 respectively. According to this mapping, in order to compute the distance between the two cubes, the needed distances between cells are:

$$d(c_1, c_7) = \frac{0.5*1/6+0.5*1/6}{0.5+0.5} + \frac{0.5*(|5-5|/|10-1|)}{0.5} = 1/6$$

$$d(c_2, c_5) = \frac{0.5*1/6+0.5*1/6}{0.5+0.5} + \frac{0.5*(|6-6|/|10-1|)}{0.5} = 1/6$$

$$d(c_3, c_5)= \frac{0.5*1/6+0.5*1/6}{0.5+0.5}+\frac{0.5*(|6-7|/|10-1|)}{0.5}=5/18$$

For the above computations we refer the reader to Figures 4.5 and 4.6 where the hierarchies of the dimensions *TIME* and *LOCATION* are presented. With the above distances, we can now compute the full distance between the cubes $CUBE_1$ and $CUBE_2$ through the first formula of the *closest relative* family functions:

$$d(CUBE_1,CUBE_2)=\frac{d(c_1,c_7)+ d(c_2,c_5)+ d(c_3,c_5)}{3}=\frac{11}{54}$$

Distance functions between two cubes expressed by Hausdorff distance. In this category, the distance between two cubes can be expressed by using the *Hausdorff* distance [HuKR93]. The Hausdorff distance between two cubes can be defined as $H(C,C') = max(h(C,C'), h(C',C))$ where $h(C,C') = \max_{c\in C}\{\min_{c'\in C'}\{dist(c,c')\}\}$ and *dist* (*c*, *c'*) is the distance between two cells *c* and *c'* from the cubes *C* and *C'* respectively. Function *h*(*C*, *C'*) is called the *directed* Hausdorff distance from *C* to *C'* and the distance measured is the maximum distance of a cube *C* to the *"nearest"* cell of the other cube *C'*. The Hausdorff distance is the maximum of *h*(*C*, *C'*) and *h*(*C'*, *C*).

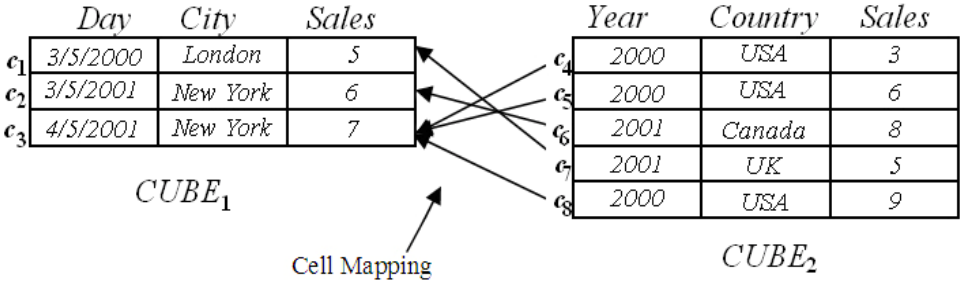


Figure 4.8. Instances of Cubes $CUBE_1$ and $CUBE_2$ and the Mapping of the Cells of the Cube $CUBE_2$ to the Cells of the Cube $CUBE_1$.

In the Hausdorff distance function, the cell mapping method is bidirectional. That means that except from the mapping that we have examined in the closest relative function, we also need the extra mapping from the cells of cube *C'* to the cells of cube *C*.

When the bidirectional mapping is completed, we obtain two sets of mapped cells. In each set, for every pair of mapped cells, we compute their distance considering their measures as well. Thus, we have two sets of minimum distances between cells, the set of minimum distances from the cells of cube *C* to the cells of cube *C*' and the set of minimum distances between from the cells of cube *C*' to the cells of cube *C*. From each of the two sets we pick the greatest distance and finally from these two distances we pick the greater one.

To make things more clear, an example follows. Assume again cubes $CUBE_1$ and $CUBE_2$ as shown in Figure 4.8. In Figure 4.8, we can observe the mapping from the cells of $CUBE_2$ to the cells of $CUBE_1$. According to this bidirectional mapping the two resulting sets of minimum distances are:

$S_1\{ d(c_1,c_7), d(c_2,c_5), d(c_3,c_5)\}$

$S_2\{ d(c_4,c_3), d(c_5,c_3), d(c_6,c_2), d(c_7,c_1), d(c_8,c_3)\}$.

The distances of the set S_1 are already computed on a previous example, so here we only need to compute the distances of set S_2. The distances $d(c_5,c_3)$, $d(c_7,c_1)$ coincide with the distances $d(c_3,c_5)$, $d(c_1,c_7)$ respectively. The computations below use the same distance functions between values and cells and also the same weight factors, as in the previous example.

$$d(c_4, c_3)= \frac{0.5*1/6+0.5*1/6}{0.5+0.5}+\frac{0.5*(|3-7|/|10-1|)}{0.5}=\frac{11}{18}$$

$$d(c_6, c_2)= \frac{0.5*1/6+0.5*3/6}{0.5+0.5}+\frac{0.5*(|8-6|/|10-1|)}{0.5}=\frac{10}{18}$$

$$d(c_8, c_3)= \frac{0.5*1/6+0.5*1/6}{0.5+0.5}+\frac{0.5*(|9-7|/|10-1|)}{0.5}=\frac{7}{18}$$

Now, the Hausdorff distance between the cubes $CUBE_1$ and $CUBE_2$ is equal to the next formula:

$d(CUBE_1,CUBE_2)=max\{max\{S_1\},max\{S_2\}\}=$

$max\{max\{1/6,1/6,5/18\}, max\{11/18,5/18, 1/6,10/18,7/18\}\}=$

$max\{5/18,11/18\}=11/18$.

4.2.2. Distance Functions that do not Include Mappings

This subsection includes the distance functions that do not include mappings. These functions are the *Weighted Sum* function, the Minkowski family of distance functions, the *Jaccard's Coefficient* and the *minimum of distances* function. The *Weighted Sum* function is expressed through the formula: $\frac{\sum_{i=1}^{l}\sum_{j=1}^{k} w_{ij} dist(c_i, c'_j)}{\sum_{i=1}^{l}\sum_{j=1}^{k} w_{ij}}$, where $dist(c_i, c'_j)$ is the distance between a cell from cube *C* to a cell from cube *C*' and w_{ij} denotes the weight factors assigned to each distance.

The distance functions of the *Minkowski family* --depending on the values of the parameter *p* (1, 2, ..., ∞)-- can be expressed as: $L_p = \sqrt[p]{\sum_{i=1}^{l}\sum_{j=1}^{k} dist(c_i, c'_j)^p}$, where $dist(c_i, c'_j)$ is the distance between a cell from cube *C* to a cell from cube *C*'.

The distance between two cubes can be expressed with regard to the *Jaccard's coefficient* [ZADB06]. The Jaccard's coefficient is defined as: $dist(C, C') = 1 - \frac{|C \cap C'|}{|C \cup C'|}$ and it expresses the ratio between the cardinalities of intersection and union of the cubes *C* and *C*'.

The *Minimum of distances* function expresses the distance between two cubes as the minimum distance among all possible distances between the cells of the compared cubes. Therefore, the distance between *C* and *C*' is expressed as: $dist(C, C') = min\{ dist(c, c') \mid c \in C, c' \in C' \}$, where $dist(c, c')$ is the distance between a cell from cube *C* to a cell from cube *C*'. In case the two cubes are disjoint i.e., $C \cap C' = \emptyset$, then $dist(C, C')$ is a positive number, whereas if the two cubes have common cells i.e., $C \cap C' \neq \emptyset$, then $dist(C, C')$ is zero.

As a simple example, assume the two cubes from Figure 4.7. and ignore the arrows that denote the cell mapping. According to the *minimum of distances* function, the distance between the two cubes is computed through the following formula where *j* denotes any cell from $CUBE_2$: $d(CUBE_1, CUBE_2) =$

$$\min_j \{ d(c_1, c_j), d(c_2, c_j), d(c_3, c_j) \}, \forall j \in \{4,5,...,8\} = 1/6.$$

4.3. Experiments

4.3.1. User Study for Distances between two Values of Dimensions

In this section, we describe a user study we conducted for discovering which distance functions between two values of a dimension seem to be more suitable for user needs. The experiment involved 15 users out of which 10 are graduate students in Computer Science and 5 that are of other backgrounds. In the rest of the section we refer to the set of users with computer science background as *Users_cs*, the set of users with other background as *Users_non* and the set of all users independently of their background as *Users_all*.

In the experiments we used the "Adult" real data set according to the dimension hierarchies as described in [FuWY05]. This dataset contains the fact table *Adult* and 8 dimension tables which are described in Table 4.1.

The purpose of the experiment is to assess which distance function between two values is best with regard to the user preferences. Each user was given 14 case scenarios. Each scenario contained a reference cube and a set of cubes, which we call *variant* cubes, that occurred by slightly altering the reference cube. The 14 scenarios included different kinds of cubes with regard to the value types and the different levels of granularity. For each reference cube which was randomly selected, the variant cubes were generated from the fact table by altering the granularity level for one dimension, or by altering the value range of the reference cube. For instance, assume a reference cube containing the dimension levels Age_level_1, $Education\_level_2$ under the age interval [17, 21]. According to the first type of modification, a variant cube could be generated by changing the dimension level to Age_level_2 or Age_level_0, or changing the level of the Education Dimension. According to the second type of modification, another variant cube could be generated by changing the age interval to [22, 26] or to [17, 26]. Among all possible variations of the reference cube we

manually chose the set of variant cubes such that each of them was most similar to the reference cube according to a distance function. In order to observe which distance function is preferred by users depending on the type of data of the cubes, we have organized the 14 scenarios into 3 sets. The first set consists of cubes containing only arithmetic type values (5 scenarios). The second set consists of cubes containing only categorical type values (2 scenarios). The third set consists of cubes containing a combination of both categorical and arithmetic type values (7 scenarios). All the scenarios used for this user study can be found in [Baik11].

Table 4.1. Adult Dataset Tables.

	Value Type	**Tuples**	**Dim. Levels**
Adult fact Table		30418	-
Age Dim.	*Numeric*	72	5
Education Dim.	*Categorical*	16	5
Gender Dim.	*Categorical*	2	2
Marital Status Dim.	*Categorical*	7	4
Native Country Dim.	*Categorical*	41	4
Occupation Dim.	*Categorical*	14	3
Race Dim.	*Categorical*	5	3
Work Class Dim.	*Categorical*	7	4

In each scenario, the users were asked to select the variant cube that seemed more similar to the reference cube based on their personal criteria. The distance functions that have been used in the experiment are shown in Table 4.2, where the first column shows the family in which each distance function belongs to according to Section 4.1.1. In the second column there is an abbreviated name for each function. To compute the distance between two cubes, the *Closest Relative* distance function is used (Section 4.1.3). The distance between two cells of cubes is the weighted sum of the partial distances of the two values, one from each cell, with all weights set to 1 (Section 4.1.2).

Table 4.2. Notation of Distance Functions Used in the Experiment.

Family	**Abbr.**	**Distance function name**
Local	δ_{M}	Manhattan
Aggregation	$\delta_{Low,c}$	With respect to a lower level of hierarchy where f_{aggr} =count
	$\delta_{Low,m}$	With respect to a lower level of hierarchy where f_{aggr} = max
Hierarchical Path	$\delta_{LCA,P}$	Lowest common ancestor through d_{path}
	$\delta_{LCA,D}$	Lowest common ancestor through d_{depth}
Percentage	$\delta_{\%}$	Applying percentage function
Highway	δ_{Anc}	With respect to an ancestor x_y
	δ_{Desc}	With respect to a descendant y_x
	$\delta_{H,Desc}$	Highway, selecting the representative from a descendant
	$\delta_{H,Anc}$	Highway, selecting the representative from an ancestor

The analysis of the collected data provides several findings. The first finding concerns the *top three most preferred distance functions* measured over the detailed data for all scenarios and all users. It is remarkable that the top three distance functions for each of the user groups were the same and with the same ordering and specifically, these are the $\delta_{LCA,P}$, the δ_{Anc} and the $\delta_{H,Desc}$. The frequencies for each one of the top three distance functions in each group of users is shown in Table 4.3.

Table 4.3. Top Three Most Frequent Distance Functions for Each User Group.

	Users_all	*Users_cs*	*Users_non*
$\delta_{LCA,P}$	40.47%	38.57%	44.28%
δ_{Anc}	18.09%	20.00%	14.28%
$\delta_{H,Desc}$	9.52%	10.71%	7.14%

The second finding concerns *the most preferred function by users depending on the type of data the cubes contained*. Table 4.4 summarizes the result of the most frequent distance function for each set of scenarios and each set of users. We observe that for the *categorical* type of cubes, all user groups prefer the $\delta_{LCA,P}$ distance function, whereas for the *arithmetic* and the *arithmetic & categorical* sets, the functions that

users mainly prefer are the $\delta_{LCA,P}$ and δ_{Anc}. More than one distance functions appear as winners in Table 4.4 due to ties in the frequency of occurrences for each function.

The third finding concerns the *winner distance function per scenario*. For every scenario, we take into account the 15 occurrences by all users and see which distance function is the most frequent. We call this function the winner function of the scenario. The most frequent winner function was $\delta_{LCA,P}$ with a 35.71% percentage for both the *Users_all* and the *Users_cs* group (5 of the 14 scenarios), and 57.14% for the *Users_non* group (8 of the 14 scenarios). The most frequent function for 14 of the 15 users was the $\delta_{LCA,P}$ function. For one user from the *Users_cs* group the most frequent function was the $\delta_{LCA,D}$.

Table 4.4 The Most Frequent Distance Function for Each Set of Scenarios.

	Users_all	*Users_cs*	*Users_non*
Arithmetic	δ_{Anc}	$\delta_{LCA,P}$, $\delta_{H,Desc}$, δ_{Anc}	$\delta_{LCA,P}$
Categorical	$\delta_{LCA,P}$	$\delta_{LCA,P}$	$\delta_{LCA,P}$
Arithmetic & Categorical	δ_{Anc}	δ_{Anc}	$\delta_{LCA,P}$, δ_{Anc}

The fourth finding concerns the *diversity and spread* of user choices. There are two major findings: (a) All functions were picked by some user, and, (b) there are certain functions that appeared as user choices for all users of a user group. Specifically, functions $\delta_{LCA,P}$, $\delta_{H,Desc}$ and δ_{Anc} were selected at least once by users of group *Users_cs*. Similarly, functions $\delta_{LCA,P}$, $\delta_{Low,m}$ and δ_{Anc} were selected at least once by *Users_non*.

The fifth finding concerns the *most preferred family of functions*. Table 4.5 depicts the absolute number of appearances of each distance function family per user group. The most preferred family of distance functions is the *Hierarchy Path* family, which also contains the top one most preferred distance function $\delta_{LCA,P}$. Moreover, we observe that the ranking of the distance function families was exactly the same for each user group.

Table 4.5. Frequencies of Preferred Distances within Each User Group for Each Distance Family.

	Local	*Aggregation*	*Hierarchy Path*	*Percentage*	*High-way*
Users_cs	1	9	69	9	52
Users_non	2	5	34	5	24
Users_all	3	14	103	14	76

The *selection stability* of users (i.e., discrepancies in users' answers at the same questions) was the sixth issue. The *selection stability* was determined by setting the 13th and the 14th scenario to be replicas of the 3rd and 10th scenario respectively. 4 out of 5 users from the set of *Users_non*, 6 out of 10 users from the set of *Users_cs* (consequently, 10 users from *Users_all* set) selected the same function for both of the two similar scenarios. The rest of the users selected the same function for only one out of the two repeated scenarios.

Summary. Overall, the findings indicate that the most preferred distance function is the $\delta_{LCA,P}$, which is expressed with respect to the shortest path of a hierarchy dimension. A null hypothesis stating that the fact that 40.47% of the times $\delta_{LCA,P}$ was chosen as a winner is due to a random phenomenon, has a *p*-value of 6.6×10^{-5}. Apart from the $\delta_{LCA,P}$, the distance functions δ_{Anc} and $\delta_{H,Desc}$ were also popular with the users. In addition, the most preferred distance function family is the *Hierarchy Path* family.

4.3.2. User Study for Distances between two Cubes

In the previous user study, the overall observation was that the users prefer the $\delta_{LCA,P}$ distance function between two values of the same dimension. Based on this result, and also by setting the *weighted sum* function as the distance function between cells, we set up the second user study in order to examine which distance function between two cubes is preferred by the users. Specifically, we try to find out which distance function among the two functions that include the *cell mapping* method (Section 4.1.3) is most closely related to the human perception. These two distance functions

are namely the *closest relative* and the *Hausdorff* distance function. Table 4.6 shows the distance functions that were used in this user study.

The user study contained 14 new scenarios. Each scenario included 4 cubes named *A*, *B*, *C* and *D*. The cube *A* in every scenario was the reference cube. The users were asked to order the rest of the three cubes from the most similar to the less similar when compared to the cube *A*. The cubes *B*, *C* and *D* were chosen such that one of them was the closest to the cube *A* according to the *closest relative* function and another was the closest to cube *A* according to the *Hausdorff* distance function. The remaining cube was chosen to be the most distant from cube *A* for both distance functions. All the scenarios used for this user study can be found in [Rogk10] and [Baik11].

All scenarios were uploaded as jpeg pictures in an html page where users were asked to complete an answer sheet and send it back to us via email. The URL of this page was sent to the email-list of the graduate students of the Computer Science Department of the University of Ioannina.

Table 4.6. The Distance Functions Used in the Second User Study.

between two cubes	*Hausdorff*
	Closest relative
between two cells of cubes	*weighted sum*
between two values of a dimension	$\delta_{LCA,P}$
between two measures	*Manhattan*

In order to test a user's answer reliability, in the 6th scenario, the cube *B* was identical with the cube *A*. Moreover, in order to measure the users' stability, the 13th and 14th scenarios were replicas of the 5th and 9th scenarios respectively with a reordering on the columns of the cubes.

Table 4.7. Frequency of Chosen as First Distance Function Among All the Answers.

	Frequency	Percentage
Hausdorff	154	38%
Closest relative	232	57%
Most distant cube	21	5%

The 12 first scenarios can be divided into three groups according to the weights in the distance function between cells. The first 4 scenarios consist of cubes that do not include measures. We refer to this group as the *no_measures* group. The next 4 scenarios consist of cubes that include measures where the weight factors on measures and dimensions in the function *between cells* are not equal. Specifically, assuming that cubes consist of k dimensions and l measures, the weight factors were set to $k/(l+k)$ for the dimensions and $l/(l+k)$ for the measures. We refer to this group as the *not_equal* group. Finally, the last four scenarios consist of cubes that include measures and the weight factors on the measures and on the dimensions in the *between cells* distance function are equal and set to 0.5. We refer to this group as the *equal* group.

Table 4.8 User Stability.

	User_OK		*User_Half_OK*		*User_Stable*	
scenario	Freq.	Perc.	Freq.	Perc.	Freq.	Perc.
13^{th}	28	75%	5	13%	24	65%
14^{th}	19	51%	8	21%	24	65%

The number of users that responded with an answer sheet was 39. Two of the 39 users did not choose the cube *B* in the sixth scenario as the most similar to the cube *A*. For that reason their answers were not taken into consideration. We refer to the remaining 37 users as *valid_users*.

The first finding of this user study concerns the most *frequent distance function* that was chosen from the users as their first choice. Among all the 11 (scenarios) * 37 (users) = 407 answers (the sixth scenario is excluded), 232 times ($\approx$57%) the users

gave as their first choice the cube that represents the *closest relative* distance function. The cube that represents the *Hausdorff* distance function was chosen 154 times (≈38%) as the first choice of the users. Only 21 times (≈5%) the users chose the most distant cube as their first choice. The summarization of the above results is shown in the Table 4.7.

Table 4.9 The Winning Functions and the Winner Functions.

Scenario Group	**Scenario**	**Winner function per scenario**		**Group Winner**
no_measures	Scen.1	*Closest relative*	29/37	*Closest relative*
	Scen.2	*Closest relative*	30/37	
	Scen.3	*Closest relative*	31/37	
	Scen.4	*Hausdorff*	25/37	
not_equal	Scen.5	*Hausdorff*	28/37	*Hausdorff*
	Scen.7	*Closest relative*	26/37	
	Scen.8	*Hausdorff*	27/37	
equal	Scen.9	*Hausdorff*	19/37	-
	Scen.10	*Hausdorff*	21/37	
	Scen.11	*Closest relative*	32/37	
	Scen.12	*Closest relative*	22/37	

The second finding of the user study concerns the stability of the user choices. As we mentioned before, the 13th and 14th scenario were replicas of the 5th and 9th scenario respectively. In each of these two scenarios a user that orders the cubes in the same way as in the original scenario is denoted as *user_OK*. A user that gave the same answer for the most similar cube but the order of the other cubes was not the same is denoted as *user_Half_OK*. Finally, a user that was denoted as *user_OK* for both replicas scenarios or denoted as *user_OK* for the one replica scenario and *user_Half_OK* for the other replica scenario is denoted as *user_ Stable*. According to the answers of the valid 37 users of this user study, in the 13th scenario there were 28 *user_OK* users and 5 *user_Half_OK* users. In the 14th scenario there were 19 *user_OK* users and 8 *user_Half_OK* users. The 24 of the 37 (≈65%) users were *user_Stable*

users. We believe that a 65% is a safe number that can ensure the stability and reliability of their answers. The Table 4.8 summarizes the above results and percentages.

The third observation concerns the *scenario winner function*. The term *scenario winner function* refers to the function that was mostly selected as the first choice from the users in a specific scenario. Our findings cannot ensure that one of the two functions is more preferred than the other: The *closest relative* function was the scenario winner function for 6 scenarios and the *Hausdorff* function was the scenario winner function for the rest 5 scenarios (Table 4.9). Observe that the findings of Table 4.7 give a 19% difference between the two prevailing functions --a finding that is not demonstrated in Table 4.9. This is explained by the fact that when the *closest relative* function is a winner, it wins with an overwhelming majority; on the contrary, when the *Hausdorff* function is a winner, the numbers are lower. The 4th column in Table 4.9 shows how many times the winner function was chosen as a first choice among the 37 valid users.

The fourth observation concerns the (*scenario*) *group winner function* (Table 4.9). For a group of scenarios, its *group winner* is the function that appeared as *scenario winner* in the majority of the scenarios of the group. For the *no_measures* group the *group winner function* was the *closest relative* function, as it was the *winner function* for the 3 out of the 4 scenarios. For the *not_equal* group the *group winner function* was the Hausdorff, as it was the *winner function* for the 2 out of the 3 scenarios. Finally, for the group *equal*, we have a draw: in two scenarios the winner function was the *closest relative* function and in two scenarios the winner function was the *Hausdorff* function. The above results reveal a user preference in the *closest relative* function for scenarios that do not include measures. On the other hand for the other types of scenarios the results are not clear.

4.3.3. Reliability and Validity Considerations

Test Reliability. A possible threat to the test's reliability is the inability of users to understand what was asked from them to perform, or did not handle the test with

seriousness and mental concentration. In the 1st user study, the users took the experiment in our presence so we can ensure there were no ambiguous situations or possible misunderstandings. In the 2nd user study, users completed the questionnaire via the web. However, there was a clear description of the setting of the experiment along with an example, so we believe there were not any misunderstandings of what the users should answer. Moreover, we excluded users that failed giving the straightforward answer (in scenario 6 of the 2nd experiment). Finally, in both user studies, we tested the stability of users via replica scenarios.

Test Validity. Possible threats to tests' external validity are the size and the mix of the corpus of users. Naturally, the size of users can always be increased; however we deem that the corpuses we have used are not negligible. Concerning the mix of users, in the 1st experiment we choose to include a group of users with a diversity of backgrounds as well as a clearly distinct group of users with background of computer science (and thus, higher affinity to the notion of comparing two data cubes). An interesting observation is the fact that there are differences of opinions between the *Users_cs* and *Users_non* (Table 4.3 and Table 4.5), however these are small and do not change the overall ranking of the preferred functions. Thus, we were able to proceed to a web-based questionnaire in the 2nd study. In addition, the possible scenarios were selected in a way that includes a variety of data types (arithmetic, categorical) and various levels of granularity over the data.

4.4. Chapter Summary and Findings

This Chapter presented a variety of distance functions that can be used in order to compute the similarity between two OLAP cubes. The functions were described with respect to the properties of the dimension hierarchies and based on these they were grouped into functions that can be applied (a) between two values from a dimension of a multidimensional space, (b) between two points of a multidimensional space and (c) between two sets of points of a multidimensional space.

In order to assess which distance functions are more close to human perception, we conducted two user study analysis. The first user study analysis was conducted in

order to discover, which distance function between two values of a dimension is best with regard to the user needs. Our findings indicate that the distance function $\delta_{LCA,P}$, which is expressed as the length of the path between two values and their common ancestor in the dimension's hierarchy was the most preferred by users in our experiments. Two more functions were widely chosen by users. These were the highway functions δ_{Anc} that is expressed with regard to the ancestor x_y and $\delta_{H,Desc}$ that is expressed by selecting the representative from a descendant.

The second user study we conducted, took into account the results of the first user study analysis. Specifically, the second user study analysis aimed in discovering which distance function (the *closest relative* or the *Hausdorff* distance function) from the category of distance function between two data cubes, users prefer. Overall, the former function was preferred by the users than the latter; however the individual scores of the tests indicate that this advantage is rather narrow.

CHAPTER 5. CONCLUSIONS

The goal of this thesis was to explore and investigate the answering of top-*k* queries through the exploitation of materialized top-*k* views. Apart from answering top-*k* queries through materialized views, we have also studied the problem of maintaining top-*k* materialized views in the presence of updates in the relation such that the views can be up to date and useful for the answering of top-*k* queries. Moreover, we explored the problem of expressing the similarity between two data collections. In order to express similarity between objects we have worked on discovering the distance functions that users prefer for computing the similarity of two data collections. To this end, we resorted to the simplest framework that can be given to users to work with and that has been OLAP Cubes.

5.1. Summary of Contributions

In this section we summarize the main research challenges and findings of this thesis.

Answering top-*k* Queries via Materialized Views

We have provided theoretical and algorithmic results for the answering of top-*k* queries through the usage of materialized top-*k* views. By adopting a geometric representation of the top-*k* query problem we have conducted a theoretical analysis for providing theoretical guarantees for the suitability of a materialized view in order to answer a top-*k* query. Specifically, we illustrated this through the notion of *safe area* of a query in regards to a view and provided the suitability theorem. Moreover, we

have proved that the theorem is strict in the sense that it cannot be inversed. Thus, we have proved that even if the safe area is not eligible for answering a top–*k* query, still the view may be suitable for answering a query and we have described this through the notion of the *critical area*. In addition, according to the theoretical establishments we have provided two algorithms for the answering of top-*k* queries through the usage of materialized views without accessing the tuples of the relation. We have provided the 2D SafArI Algorithm for the 2D case, and the SafArI Algoroithm for the *n*-D case. Furthermore, we have theoretically proved that the safe areas of a query in regards to more than one views do not offer further usefulness for answering the query compared to the safe area of a single view. We have also discussed the issue of providing partial results for a query via a materialized view by splitting the range of scores into appropriate sub-ranges and provided the Compute Query Extent Algorithm. We have proved the efficiency and effectiveness of our method through an extensive set of experiments. The experiments that concerned the 2D SafArI Algorithm, revealed that the effectiveness of the method has been rather stable and around 30-35%. The efficiency of our method showed a consistent increase for reasonable sizes of *k* that rose up to 24%. The second set of experiments concerned the *N*-D case. The effectiveness as well as the efficiency of our method revealed that for random and anticorrelated datasets there was an influence on the results in regards to the dimensionality. However, for the correlated datasets the effectiveness was unaffected by dimensionality almost 100%. The real dataset experiments revealed and effectiveness above 35% in all scenarios and increased significantly when the number of materialized views increased.

Maintaining Materialized top-k Views

Considering the problem of maintaining top-*k* materialized views, we have provided results in two directions. As for the first direction we have provided a principled method that complements the inefficiency of the state of the art independently of the statistical properties of the data and the characteristics of the update streams for the maintenance of materialized views. Specifically, the method we have provided consists of three steps: (a) computes the rate that actually affects the materialized view, (b) computes the necessary extension to *k* in order to handle the augmented number of deletions that occur, and (c) fine tunes by adjusting this value to take the

fluctuation of the statistical properties of this value into consideration. The second direction concerned the case of multiple top-*k* views and their efficient maintenance in the presence of updates to their base relation. We have provided theoretical guarantees for the establishment of the effect of updates to a certain view, whenever we know that another view has been updated. We have also provided algorithmic results towards the maintenance of a large number of views, via their appropriate structuring in a hierarchy of views. Our experiments have shown that our method accurately sustains intervals with high deletion activity in the workload and specifically in at least 95% of the cases there were top-*k* materialized views that contained at least *k* items. The experiments indicate that our method outperforms the state-of-the-art [YYY+03] in terms of efficiency as the computation of the exact number of auxiliary view tuples has shown to be faster than the computation of refill queries as proposed in the related literature. At the same time, the number of auxiliary view tuples has been less than the number proposed in [YYY+03]. Moreover, the fine tuning method we proposed, gave zero losses.

Similarity Measures for Multidimensional Data

The contribution towards the problem of discovering the distance functions for computing the similarity of two data collections, according to what real users actually think was again into two directions. We firstly presented a variety of distance functions that can be used in order to compute the similarity between two OLAP cubes and were described with respect to the properties of the dimension hierarchies. Thus, they were grouped into functions that can be applied (a) between two values from a dimension of a multidimensional space, (b) between two points of a multidimensional space and (c) between two sets of points of a multidimensional space. Following, we assessed which distance functions are more close to human perception, where we have conducted two user study analysis. The first user study analysis was conducted in order to discover, which distance function between two values of a dimension is best with regard to the user needs. Our findings indicated that the distance function $\delta_{\text{LCA,P}}$, which is expressed as the length of the path between two values and their common ancestor in the dimension's hierarchy was the most preferred by users in our experiments. Two more functions were widely chosen by users. These were the highway functions δ_{Anc} that is expressed with regard to the

ancestor x_y and $\delta_{H,Desc}$ that is expressed by selecting the representative from a descendant. The second user study we conducted, took into account the results of the first user study analysis. Specifically, the second user study analysis aimed in discovering which distance function (the *closest relative* or the *Hausdorff* distance function) from the category of distance function between two data cubes, users prefer. Overall, the former function was preferred by the users than the latter; however the individual scores of the tests indicate that this advantage is rather narrow.

5.2. Open Problems and Insights for Future Work

In this section we provide directions for future research on issues that are still open and can be based on the results of this thesis.

5.2.1. View selection and caching

The problem of answering top-*k* queries through the usage of materialized ranking views raises the problem of selecting the appropriate views in order to process efficiently and effectively the posed queries. The view selection problem has been addressed by both PREFER and LPTA algorithms. However, these works either assume that the materialized views contain *all* tuples of the underlying relation ranked according to the view's scoring function, or, they select the most suitable ranked view based on an estimation of the score of the last tuple of the top-*k* query. Thus, in the second case there is no theoretically established guarantee that the selected views will be able to answer the query. In any case, the estimation of the last tuple in the query might lead to selecting a view that is not the most appropriate either in the sense that it cannot provide an answer to the query or in the sense that is not the most efficient one. Given, the theoretical established guarantees we have proved, it would be interesting to study the problem of selecting the appropriate materialized view in order to answer the top-*k* query in terms of efficiency. Thus, by adopting a cost formula for each materialized view that safely guarantees the answer to the top-*k* query, it could be possible to select the most appropriate view for answering the query. The cost formula can express the cost of the usage of a given materialized view

in order to provide the answer to the top-*k* query through the number of tuples that should be fetched, or, as the area of the materialized view in terms of surface units.

5.2.2. View caching

Similar to the view selection problem, another open issue involves the view caching problem. In particular, in the context of distributed settings, where each underlying server contains some local data, it is interesting to decide appropriately which materialized views would be cached and which servers contain which cached results. In general, the view caching problem is closely related to the view selection problem since the overall idea is to identify the most promising set of views for the upcoming queries. In other words, the caching problem is addressed as selecting the most useful views in terms of the ability to provide an answer for a top-*k* query as well as efficiency in the presence of resource constraints. On one hand, a view should be contained in the set of cached results if it is likely enough to provide an answer for most of the top-*k* queries. This could be achieved by caching a set of materialized views that capture most of the space of the relation, so that there would always be a materialized view that could provide the answer to any possible top-*k* query. However, another idea would be to cache those materialized views that are most likely to be used for the majority of the top-*k* queries leaving out the outlier top-*k* queries. In order to decide the most appropriate set of materialized views, the above two ideas should be taken into consideration and balanced in a way that the best combination would provide the less cost for the answering of the new top-*k* queries. Similarly to the view selection problem, a cost formula that expresses the cost of providing the answer of a top-*k* query from a specific materialized view should be constructed. In addition, since the views are materialized, the cost formula should also contain in its expression the cost of maintaining a view in the presence of updates. This cost formula would help in eliminating from the cached views those that provide the answer to top-*k* queries with high costs when compared to all the rest views.

5.2.3. Combining indexing techniques with materialized views for query processing of top-k queries in multi dimensional space

The usage of materialized views as well as indexing techniques has been used in query processing mainly in terms of performance. Materialized views are used in order to provide an answer to a query that is pre-computed. Indices could prove helpful when they index the views and thus the later are selected and obtained faster. An initial attempt of indexing materialized views for the answering of top-*k* queries has been proposed by Tsaparas et. al. in Ranked Joined Indices. However, *RJI* solve the problem only for the 2 dimensional case. It would be interesting to see how an index for materialized views could be constructed and proved helpful for the answering of top-*k* queries in multidimensional space. Two main characteristics of the materialized ranked views play significant role in the answering of a top-*k* query. The first is the depth of the view, i.e., the number of tuples that are materialized in the view. The second is the closeness of the view to the top-*k* query. Specifically, the second factor is the closeness of the line that characterizes a view to the line that characterizes the query. An interesting idea would be to efficiently structure the collection of materialized views in main memory where indices could be used for this purpose. The depth of the view could be expressed either as the number of tuples contained in the view, or the actual score of the last tuple materialized in the view. As for the second characteristic, it is more complicated due to the fact that the scoring function of the query is not obligatory know a-priori. Therefore, it would be interesting to find a way to describe the position of the line that characterizes the view in the space regardless of the query line. For a line in N dimensional space, N-1 angles are needed in order to position the slope of the line in space. N-1 angles however are not so efficiently indexed, in general. This could possibly be solved by adopting spherical coordinates.

REFERENCES

[BNST05] W. Balke, W. Nejdl, and W. Siberski, U. Thaden, “Progressive Distributed Top k Retrieval in Peer-to-Peer Networks”, Proceedings of the International Conference on Data Engineering (ICDE), pp. 174-185, 2005.

[Baik11] Similarity Measures for Multidimensional Data User study. Available at http://www.cs.uoi.gr/~ebaikou/publications/2011_ICDE.

[BaRV11] E. Baikousi, G. Rogkakos, P. Vassiliadis, "Similarity Measures for Multidimensional Data", Proceedings of International Conference on Data Engineering (ICDE), pp. 171-182, 2011.

[BaVa07] E. Baikousi, P. Vassiliadis, “Tuning the top-k view update process”, Proceedings of 3rd Multidisciplinary Workshop on Advances in Preference Handling (M-Pref), 2007.

[BaVa09] E. Baikousi, P. Vassiliadis, "View Usability and Safety for the Answering of top-k Queries via Materialized Views", Proceedings of the International Workshop on Data Warehousing and OLAP (DOLAP), pp. 97-104, 2009.

[BaVa10] E. Baikousi, P. Vassiladis, "Maintenance of top-k materialized views", Distributed and Parallel Databases (DAPD), vol. 27(2), pp 95-137, 2010.

[BoKS01] S. Borzsonyi, D. Kossmann, K. Stocker, "The skyline operator", Proceedings of the International Conference on Data Engineering (ICDE), pp. 421-430, 2001.

[CaWa04] P. Cao, Z. Wang, “Efficient top-K query calculation in distributed networks”, Proceedings of the Principles of Distributed Computing (PODC), pp. 206-215, 2004.

[CBC++00] Y.Chang, L. D. Bergman, V. Castelli, C. Li, M. Lo, J. R. Smith, "The onion technique: Indexing for linear optimization queries", Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 391-402, 2000.

[ChGM04] S. Chaudhuri, L. Gravano, A. Marian, "Optimizing Top-k Selection Queries over Multimedia Repositories", IEEE Trans. Knowl. Data Eng., vol. 16(8), 2004.

[ChGr99] S. Chaudhuri, L. Gravano, "Evaluating Top-k Selection Queries", Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 397-410, 1999.

[DeSc02] M. H. DeGroot, M. J. Schervish, “Probability and statistics”, Addison Wesley, 2002.

[DGKT06] G. Das, D. Gunopulos, N. Koudas, D. Tsirogiannis. “Answering Top-k Queries Using Views”, Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 451-462, 2006

[Fagi96] R. Fagin, “Combining fuzzy information from multiple systems”, Proceedings of the Symposium on Principles of Database Systems, pp. 216-226, 1996.

[Fagi98] R. Fagin, “Fuzzy queries in multimedia database systems”, Proceedings of the Symposium on Principles of Database Systems, pp. 1-10, 1998.

[FaLN01] R. Fagin, A. Lotem, M. Naor, "Optimal aggregation algorithms for middleware", Journal of Computer and System Sciences, vol. 66, pp. 614-656, 2003.

[FuWY05] B. C. M. Fung, K. Wang, and P. S. Yu, "Top-Down Specialization for Information and Privacy Preservation", Proceedings of the International Conference on Data Engineering (ICDE), pp. 205-216, 2005.

[GMNS09] A. Giacometti, P. Marcel, E. Negre, A. Soulet, “Query Recommendations for OLAP Discovery Driven Analysis”, Proceedings of the International Workshop on Data Warehousing and OLAP (DOLAP), pp. 81-88, 2009.

[Graef00] G. Graefe, "Dynamic Query Evaluation Plans: Some Course Corrections?", IEEE Data Eng. Bull., vol. (23) 2, 2000.

[GuBK00] U. Güntzer, W. Balke, W. Kießling, “Optimizing Multi-Feature Queries for Image Databases”, Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 419-428, 2000.

[HrKP01] V. Hristidis, N. Koudas, Y. Papakonstantinou, “PREFER a system for the efficient execution of multi-parametric ranked queries”, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 259-270, 2001.

[HrPa04] V. Hristidis, Y. Papakonstantinou, “Algorithms and applications for answering ranked queries using ranked views”, VLDB Journal, vol. 13(1), pp. 49-70, 2004.

[HuKR93] D. P. Huttenlocher, G. A. Klanderman, W. J. Rucklidge, “Comparing images using the hausdorff distance”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15(9), pp. 850-863, 1993.

[IlBS08] I.F. Ilyas, G. Beskales, M.A. Soliman, "A survey of top-k query processing techniques in relational database systems", ACM Computing Survey, vol. 40(4), 2008.

[Josl04] C. Joslyn, “Poset Ontologies and Concept Lattices as Semantic Hierarchies”, Proceedings of the International Conference on Conceptual Structures (ICCS 2004), pp. 287-302, 2004.

[Koss00] D. Kossmann, "The State of the art in distributed query processing", ACM Computing Survey, vol. 32(4), pp. 422-469, 2000.

[LiBM03] Y. Li, Z. A. Bandar, D. McLean, “An approach for measuring semantic similarity between words using multiple information sources”, IEEE Transactions on Knowledge and Data Engineering, vol. 15(4), pp. 871-882, 2003.

[MaBG04] A. Marian, N. Bruno, L. Gravano, "Evaluating top-k queries over web-accessible databases", ACM Transactions on Database Systems, vol. 29(2), pp. 319-362, 2004.

[MCYC06] N. Mamoulis, K. H. Cheng, M. L. Yui, D. W. Cheung, "Efficient aggregation of ranked inputs", Proceedings of the International Conference on Data Engineering (ICDE), pp. 72-83, 2006.

[MiTW05] S. Michel, P. Triantafillou, G. Weikum, “KLEE: A Framework for Distributed Top-k Query Algorithms”, Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 637-648, 2005.

[NeRa99] S. Nepal, M. V. Ramakrishna, “Query processing issues in image (multimedia) databases”, Proceedings of the International Conference on Data Engineering (ICDE), pp. 22–29, 1999.

[Rogk10] G. Rogkakos, “Similarity Measures for Multidimensional Data,” MSc thesis, Univ. of Ioannina, Ioannina, Greece, July. 2010.

[Rous97] N. Roussopoulos, "Materialized Views and Data Warehouse", Proceedings of the KRDB Workshop, pp. 12.1-12.6, 1997.

[SaJa95] S. Santini and R. Jain, “Similarity matching”, Proceedings of the Asian Conference on Computer Vision (ACCV), pp. 571–580, 1995.

[SaJa99] S. Santini and R. Jain. ”Similarity measures”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21(9), pp.871–883, 1999.

[Sara01] S. Sarawagi, “idiff: Informative summarization of differences in multidimensional aggregates”. Data Mining and Knowledge Discovery, vol. 5(4), pp.255–276, 2001.

[Sara99] S. Sarawagi, “Explaining differences in multidimensional aggregates”, Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 42-53, 1999.

[SaSc05] P. Sanders and D. Schultes, “Highway Hierarchies Hasten Exact Shortest PathQueries”, Proceedings of the Annual European Symposium (ESA), pp. 568-579, 2005.

[SGAE04] O. D. Sahin, A. Gupta, D. Agrawal, A. El Abbadi, “A Peer-to-peer Framework for Caching Range Queries”, Proceedings of the International Conference on Data Engineering (ICDE), pp. 165-176, 2004.

[TPK++03] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, D. Srivastava, "Ranked join indices", Proceedings of the International Conference on Data Engineering (ICDE), pp. 277-288, 2003.

[Triv02] K. Trivedi, “Probability and statistics with reliability, queuing and computer science applications”, John Wiley & Sons, Inc, 2002.

[TrNY04] P. Triantafillou, N. Ntarmos, J. Yannakopoulos, “A Cache Engine for E-Content Integration”, IEEE Internet Computing, vol. 8(2), pp. 45-53, 2004.

[VaSk00] P. Vassiliadis, S. Skiadopoulos, “Modeling and Optimization Issues for Multidimensional Databases”, Proceedings of the International Conference CAiSE, pp. 482-497, 2000.

[VDNV08] A. Vlachou, C. Doulkeridis, K. Norvaag, M. Vazirgiannis, “On efficient top-k query processing in highly distributed environments”, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 753-764, 2008.

[YYY+03] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, Yuguo Chen. “Efficient Maintenance of Materialized Top-k Views”, Proceedings of the International Conference on Data Engineering (ICDE), pp.189-200, 2003.

[ZADB06] P. Zezula, G. Amato, V. Dohnal and M. Batko, “Similarity Search: The Metric Space Approach”, Advances in Database Systems, Springer, vol. 32, 2006.

[ZhTZ07] K. Zhao, Y. Tao, S. Zhou, “Efficient top-k processing in large-scaled distributed environments”, Data Knowledge Engineering, vol. 63(2), pp. 315-335, 2007.

ΔΗΜΟΣΙΕΥΣΕΙΣ ΣΥΓΓΡΑΦΕΑ

[BaRV11] E. Baikousi, G. Rogkakos, P. Vassiliadis, "Similarity Measures for Multidimensional Data", Proceedings of International Conference on Data Engineering (ICDE), pp. 171-182, 2011.

[BaVa07] E. Baikousi, P. Vassiliadis, “Tuning the top-k view update process”, Proceedings of 3rd Multidisciplinary Workshop on Advances in Preference Handling (M-Pref), 2007.

[BaVa09] E. Baikousi, P. Vassiliadis, "View Usability and Safety for the Answering of top-k Queries via Materialized Views", Proceedings of the International Workshop on Data Warehousing and OLAP (DOLAP), pp. 97-104, 2009.

[BaVa10] E. Baikousi, P. Vassiladis, "Maintenance of top-k materialized views", Distributed and Parallel Databases (DAPD), vol. 27(2), pp 95-137, 2010.

SHORT CV

Eftychia Baikousi was born in 1982 in Ioannina, Greece. She received her B.Sc. in Mathematics in 2003 from the Department of Mathematics in the University of Ioannina. She received her M.Sc. in Computer Science from the University of Manchester Institution Science and Technology in 2004. Ms Baikousi joined the Distributed Data Management Laboratory in 2005. Her research interests focus on top-*k* query processing and similarity of data points.