Towards Keyword-Driven Analytical Processing

Ping Wu[1][*] Yannis Sismanis[2] Berthold Reinwald[2]

[1]University of California
Santa Barbara, CA, USA
pingwu@cs.ucsb.edu

[2]IBM Almaden Research Center
San Jose, CA, USA
{syannis,reinwald}@us.ibm.com

ABSTRACT

Gaining business insights from data has recently been the focus of research and product development. OnLine-Analytical Processing(OLAP) tools provide elaborate query languages that allow users to group and aggregate data in various ways, and explore interesting trends and patterns in the data. However, the dynamic nature of today's data along with the overwhelming detail at which data is provided, make it nearly impossible to organize the data in a way that a business analyst needs for thinking about the data. In this paper, we introduce "Keyword-Driven Analytical Processing"(KDAP), which combines intuitive keyword-based search with the power of aggregation in OLAP without having to spend considerable effort in organizing the data in terms that the business analyst understands. Our design point is around a user mentality that we frequently encounter: "users don't know how to specify what they want, but they know it when they see it". We present our complete solution framework, which implements various phases from disambiguating the keyword terms to organizing and ranking the results in dynamic facets, that allow the user to explore efficiently the aggregation space. We address specific issues that analysts encounter, like joins, groupings and aggregations, and we provide efficient and scalable solutions. We show, how KDAP can handle both categorical and numerical data equally well and, finally, we demonstrate the generality and applicability of KDAP to two different aspects of OLAP, namely, finding exceptions or surprises in the data and finding bellwether regions where local aggregates are highly correlated with global aggregates, using various experiments on real data.

Categories and Subject Descriptors

H.2 [**Database Management**]: Miscellaneous

[*]Work performed at IBM Almaden Research Center

General Terms

Algorithms, Experimentation, Theory

Keywords

keyword search, OLAP, data warehouses, aggregation, ranking

1. INTRODUCTION

OnLine-Analytical Processing (OLAP) and Decision Support (DSS) are recognized as challenging problems with huge practical impact. Much effort has been concentrated in providing easy to use techniques that efficiently provide business insights. Off-the-shelf OLAP and DSS tools are based on a *navigational search* paradigm, that guides a decision maker through hierarchical dimensions. Domain experts use a set of hierarchical dimensions to navigate historical data, looking for interesting trends and patterns. Such hierarchies include the organization of time in months and quarters, the grouping of customers depending on age and income, and the organization of products based on classes, types and families. An OLAP/DSS expert uses such hierarchies to navigate and aggregate existing sales data, trying to gain insight and support business decisions. Data warehouse administrators spend a considerable amount of time and effort, in order to provide a sufficient set of such hierarchical dimensions that will allow OLAP analysts to properly analyze historical data. However, the overwhelming level of detail and the dynamic nature of the data that modern businesses are dealing with, rarely reflect the way that business analysts think about the data. Therefore, considerable resources are continuously spent to organize and reorganize data in a way that is useful to a business analyst.

An analyst might be interested in sales about "LCD", regardless of the underlying data being organized in categories like "Projector Technology=LCD" or "Department=Monitor, Product=Flat Panel(LCD)" or even "Department=Electronics, Product=Televisions, Category=LCD TVs". In practice, the problem is much more difficult and interesting, since the analyst is interested in combinations of many different dimensions -and not just the product which we use in this tiny example-. We make the key observation, that for a large class of practically motivated scenarios, conventional OLAP queries do not need to be expressed in a strict and exact query language such as SQL or MDX, but a much simpler and intuitive keyword based interface is sufficient. In this paper, we propose a novel framework *Keyword-Driven Analytical Processing*, that combines elements of *navigational-*

search with *multi-faceted* search [23, 22, 11] while -most importantly- addressing multi-dimensional aggregation that OLAP/DSS applications require.

In our system, a user starts by entering a keyword query such as "Columbus LCD". The system finds the best matches and interpretations for the query and provides a navigational refinement on the results of the query. Unlike traditional multi-faceted search, the results of the query are not existing documents, but the result of aggregating many database records. In the above example, the system will aggregate the sales about "LCD", and it will break-down the result into sub-aggregates for "Projector Technology=LCD", "Department=Monitor, Product=Flat Panel(LCD)", etc. It will do so only for records that link with "Columbus", which might have different meanings. It can mean "Columbus Day" or Columbus City or even customers called Columbus. The system will interact with the user to disambiguate the meaning of the keyword. Then the system will organize the aggregated results in facets that will allow the system analyst to navigate the relevant aggregation space. To enhance this navigational ability, the returned facets include dimensions that were not directly specified in the keyword query. In our example, those can be time and customer age and income.

There are many challenges in such an approach and -to the best of our knowledge- our system is the first one that addresses all of them, combining the ease-of-use of multi-faceted search with the aggregation power of OLAP. The first challenge towards keyword-driven analytical processing is to determine what the user is looking for. Instead of retrieving particular objects or entities from the database, the user is looking for aggregated results over certain portions of the data space. In this paper, we formulate this goal as the *subspace retrieval problem* in which the system automatically performs some navigational operations on behalf of the user and identifies interesting subspaces for further analysis. Unfortunately, the freedom of expressiveness of a keyword query comes at the price of precision. In fact, as we will show, keyword queries can almost always be interpreted in many different ways leading to a large number of subspaces over complex data spaces. The second challenge that we addressed is a systematic method to organize the aggregation results so that (a) interesting insights can be easily observed without overwhelming the user with way too many aggregates for different groupings and (b) it is easy for the user to further navigate the result space.

Our solution for KDAP takes into account the limited screen real-estate and consists of two phases. The first phase enumerates all possible subspaces for a given keyword query and groups and ranks these interpretations for the user to choose from. In our example, the user could be interested in LCD sales from stores in Columbus City. The system identifies the join paths and lets the user pick the one they want. Our approach follows the mentality that "users don't know how to express what they need, but they know it when they see it". Once the specific subspace is identified by the user, in the second phase, the system organizes "neighborhood-partitions" of the subspace into *dynamic facets*. These dynamic facets allow the user to rollup and drill-down and in general navigate the result space. The potential number of dynamic facets grows exponentially with the number of dimensions, the number of attributes and the number of hierarchical levels of the dimensions. Our system handles this problem by ranking the dynamic facets and the content of each facet such that interesting aggregations surface high on the ranked list, based on a novel measure of *interestingness*. Interestingness is application-specific and for the purpose of this paper, we focus on (a) surprising aggregates that deviate a lot with respect to the rollup aggregates[19, 20] and (b) correlated aggregates that hint the existence of bellwethers[9] in the data. For the first scenario, a user might be interested in finding exceptions and surprises in the data, and therefore, deviation from the average is the measure of interestingness. In our example, a surprising result could be that the sales for LCD equipment and young customers is significantly lower (or higher) than the average sales of LCD equipment. For the second scenario, the users are interested for quite the opposite. They are looking for local regions which determine aggregates for larger and maybe global regions. For example, assume that the sales of LCDs in Columbus City during January are very correlated with the total sales of LCDs. In that case, the dynamic facet that corresponds to January will be ranked much higher than other facets. Our framework accommodates such interestingness measures, and therefore, is useful in many different aspects of OLAP.

In particular, we make the following contributions:

1. We introduce keyword-driven analytical processing, a novel framework that allows the user to get business insights by combining the benefits of multi-faceted search and navigation with the power of OLAP aggregation.

2. We formalize many challenges that were raised by this framework, demonstrating the richness of the problem and many opportunities for future research.

3. We develop many efficient and scalable algorithms that make KDAP practical, in particular the candidate subspace generation algorithm, novel ranking for the candidates, and dynamic facet construction.

4. We demonstrate the value of our approach using various datasets.

The rest of the paper is organized as follows. The related work is discussed in Section 2. We present the fundamentals in Section 3. We discuss our candidate subspace enumeration and probabilistic subspace ranking algorithms in Section 4. In Section 5, we introduce our dynamic facet construction algorithms. The qualitative results are presented in Section 6. Finally, we discuss extensions and draw conclusion in Section 7.

2. RELATED WORK

Motivated by the wide acceptance of keyword search, keyword search over relational databases has attracted significant attention in the research community[5, 6, 18, 3, 16, 4, 15, 7]. The research efforts primarily focus on the problem of retrieving entity relationships from structured databases using user-friendly keyword search. Some well-known systems in this area include DBExplorer [6], Discover [16], and BANKS [5]. Given a keyword query, these systems combine relevant information scattered in different relational tables that match one of the keywords and join together through primary key/foreign key relationships. The query result is a *joined tuple tree*. Similar functionality has been investigated for semi-structured databases [10, 13, 17], where the inherent rich graph structure can be exploited for authority-based ranking [13] as well as query result presentation.

On the analytical side, OLAP systems are the primary solution. OLAP systems perform adhoc analytical queries over multi-dimensional historical data with fact data and dimension data. The fact data contains numeric measures such as sales volume, and the dimension data specifies the context in which the corresponding fact occurred. Dimension data is typically hierarchical in nature, and links to the fact table via primary key/foreign key relationships. Business analysts iteratively explore different views of the multi-dimensional data space through a set of OLAP navigation operations such as *slice-dice*, *drill-down*, *roll-up* and *pivot*, interact with the data presentation and adjust the aggregation level and granularity until actionable business insight can be drawn. We refer to [8] for an excellent survey of this area.

The exploratory nature of OLAP systems bears some resemblance to multi-faceted search systems developed in the IR community [23, 22]. Multi-faceted search systems provide operations to browse complex data spaces and zoom in/out along multiple facets. Such search interfaces [14, 23] have been proved to be very effective in locating items of interest. For example, in the Flamenco system [1], users can issue keywords to search an image database of Fine Art, and then use the multi-faceted search interface to browse the result set. A user may first zoom in along the location dimension (`all` → `North America` → `United States` → `New York`), and then browse through the Media dimension ("Sculpture"). However, multi-faceted search systems provide support only for "object retrieval" in nature, and are not suited for analytics or OLAP aggregation.

Keyword-based queries are currently not adequately supported in data analysis scenarios. To our knowledge, Google Trends [2] is the only system that provides some rudimentary KDAP functionality to end users. Google Trends exposes a multi-faceted search interface over the query log, and shows the aggregated search query volume for the typed keywords over time and location. However, general OLAP data models are often much more complicated, and for high-dimensional dataspaces a dynamic consideration of group-by attributes based on interestingness of the aggregation values is required.

3. SOLUTION FRAMEWORK

In this paper, we use "dataspace" to refer to the entire multi-dimensional dataspace DS in an OLAP database and "subspace" to refer to a subset DS' of the dataspace. We refer to dimension D_i as a group of attributes and denote the j^{th} attribute from D_i as "attribute" $attr_i^j$ and "attribute instance" $attr_i^j.cat_p$ as the p's attribute value (category) within the domain of $attr_i^j$.

The proposed solution is depicted in Figure 1 and operates in two phases: *differentiate* and *explore*. In the differentiate phase, the system uses the user keywords to generate the candidate subspaces. Each subspaces essentially corresponds to a possible join path between the dimensions and the facts. We refer to such join paths as "star joins". Different interpretations of the keywords reflect different star join expressions. A selected interpretation determines the subspace in the multi-dimensional dataspace that contains

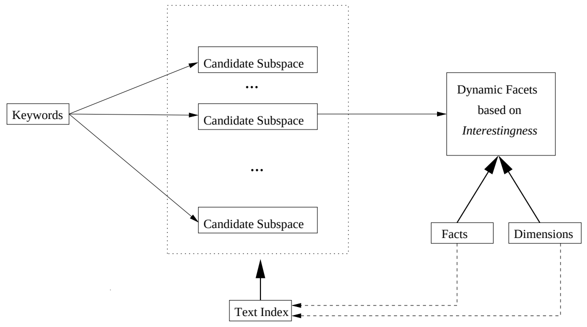


Figure 1: Solution Overview

a fraction of the tuples from the fact table. For example, a subspace might contain all the sales transactions for iPods occurred in stores in California. In the explore phase, the system first calculates for the subspace the aggregated values for some predefined measure, and then dynamically finds for each dimension the top-k *interesting* group-by attributes to partition the sub data space.

Interestingness is application-specific and for the purposes of this paper we focus on (i) surprising aggregates[19, 20] that deviate a lot with respect to the rollup aggregates and (ii) correlated aggregates that hint the existence of bellwethers[9] in the data. Our focus is to demonstrate the applicability and usefulness of our approach to many different scenarios and not currently to demonstrate performance.

In order to complete the construction of each dynamic facet the system uses the aggregation values of the top-k most interesting attribute instances. The result is organized similar to a multi-faceted search interface and can enable seamless incorporation of existing OLAP navigational operations as well. For example, each attribute instance may serve as an entry point for drill-down operations to more detailed subspaces.

In addition to the typical dimensional and fact tables and indexes our system requires the population and usage of a text-index which enables fast and approximate direct search on both dimensions and facts. We store the distinct attribute values from each attribute domain and treat each attribute instance as a virtual document. Conceptually we are using a relation like ⟨ TabName, AttrID, Document ⟩ which is indexed on Document and allows for keyword search on Document. The pair ⟨ TabName, AttrID ⟩ encodes the origin of the corresponding attribute domain. Note that current research on keyword search over relational databases uses indexes on a tuple level instead of an attribute level. More specifically, each tuple is treated as a virtual document and the result of a single keyword query is called the *tuple set.* This technique is sufficient for the object-retrieval scenarios, where the focus is on finding objects/relationships associated with the keyword, however, it is not enough for the demand of analytical operations. Consider the following simple example, two dimensional tuples from product dimension: PRODUCT$_A$ = {Product{ABC, EFG}, Category{TGS, SDF}} and PRODUCT$_B$ = {Product{ERT, EFG}, Category{ABC}}. If the keyword is "ABC", tuple level indexing will treat both tuples as equal matches and there is no way to *differentiate* the semantic differences. As we will discuss in the next section, query disambiguation is crucial for keyword-driven analytical processing.

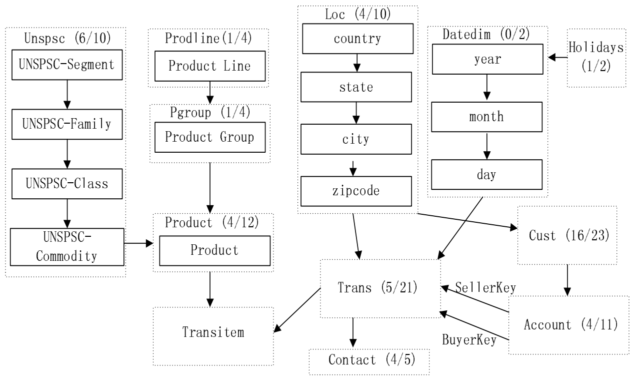


Figure 2: Running Example: the EBiz Schema

The text-index is used for both the differentiate and explore phase. It's role is (a) to provide a direct and uniform search mechanism using techniques like partial matches and stemming over OLAP data and (b) to provide a relevance metric over the returned results that our system uses to complete the computation of the score for different query interpretations.

Figure 2 shows the running example that we use throughout the paper. The EBiz data schema describes a typical e-commerce data warehouse. Conceptually, there are 4 dimensions: `Time`, `Store`, `Customer`, and `Product`. The time dimension consists of two tables and has the hierarchy `Year` $\rightarrow$ `Quarter` $\rightarrow$ `Month` $\rightarrow$ `Week` $\rightarrow$ `Date`. Each dimension is implemented in multiple tables. The dot-lined rectangles represent one logical table in the database. For each table, we also show in parenthesis the number of text attributes that are full-text searchable and the total number of attributes. Within each dot-lined rectangle, the arrows between attributes represent aggregation hierarchies, whereas the arrows between tables show primary key/foreign key relationships. For example, the Product dimension has two hierarchies: the UNSPSC hierarchy and the Product Line/Product Group hierarchy. The Customer table joins with the fact table through the Account table. Since the same customer can be both seller and buyer, the Account table joins with transaction table on the `BuyerKey` as well as the `SellerKey`.

4. SUBSPACE DISAMBIGUATION

In this section, we focus on the differentiate phase of our system. We describe the candidate subspace generation algorithm, and show how to effectively organize and rank the generated candidates.

4.1 Ambiguity of Keyword Queries in Complex Dataspaces

In large and complex OLAP dataspaces, a keyword almost always matches different attribute domains in different dimensions. This issue, combined with the fact that the multi-dimensional data model allows for potentially any combination between different matched dimensions, creates an exponentially large number of possible query interpretations. We illustrate the keyword ambiguity problem in KDAP with the following example.

Example 3.1 In Figure 2, consider the simple keyword query "Columbus LCD". This query might be very ambiguous as the keyword "Columbus" may refer to either a holiday or a city. Furthermore, even as a city, users may be interested in either sales in `Stores` in Columbus or `Customers` from Columbus. In addition, "LCD" may hit different instances at different levels of the `Product` dimension. Therefore, in a multi-dimensional database, all the possible combinations of these interpretations lead to a large number of different subspaces. □

Two kinds of ambiguities can be observed in the above example. We define *attribute instance ambiguity* to describe cases where a keyword matches attribute domains at different levels across dimensions. In the above example, the ambiguity associated with the keyword "LCD" falls into this category. We define *join path ambiguity* to describe cases where the corresponding tables of matching attribute domains can join with the fact table through different paths. In the above example, the existence of three different join paths between the `Location` table and the fact table (`TRANS`) creates different semantic interpretations for keyword "Columbus". Note that both ambiguities may occur on the same keyword.

The ambiguity problem has not been addressed by existing research on keyword search over relational databases. In KDAP, the correct interpretation of the keywords in a query becomes crucial for two reasons. First, different interpretations may result in completely different subspaces. Selecting the right query interpretation early on may eliminate error propagation to subsequent phases in the system. The second reason is more on the performance and interactivity side. Due to the potentially large number of matched query subspaces, computing the aggregation values for *all* possible interpretations before checking with the user may greatly reduce the responsiveness of the system. Third, users know exactly the semantic meaning of their keywords. By putting them in the the query processing loop, such ambiguities can be easily resolved.

4.2 Candidate Interpretation Generation

Given a *keyword query* q=$\{k_1, k_2, ...k_n\}$, the system generates candidate interpretations CI =$\{C_1, C_2, ..., C_m\}$ of q. For each keyword k_i, the CI generator first probes the full-text index to obtain the **hit set** H_i for k_i. Each **hit** $h_i^j \in H_i$ represents the attribute instance that matches the keyword k_i. Each hit h_i^j represents a triplet: the table name $h_i^j.R$, the attribute name $h_i^j.Attr$ and the attribute instance value $h_i^j.Val$. In Example 3.1, the hit set for keyword "Columbus" has two hits: Loc/City/"Columbus" and Holiday/Event/"Columbus Day". Within each hit set H_i, we organize the hits into different **hit group**s. All hits in the same hit group HG_i^k are drawn from the same attribute domain, i.e. $HG_i^k \subset H_i, \forall h_i^j, h_i^l \in HG_i^k, h_i^j.R = h_i^l.R \wedge h_i^j.Attr = h_i^l.Attr$. For example, the hit groups for keyword "LCD" could be {PGROUP/Group Name/("LCD Projectors" OR "Flat Panel(LCD)")}.

We define a **star seed** SS of a keyword query q as a set of n hit groups $HG_1^{K_1}, HG_2^{K_2}, ..., HG_n^{K_n}$, each of which is drawn from a different hit set. K_i denotes the number of hit groups for keyword k_i. For example, one candidate star seed for the keywords "Columbus LCD" could be {Loc/City/"Columbus", {PGROUP/Group Name/("LCD Projectors" OR "Flat Panel(LCD")}}. We define a **star net** SN of SS as a join that connects all the hit groups from the SS. A single star seed could correspond to several star nets, as there may exist multiple join paths that connect the hit groups together.

Existing techniques on keyword-based search over relational databases deploy a breadth-first graph traversal strategy on the schema graph to enumerate possible join paths. We adopt this approach as well but with modifications to accommodate the unique characteristics of KDAP. First, existing techniques have the hard-wired assumption that the answer tuple tree should obey the *minimal principle*[16], i.e. the leaf tables of any candidate tuple tree should contain at least one keyword. If we apply the same principle to KDAP, the keyword query "Home Electronics, VCR" will produce `UNSPSC` $\bowtie$ `Product` $\bowtie$ `ProductLine` as one star net. The star net returns common product records that belong to both hit groups in `UNSPSC` and the `ProdLine` tables. However, our result semantics require us to slice the subset of tuples from the fact table although they may not contain any keywords. Therefore, we require that the star net must contain the fact table, and the join expression must go through the fact table. Intuitively, hit groups from the dimensional data specify the region of the subspace, whereas hit groups from the fact table further select a subset of data points that belong to that particular subspace (e.g. purchase transactions). As a result, the same query in KDAP includes all the purchase transactions from the fact table with product items whose UNSPSC family title is "Home Electronics" and product line is VCR. This change commands a modification of the stopping condition in existing algorithms.

Furthermore, the complex data models in OLAP call for a more careful treatment of join paths as well as table aliases. On one hand, the same table in OLAP may belong to multiple dimensions (e.g. the `Location` table belongs to both, the `Customer` dimension and the `Store` dimension). For instance, one candidate interpretation of query "Seattle Portland TV" could be: all TV purchases made by `Customers` from Seattle in `Stores` located in Portland. In such a case, table aliases need to be used for the `Location` in order to distinguish the different semantic meanings of the same table. On the other hand, there may exist multiple hierarchies in one single dimension. In Figure 2, `UNSPSC` and `ProdLine` form two different hierarchies on the `Product` dimension and both tables join with the fact table through the `Product` table. In this case, although the `Product` table appears twice in the star net, the "intersection" semantic suggests to merge both into one single table expression. In summary, unlike [16], only a subset of the candidate join networks is considered as a valid interpretation according to the OLAP semantics.

Based on the above observations, we briefly describe our star net generation algorithm in Algorithm 1. First, the system probes the full-text index and produces a hit set for each keyword. The hits in each hit set are then divided into hit groups. For each hit group, the system finds all the join paths from its hit table to the fact table in the schema graph. Finally, it enumerates all the possible combinations of the hit groups' paths from different hit sets to produce candidate star nets. For each resulting star net, table aliases are used if the same table is involved in multiple join paths of different dimensions.

4.3 Handling of Phrase Queries

The method discussed so far does not handle phrase queries accordingly. Consider the query "San Jose". Although the system will return "San Jose" as a city in the hit set of either keyword "San" and "Jose", it also generates other noise as well such as "San Antonio", etc. Furthermore, the score function does not take into consideration that "San Jose" perfectly matches two keywords and therefore should be assigned much higher score than "San Antonio". In order to handle phrase queries accordingly for any star net, our system further merges the hit groups $HG_i^{k_i}$, $HG_j^{k_j}$ from two hit sets H_i and H_j, if (a) both groups are from the same attribute domain, i.e. $HG_i^{k_i}.Attr = HG_j^{k_j}.Attr \wedge HG_i^{k_i}.R = HG_j^{k_j}.R$, and (b) the intersection between both groups is not empty, i.e. $\exists$ hit $h_p, h_p \in HG_i^{k_i} \wedge h_p \in HG_j^{k_j}$. The latter requirement prevents us from merging two non-overlapping hit groups even though they belong to the same attribute domain. For example, if the user wants to get two "slices" from the product dimension by typing "Software" and "Electronics", then the system should display these two independent instances side-by-side.

After the merge, the two hit groups are replaced by their intersection. Please note that the above merge process can be easily generalized to cases beyond two hit groups if the phrase consists of more than 2 keywords. The relevance score for all the hits within the merged groups may become obsolete after the merge since the original score only reflects the similarity between the single keyword and the textual attribute instance. Consequently, the system also needs to update the score by consulting the full-text engine again with the newly-merged phrase query.

Algorithm 1 Candidate Star Net Generation

```
 1: Procedure
 2: for each keyword $k_i$ in $q$ do
 3:     produce hit groups $HG_i^1$, $HG_i^2$,...,$HG_i^{K_i}$
 4: end for
 5: for each hit group $HG_i^j$ of $k_i$ do
 6:     find all the join paths connecting to the fact table
 7: end for
 8: generate all the combinations of hit groups' join paths
    from different hit sets $\longrightarrow SN$
 9: for each star net $sn_p$ in $SN$ do
10:     merge tables from the same dimensions and distin-
        guish same tables from different dimensions
11: end for
12: End Procedure
```

4.4 Ranking the Candidate Star Nets

In complex dataspaces, the number of all interpretations for some keyword query may be large. Therefore, it is necessary to rank these interpretations, before returning them to the user to select the intended true interpretation.

Recent research on keyword search over relational databases proposes different ranking heuristics. For example, both DBExplorer[6] and DISCOVER[16] rank tuples simply based on the size of the corresponding join networks. The later refined algorithm in DISCOVER introduces the relevance score between the text attribute values and the keyword query into the ranking formula. A most recent study[18] adapts even more sophisticated normalization techniques from the IR literature and makes it suitable for ranking relational tuple trees.

We propose our own ranking strategies primarily based on existing IR principles while taking into account the special semantics of the multi-dimensional data model. Our ranking formula is given as follows:

$$SCORE\,(SN, q) = \frac{\sum_{\forall \mathrm{HG_k} \in SN} \left(\sum_{\forall \mathrm{h_i} \in \mathrm{HG_k}} (Sim(h_i.val, q)) \Big/ |HG_k|(1 + \ln(|HG_k|) \right)}{|SN|^2}$$

where $Sim(h_i.val, q)$ denotes the similarity between the textual content of the hit's attribute value and the keyword query. The similarity value is calculated according to the state-of-the-art document-query similarity function in IR, which is implemented in both RDBMS full-text search modules and stand-alone text search engines (e.g. Lucene). This value captures the similarity such as proximity of matched keywords within the textual attribute value, the term frequency and inverse document frequency, etc. For each star net, our scoring method further aggregates these similarity values of all the hits based on the following heuristics. First, for each hit group $|HG_k|$, the average hit similarity value is normalized by the factor $ln(1 + |HG_k|)$. This is to penalize the hit groups whose corresponding attribute domain contains many matched attribute instances. For example, keyword "California" may hit California as a State or a large number of distinct addresses on "California Street". Second, the summation of hit group scores is divided by $|SN|^2$. This takes into account the number of hit groups in the star net and thereby prioritizes the star nets where multiple keywords occur in the same attribute instances. For example, star nets with "San Jose" as a city will be ranked higher than others with hit groups "San Antonio{city}" and "Jose{Customer First Name}".

5. AUTOMATIC FACET CONSTRUCTION FOR SUB-DATASPACES

In this section, we first formulate the automatic facet construction problem. Then we present our solutions for ranking group-by attributes within each dimension as well as ranking attribute instances within each selected attribute domain. Our explanation covers both categorical attribute domain and the numerical attribute domain. While our discussion is limited to summations over sales' volume, it can be extended to allow users to define their own measures and aggregation functions.

5.1 Problem Formulation: Adapting Multi-Faceted Interface for KDAP

After the interpretation disambiguation phase, a unique sub-dataspace DS' has been identified by the user, and the system computes the aggregation values over the measure from all qualified fact points in DS'. The system then dynamically constructs a multi-faceted search (MFS) interface for the user to explore detailed level aggregations in DS'.

Attributes from the same dimension are displayed together. For each dimensional attribute $attr_i^j$, the breakdown of the aggregation values over different attribute values are shown. Since real-world OLAP schemata often have far more dimensional attributes than the number of facets in common MFS systems, users will be overwhelmed by the large number of group-by attributes if all of them are shown. Hence, effective organization of these interface elements becomes necessary for data exploration.

We further divide our problem into three subproblems: *dimension ranking*, *attribute ranking* and *instance ranking*. Such ranking strategies prioritize the interface elements according to their corresponding interestingness with respect to the current query. Users are still able to identify business insights hidden inside DS' with limited screen real estate. In most applications, the number of dimensions is relatively controllable, mostly under 10. Therefore, this paper assumes a static order over all dimensions (e.g. alphabetical order), and focuses only on attribute and instance ranking.

Previous work [21, 20] on OLAP exploration studied the notion of interestingness for data cube cells. In particular, Sarawagi et. al.[21] proposes a method to help users effectively navigate the cube space by highlighting the aggregation values that are surprising in the statistical sense. However, this approach assumes the availability of a fully-materialized cube with all the grouping attributes manually defined a-prior to construct the cube. In other words, regardless of the keywords of the current session, the set of group-by attributes remains the same for any sub-dataspaces during the exploration process. In contrast, our goal is to dynamically determine interesting grouping attributes as well as attributes instances for a particular sub-dataspace, that the user selects. The importance of attribute ranking has been recognized lately in object retrieval scenarios [12]. In [12], for each tuple in the top-k query result, the system dynamically highlights the "dominating attributes" that best explain the reason for the ranking of the corresponding tuple. While we share the goal of improving data exploration by prioritizing the important attributes, the techniques developed there are not applicable in KDAP.

5.2 Ranking Group-by Attributes

Intuitively, every group-by attribute defines a way to partition a dataspace and hence provides an unique angle for users to understand the data. Often some angles bring more interesting perspectives than others. We thus wish to rank the candidate group-by attributes based on the interestingness of their resulting partitions.

For an arbitrary sub-dataspace DS' associated with the star net SN_p, we dynamically select the top k most interesting group-by attributes from each dimension D_i for display. For each candidate attribute $attr_i^j$ from D_i, the set of all distinct values projected from DS' on attribute $attr_i^j$ is denoted as $DOM(DS', attr_i^j) = \{attr_i^j.cat_1, ..., attr_i^j.cat_m\}$, where $attr_i^j.cat_h (1 \leq h \leq m)$ stands for either one distinct categorical value or one numerical interval. We use cat_h to represent $attr_i^j.cat_h$ whenever $attr_i^j$ can be clearly inferred from the context. We denote the resulting partition of DS' grouped by $attr_i^j$ as: $PAR(DS', attr_i^j) = \{DS'|_{attr_i^j = cat_1}, ...,$ $DS'|_{attr_i^j = cat_m}\}$ where each $DS'|_{attr_i^j = cat_p}$ $(1 \leq p \leq m)$ stands for a group of fact points in DS' whose $attr_i^j$ value belongs to cat_p. Our algorithm measures the interestingness of each such partition with a score function and outputs the group-by attributes whose corresponding partitions are among the top k.

5.2.1 *Roll-up Partitioning*

We take a novel approach called "*roll-up partitioning*" (RUP) for selecting group-by attribute. The main observation is that by only looking at the sub-dataspace alone, it is impossible to define interestingness of a certain partition in a robust way. The basic idea of RUP is thus to leverage on the hierarchical dimensions available in todays' databases. More specifically, the system enlarges the targeted sub-dataspace DS' by generalizing (roll-up) along some dimensions to a bigger space $RUP(DS')$ as the background dataspace for comparison. Then we use the same candidate group-by attribute to partition both spaces and calculate the similarity of the two distributions of the aggregation values from the resulting groups in both partitions. The more similar the two distributions are, the less the candidate group-by attribute is considered as surprising.

Example 4.1 Suppose that we want to determine whether the `zip code` attribute of `Store` dimension is an interesting group-by attribute for the subspace associated with "Television" Products. We roll-up to a bigger space along the `Product` dimension to "Home Entertainment Electronics" category and see whether the trend of sales distribution across zip codes holds again or not. Let's assume that the distribution of aggregated TV sales volume over zip code significantly deviates from the general Home Entertainment products. Then for applications that are looking for exceptions, the zip code is considered as an interesting attribute to be shown to the user and it is ranked higher. On the other hand, for applications looking for bellwethers, we would rank higher the zip code iff the larger region aggregates are correlated with the local region aggregates.□

At this point, there are two subtle issues that should be clarified. First, the candidate attribute $attr_i^j$ is likely to partition $RUP(DS')$ into more than m groups simply because $RUP(DS')$ is the superset of DS' and hence contains more distinct attribute values in the domain of $attr_i^j$ than DS'. If this is the case, we only consider the resulting groups that also exist in the DS'. For ease of presentation, we use $PAR(RUP(DS'), attr_i^j)$ to denote only the same segments that also exist in $PAR(DS', attr_i^j)$, i.e. $PAR(RUP(DS'), attr_i^j) = \bigcup RUP(DS')|_{attr_i^j = cat_p}, cat_p \in DOM(DS', attr_i^j), 1 \leq p \leq m$. Second, not every attribute is eligible for being a group-by candidate. There are some attributes which clearly cannot produce any meaningful groups (e.g. product description articles, or Product IDs). In our current implementation, we manually specify the candidate group-by attributes within each dimension and the problem of automatic candidate group-by attribute discovery is left as future work.

Based on whether containing the hitted groups or not, dimensions are divided into two disjoint sets, **hitted dimensions** D_{hit} and **non-hitted dimensions** $\overline{D}_{hit}$. In D_{hit}, the attributes from the hit groups are directly selected as the group-by attributes of the corresponding. For example, if the star net SN_p contains hit group {("Home Electronics" OR "Office Electronics")\\`GroupName`\\`PGROUP` }, then in the `Product` dimension, `GroupName` is directly promoted to the group-by attribute to provide an entry point for the user to further reduce the ambiguity between the two product groups relating to the "Electronics".

If there exists more than one roll-up dimension, we repeat the roll-up partitioning process for each roll-up dimension. We pick the worst score from all scores, so that the most dissimilar case can be captured to reflect the possible surprising cases. For example, the Star Net {"Columbus"(`Location` →`Trans` →`Transitem`), "TV"(`Product`→`Transitem`)} has two "hitted" dimensions, and therefore can be rolled up along both dimensions, `STORE` and `PRODUCT`.

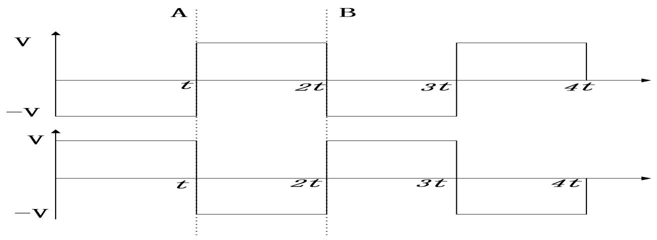


Figure 3: Numerical Domain Partitioning

For applications that are looking for exceptions/surprises, the final score of the group-by attribute $attr_i^j$ is computed as the negated correlation value between the two aggregation value series:

$$SCORE(attr_i^j, DS') = -\frac{E((X-\mu_x)(Y-\mu_y))}{\sigma_x \sigma_y} \quad (1)$$

where X, Y stand for the aggregation values on partition $PAR(DS', attr_i^j)$ and $PAR(RUP(DS'), attr_i^j)$, respectively. And each series consists of m values. E stands for the expected value, and μ, σ denotes the average and standard deviation of the corresponding data series, respectively. Similar formulas, can be derived for bellwether or other OLAP applications.

5.2.2 *Ranking both Categorical and Numerical Attributes*

The idea of roll-up partitioning can be easily applied to both categorical and numerical attributes. In the categorical case, the calculation is trivial since the segmentation of the attribute domain is determined by the attribute value and our previous score function can be directly applied (Equation 1). However, in the case of numerical attributes, the correlation value of the two aggregation series heavily depends on how the numerical domain is "bucketized". For instance, Figure 3 shows two distribution functions. With the aggregation function being summation, splitting points A and B lead to completely different correlation values.

For numerical candidate group-by attributes, our method first splits the domains of the candidate attribute into "sufficiently" many buckets. We call each of these small intervals a "**basic interval**", and all the fact tuples whose values in attribute $attr_i^j$ fall into the same basic interval, are aggregated together to produce new attribute values. The intuition is, that the correlation value of two distributions can be preserved as the bucket number becomes large and the interval range becomes small. We will empirically evaluate this assumption and investigate the effect of the number of buckets. In summary, our result supports that in most cases when the bucket number goes beyond a certain value (by default 20), the attribute score converges, i.e. the correlation value between the aggregated values of the roll-up space and the query space almost remains the same when we split the attribute domain into more than 20 intervals.

5.3 Organizing Attribute Instances

Once the group-by attributes are chosen, we are still faced with the problem of organizing the values within each attribute domain. Our goal is two-fold: first, we want to optimize the exploration so that more interesting categories can be quickly discovered; Second, we also want to optimize the navigational access. If users are looking for one particular aggregation value, they are able to find it quickly.

5.3.1 Categorical Attributes

We propose a simple ranking scheme, that captures the interestingness of the corresponding category. For each attribute value $attr_i^j.cat_q$, we compute the intra-attribute score as:

$$SCORE(attr_i^j.cat_p, DS') = \left| \frac{G(DS'|_{attr_i^j=cat_p})}{G(DS')} - \frac{G(RUP(DS')|_{attr_i^j=cat_p})}{G(RUP(DS'))} \right| \quad (2)$$

where G is the aggregation function. The score reflects the degree of deviation of the aggregation value of targeted subspace from its background space with respect to the subspace DS'. If there are several hitted dimensions, the scores of multiple roll-up partitionings must be combined.

5.3.2 Numerical Attributes

In the case of numerical attributes, the problem boils down to numerical domain partitioning. Given m basic intervals from the previous phase, our goal is to merge adjacent intervals together into K **numeric categories**.

The merge scheme must take into consideration three objectives. First, for the purpose of navigational access, the number of resulting intervals must be suitable for human browsing. While there can be more than 30 buckets in the attribute selection phase, the intervals that are actually shown to the user, must not be more than a dozen considering limited screen real estate. This is acceptable for multi-faceted search sessions, as a user can always choose to expand further into subsequent subintervals. Second, for navigational access, we require that the number of merged intervals should not be "skewed". For example, an interval is not a good candidate for splitting, if one bigger interval consists of twenty basic intervals, while another smaller interval consists of only two intervals. Specifically, we require that the number of intervals in the largest range should not exceed L times the number of intervals in the smallest range, where L is another system parameter. Third, and most importantly, for the purpose of exploration, the merged partition should preserve the original interestingness value, i.e. correlation value from basic intervals. Therefore, we require that the difference between the correlation values before and after the merge is minimized.

Our algorithm merges adjacent "basic intervals" created from the previous correlation computation phase into K numerical ranges for display, where K is given a-prior. The optimization goal is to minimize the change in the score of the attribute, and the constraint is, that no numerical category is more than L times bigger than the other. According to the above description, the numerical domain partitioning problem becomes a combinatorial optimization problem with constraints. Our solution adapts the simulated annealing algorithm to automatically assign "good" splitting positions over the attribute domain.

The basic idea is to start with equal-width partitioning and at each step to generate a neighbor of the current splitting scheme and check a) if the neighbor is a valid splitting scheme according to the constraint, and b) if the neighbor produces a better solution than the current solution, i.e. if the difference of the new correlation value (over merged intervals) and the old value (over the basic interval) becomes smaller. If yes, the neighbor splitting scheme is accepted. In order to avoid being trapped in local maximum, we may deliberately accept neighbors that increase the difference. The above process repeats N times and returns the best splitting scheme. A neighbor is generated by selecting one point from the current $K-1$ splitting points and randomly increasing or decreasing its position by one unit of the basic interval.

Algorithm 2 Splitting Points Assignment

```
1: Procedure
2: Basic Intervals $BI = \{BI_1, BI_2, ..., BI_m\}$
3: Current Splitting Points $CSP \leftarrow$ equal width splitting
   points of $BI$;
4: $BSP \leftarrow CSP$
5: i ← 0;
6: while $i < N$ do
7:    $TEMP \leftarrow$ generate a valid neighbor of $CSP$
8:    $a \leftarrow SCORE(TEMP, DS', RUP(DS'))$
9:    $b \leftarrow SCORE(BI, DS', RUP(DS'))$
10:   $c \leftarrow SCORE(BSP, DS', RUP(DS'))$
11:   if $| a - b | < | c - b |$ then
12:      $BSP \leftarrow TEMP$;
13:   end if
14:   if RANDOM() > some constant then
15:      $CSP \leftarrow TEMP$;
16:   end if
17:   i ← i + 1;
18: end while
19: return $BSP$
20: End Procedure
```

Please note that the above interval merge algorithm works on the basic intervals, that can be fetched during the previous group-by attribute ranking. Therefore, the partition merge algorithm runs solely in the application level and does not involve any further interactions with the RDBMS.

6. EXPERIMENTS

It is very challenging to provide a qualitative evaluation of the results, that an inherently imprecise keyword-based search system returns. We cannot directly use —or adapt— traditional benchmarks found in the IR community, since the scope of our work is novel and extends way beyond these benchmarks. Additional functionality that our system depends on, such as joins, groupings and aggregations, plus the requirements of the business analyst make the problem of evaluating KDAP very challenging.

Nevertheless, in this section, we show the applicability and effectiveness of our solution with sample results that we believe are representative of a practical setting. In addition, we experimentally analyze the quantified quality of the proposed subspace ranking methods and also study the impact of several key system parameters on the quality of dynamically constructed faceted interfaces.

6.1 Experimental Setup

For the qualitative evaluation, we use the AdventureWorks data warehouse that comes with SQL Server 2005 Analysis Server. We use two sizable fact tables, which are Internet Sales Fact and Reseller Sales Fact. Each contains more than 60,000 fact records. Since each fact table is associated with a slightly different set of dimension tables, we divide the data warehouse into two separate databases, which we denote as AW_ONLINE and AW_RESELLER, respectively. AW_ONLINE contains 5 dimensions, 10 tables, and three of the dimensions are hierarchical. AW_RESELLER, on the other hand, contains 7 dimensions, 13 tables, and 4 of its dimensions are hierarchical. Both databases have more than 20 full-text search-able attribute domains. The offline full-text index, that our KDAP system requires, takes around 5MB in both cases. We use sales revenue as the only measure, which is defined as the summation of the `UnitPrice` multiplied by `Quantity`.

We have implemented all the methods discussed in the paper in a prototype system built on top of a commercial RDBMS. We employ Lucene as our full-text search engine. The prototype system is written in Java and connects to the database server via a JDBC interface. All the experimental results are from a computer system with 1.66GHz CPU and 1GB of RAM.

6.2 Qualitative Sample Results

We demonstrate KDAP by describing a typical user interaction scenario. Consider a business analyst who wants to explore sales information about mountain bikes in California. The analyst enters the keywords "California Mountain Bikes", and the system returns a list of star nets, each of which represents one semantic interpretation of the query. In Table 1, we show the Top 3 hits returned by our prototype system. We omit join paths in the table due to space limitations, and only the hit groups are shown. We acknowledge that the correct answer is shown at the top of our list. Content summaries can be rendered as snippets when the textual attribute is big (e.g. product description).

DimGeography/ StateProvinceName/ {California}	DimProductSubcategory/ ProductSubcateory-Name/ {Mountain Bikes}		0.607669
DimCustomer/ AddressLine1/ {345 California Street, OR 392 ...}	DimProductSubcategory/ ProductSubcateory-Name/ {Mountain Bikes}		0.482669
DimGeography/ StateProvinceName/ {California}	DimProduct/ EnglishProduct-Name/ {Fender Set - Mountain(0.8) OR Mountain Pump}	DimProduct -Category/ CategoryName/ {Bikes}	0.322216

Table 1: Sample star nets returned for the keyword query "California Mountain Bikes"

The analyst proceeds by selecting the first star net. The system aggregates the fact tuples associated with the star net, and displays the content of the dynamically constructed facets as well as the aggregation values for different attribute instances. Elements on the facets are clustered according to dimension, attributes and attribute instances. Table 2 shows the system output of the corresponding attributes and the attribute instances from the product dimension. Please note, as "Mountain Bikes" appears in the keyword query, it will always be selected for navigational purposes, and the user can start the exploration from this value.

Product Dimension	
Group-by Attribute Selected	**Attribute Instances**
ProductSubCategory	Mountain Bikes
DealerPrice	323 – 470
	470 – 1378
	1378 – 2040
ModelName	Mountain-200
	Mountain-100
	Mountain-400-W
	Mountain-500
Color	Black
	Sliver

Table 2: Selected Attributes and Attribute Instances

1 Overstock	2 Tire
3 Sport100	4 October
5 fernando35@adventure-works.com	6 Bolts
7 Europe	8 Australia
9 Bachelors	10 Blade
11 Mountain Tire	12 Flat Washer
13 Internal Lock	14 California US
15 Brakes Chains	16 Road Bikes
17 Blade California	18 Chainring Bikes
19 Keyed Washer	20 Silver Hub
21 2001 January US	22 Caps Gloves Jerseys
23 HalfPrice Pedal Sale	24 Sydney Helmet Discount
25 Sydney California Promotion	26 Discount California December
27 Mountain Bike Socks	28 Cycling Cap Alexandria
29 HL Road Frame	30 Ithaca Accessories Clothing
31 New South Wales Professional	32 San Jose Metal Plate
33 Washington Tires Tubes	34 Germany US Dollar 2000
35 California Accessories 2001 September	36 Bikes Components Clothing Accessories
37 Central Valley Torrance Denver	38 Black Yellow handcrafted bumps
39 ML Fork North America	40 Central United States HeadSet
41 Allpurpose bar for on or off-road	42 December November Mountain Tire Sale
43 US 2001 2002 2003 2004	44 Seattle Saddles 1245550139
45 San Francisco Palo Alto Santa Cruz	46 7800 Corrinne Court Sunday
47 North America Europe Pacific Bikes 2003	48 Sealed cartridge Horquilla GM
49 LL Mountain Front Wheel US	50 Headlights Dual-Beam Weatherproof

Table 3: 50 keyword queries for the AW_ONLINE database

6.3 Evaluation of Subspace Ranking algorithm

In this section, we demonstrate the effectiveness of our star net ranking algorithms. We have manually written 50 keyword queries for the Adventureworks Internet Sales database (Table 3). The 50 keyword queries selected are evenly distributed in terms of the number of keywords contained. We evaluate four ranking methods. The standard method is the one that we proposed in Section 3.4. For comparison reasons, we introduce two additional methods that disable either the hit group number normalization or the group size normalization. The baseline method is similar to the one used in [15] where the scores from the full-text engine are directly averaged.

Figure 4 shows the results. The x-axis represents the rank of results and the y-axis corresponds to the percentage of the queries satisfied. A point (x, y) on the curve, represents that the most relevant star nets of y percent of the queries, can be found in the top-x result list. The relevance is checked manually for all the 50 queries. As we can see, for 47 query cases (94%), our standard method returns the most relevant star net in the Top 1, and overall, our method managed to reveal the relevant star nets in the Top 5. The standard method significantly outperforms the baseline method as well as the method with no hit group number normalization. By manually examining the output, we also note that

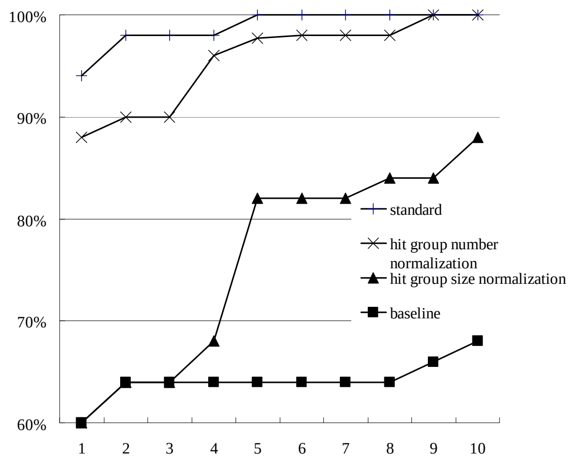


Figure 4: Evaluation of Four Star Net Ranking Methods

the worst case for our method (Top 5) is the query "Sydney Helmet Discount" in which "Sydney" happens to perfectly match the first name of a single customer. On the other hand, the method with no hit group size normalization did surprisingly well, with 88% of the relevant star nets ranked first. This implies, that the group size normalization does not play an important role in the ranking formula. On the other hand, the group number normalization is much more significant.

We applied the same evaluation process over the Reseller Sales (AW_Reseller) database. In this case, the queries for the AW_Reseller database consisted of keywords extracted from dimensions that were not used by the AW_Online fact table, like the Reseller dimension and the Employee dimension. The results are almost identical to what we have shown in Figure 4.

6.4 Effects of Bucket Number in Group-by Attribute Ranking

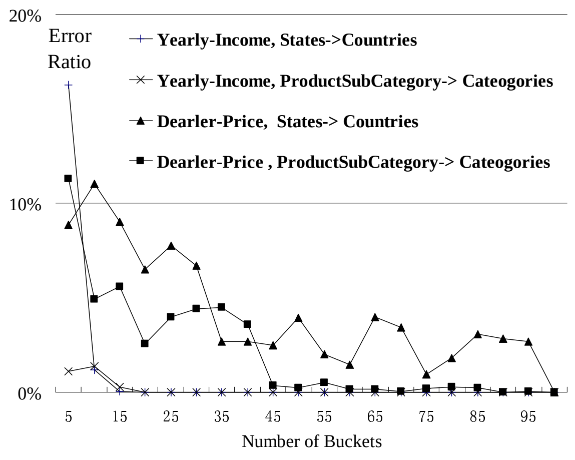


Figure 5: Bucket Number v.s. Group-by Attribute Scores (AW_ONLINE)

In this section we evaluate our approach on partitioning numerical grouping candidates. More specifically, we empirically test the assumption made in Section 4.2, that with a "sufficiently" large number of basic intervals, the actual correlation value can be captured.

We are interested in the effects of the number of buckets on the grouping quality. For these experiments, we use both databases AW_ONLINE and AW_RESELLER. In the AW_Online database (Figure 5), the numerical attributes are the `Yearly-Income` column from the `Customer` table and the `Dealer Price` attribute of the Product table. The roll-up operations are: from StateProvinces to Countries, and from `Product Subcategory` to `Product Category`. Therefore, Figure 5 shows 4 lines.

In the AW_Reseller database (Figure 6), we use a roll-up scenario from the product subcategory to the more general categories and select the following three numerical attributes as the candidate group-by attributes: `AnnualSales`, `AnnualRevenue` of the Reseller table and the `NumberOfEmployees`.

The x-axis in these figures represent the number of buckets used to partition the space and the y-axis show the error percentage. The error percentage is computed by the difference between the computed correlated value with the ground truth value. We calculate the ground truth value by dividing the attribute domain into smallest intervals such that each distinct value from the subspace has its own bucket. For each roll-up case, we run the experiment once and the average over all the roll-up cases is reported. For example, there are 81 Product subcategories to Category mappings, and therefore our algorithm is evaluated 81 times.

According to both figures, the error ratio generally decreases rapidly with the increase of the bucket number and tend to "converge" beyond 80 buckets. Most error ratio values are reduced to less than 5 percent with 40 "basic intervals". This fact supports our previous assumption, and we set the default system value of the number of buckets to 40.

6.5 Study of Numerical Partitioning Methods

In this section, we evaluate the performance of our interval merge methods. The interval merge happens after the basic interval has been fetched from the DBMS. We measure the quality of a merged partition with the error per-

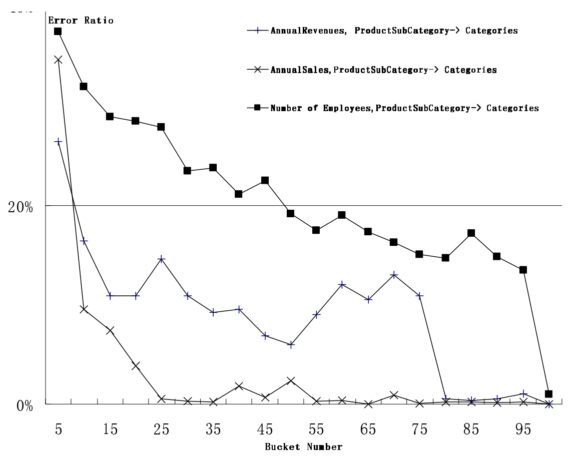
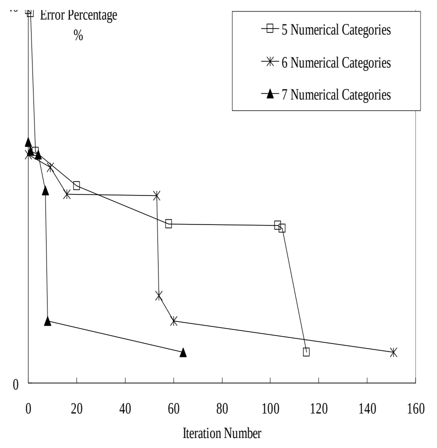
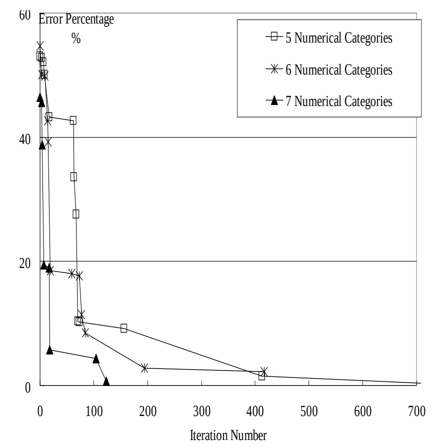


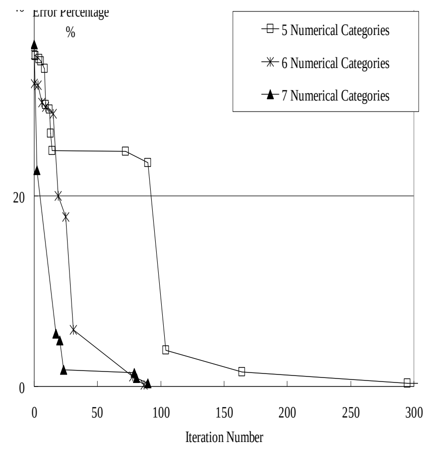
Figure 6: Bucket Number v.s. Group-by Attribute Scores (AW_RESELLER)



(a) Query: "France Clothing", Attribute Domain: `Yearly Income`



(b) Query: "France Accessories", Attribute Domain: `Yearly Income`



(c) Query: "British Columbia", Attribute Domain: `Number of Employees`

Figure 7: Exclusive Dominance Region Examples

centage (y-axis), which measures the difference between the correlation value computed over the basic intervals and the value over the merged intervals. Since our algorithm discussed in Section 5 is iterative in nature, the x-axis stands for the increase of iteration rounds. We use both databases, AW_RESELLER and AW_ONLINE, in this experiment. Each figure consists of three experiments by varying the target number of merged intervals from 5 to 7. There is one subspace associated that is defined by a keyword query and an numerical attribute domain that we try to partition into numerical categories. For example, in Figure 8(a), the query is "France Clothing" and the attribute domain for merging is "Yearly Income".

As we can see, in all three figures, the difference between the attribute score on the merged intervals and the basic intervals can be reduced dramatically as the iteration advances. With 100 iterations, the algorithm can discover partitions that are almost as good as the basic interval partition in most cases. On the other hand, we also observe that the number of targeted intervals has an impact on the convergence rate. Specifically, with a smaller number of targeted intervals, it takes longer to find an optimal partition.

We also want to point out that the simulated annealing algorithm is very efficient since none of the iterations require DBMS access and that at each step, all the operations incurred are main-memory array manipulations. For example, a 500 iterations interval merge operation takes less than 5 milliseconds.

7. CONCLUSIONS AND FUTURE WORK

We presented our novel KDAP approach for integrating the simplicity of keyword search with the efficient aggregation power of OLAP. KDAP provides an efficient and easy-to-use solution for business analysts who are faced with an overwhelming amount of dynamic and detailed data, that is usually not organized in a way that the business analyst thinks about the data. We have described our complete framework, which addresses many challenges raised by KDAP. We introduced several novel algorithms that make KDAP practical, like the candidate subspace generation, the ranking of the candidates and the dynamic facet construction. Finally, we demonstrated the generality, applicability and extensibility of our system with two different OLAP applications: (a) finding exceptions and surprises in the data and (b) locating local regions that are very correlated with global aggregates.

We also identify ample scope for future work. For example, our current model does not consider measure attributes as hit candidates; it is interesting to investigate how we can incorporate such measure in the KDAP model. Our simulated annealing solution for merging numerical intervals has been shown to be effective, but we hypothesize the existence of more efficient algorithms for finding partitions. Optimizing the human computer interaction is also very interesting. While our facet construction solutions are useful for explorations, they may become inadequate whenever the users have a very concrete goal for their aggregations. In such cases, the *consistency* of the interface organization becomes critical and a hybrid solution may be better. Finally, our current implementation requires aggregation over the sub-dataspace associated with a given keyword query. This can be quite expensive on sizable data warehouses. We plan to leverage the optimization power of existing OLAP engines and to develop new specialized techniques optimized for KDAP.

8. REFERENCES

[1] Flamenco faceted search system. http://flamenco.berkeley.edu/.

[2] Google trends. http://www.google.com/trends.

[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[4] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Authority-based keyword queries in databases using objectrank. In *VLDB*, 2004.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, 2002.

[6] S. Chaudhuri, G. Das, , and V. Narasayya. Dbexplorer: A system for keyword search over relational databases. In *ICDE*, 2002.

[7] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum.

Probabilistic ranking of database query results. In *VLDB*, 2004.

[8] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, March 1997.

[9] B. Chen, R. Ramakrishnan, J. Shavlik, and P. Tamma. Bellwether analysis: Predicting global aggregates from local regions. In *VLDB*, 2006.

[10] I. M. D. Florescu and D. Kossmann. Integrating keyword search into xml query processing. In *WWW*, 2000.

[11] W. Dakka, R. Dayal, and P. G. Ipeirotis. Automatic discovery of useful facet terms. In *SIGIR Faceted Search Workshop*, 2006.

[12] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results.

[13] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD*, 2003.

[14] M. Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, April 2006.

[15] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, 2003.

[16] V. Hristidis and Y. Papakonstantinou. Discover:keyword search in relational databases. In *VLDB*, 2002.

[17] Y. Li, C. Yu, and H.V.Jagadish. Schema-free xquery. In *VLDB*, 2004.

[18] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[19] S. Sarawagi. Explaining differences in multidimensional aggregates. In *VLDB*, 1999.

[20] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, 2000.

[21] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.

[22] D. Tunkelang. Dynamic category sets: An approach for faceted search. In *SIGIR Faceted Search Workshop*, 2006.

[23] K.-P. Yee. Faceted metadata for image search and browsing. In *CHI*, 2003.