# SeeDB: Visualizing Database Queries Efficiently

Aditya Parameswaran
Stanford University & UIUC
adityagp@illinois.edu

Neoklis Polyzotis
Google & UCSC
alkis@cs.ucsc.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

## ABSTRACT

Data scientists rely on visualizations to interpret the data returned by queries, but finding the right visualization remains a manual task that is often laborious. We propose a DBMS that partially automates the task of finding the right visualizations for a query. In a nutshell, given an input query Q, the new DBMS optimizer will explore not only the space of physical plans for Q, but also the space of possible visualizations for the results of Q. The output will comprise a recommendation of potentially "interesting" or "useful" visualizations, where each visualization is coupled with a suitable query execution plan. We discuss the technical challenges in building this system and outline an agenda for future research.

## 1. INTRODUCTION

*... today's researchers must consume ever higher volumes of numbers that gush, as if from a fire hose ...*

— R.M. Friedhoff and T. Kiely

Data analysts must sift through huge volumes of data looking for valuable data-specific insights, trends, or anomalies. This process involves selecting the "right" subset of the data, and the "right" way to view it, so that the "insights" become apparent. Moreover, the process is often ad-hoc and consumes a lot of the analyst's time. Our vision is that *some especially cumbersome aspects* of this search for interesting insights can be automated.

To illustrate, consider the following interactive exploration workflow, which we believe is often used in practice.

*Step (1):* First, the analyst poses a relational query to extract some subset of data they are interested in exploring. For example, the analyst may select all records associated with "stapler" products.

*Step (2):* Then, the analyst considers several candidate views over this subset of data, formed by, say, aggregation and grouping; the analyst must study all of these views one by one. For example, one view may be total stapler sales by year, while another view may be the quantity in stock by sales region. Since these views have two-attributes each, we can view them as 2-dimensional graphs. For example, Figure 1(a) may be the stapler sales (y-axis) by year (x-axis), while Figure 1(c) may be the quantity (y-axis) by region (x-axis). (Figures 1(b) and 1(d) are discussed below.)


*Proceedings of the VLDB Endowment,* Vol. 7, No. 4


*Step (3):* Next, the analyst steps through each view, and decides which views are "interesting." This of course is the critical and time-consuming step. What makes a view like Figure 1(a) interesting or not? Well, it all depends on the application semantics and what we are comparing against. For example, Figure 1(a) shows decreasing sales over time. If we are in a recession and all product sales are down then this observation is not very interesting. However, say that Figure 1(b) shows the aggregate (all) product sales over the same time periods. Then the stapler sales view goes against the general trend: overall sales are up, but stapler sales are going down. In this case, the view is *potentially* "interesting" because it depicts a trend in the subset of data that the analyst is interested in (i.e., stapler-related data) that *deviates* from the trend in the overall data. Of course, the analyst must decide if this deviation is truly an insight for this application. Even so, our key insight is that we may be able to *identify and highlight to the analyst potentially interesting views using automated mechanisms based on deviation*. By doing so, we eliminate the laborious process of stepping through all possible views that the analyst currently performs. Once we recommend potentially interesting views, we can let the analyst make the final decision.

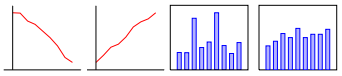


*Figure 1: Views (a), (b): Sales over Time. (c), (d): Quantity by Region.*

Figures 1(c) and 1(d) illustrate a different type of deviation. The first figure shows the distribution of staplers across regions, while the second figure shows the overall product distribution. Again, the stapler view does not follow the general trend: the regions that have the most staplers are not the larger regions that have most product in stock. The analysis must decide if this observation is interesting: perhaps the region that has many staplers is near the world-famous stapler-gun-wrestling contest, in which case the observation is expected. But perhaps there is a problem with the product shipping strategy, in which case the deviation is very important.

In this vision paper, we sketch our design for a new data base management system (DBMS), SeeDB, that automates the especially laborious aspects of the search for useful data insights. Figure 2 depicts SeeDB, together with a conventional DBMS. In the conventional system, the user or analyst submits a query $Q$ and obtains data subsets. Thus, conventional systems do not provide any means for the analyst to get intuitive visual insights directly. In SeeDB, the analyst also submits a query, but instead automatically obtains views (or visualizations) of the query result that are potentially of interest. As illustrated in our examples, these visualizations help the analyst quickly interpret and understand specific "interesting/useful" aspects of the query result. Thus, with SeeDB, we fundamentally modify the query-result paradigm of databases: SeeDB is provided a query $Q$, and outputs visualizations of interesting aspects of $Q$.

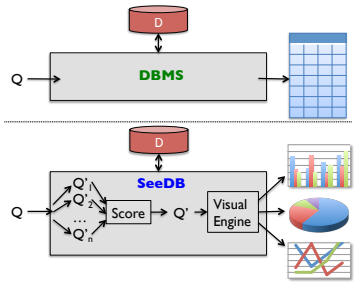


*Figure 2: SeeDB Comparison with regular DBMS (The workflow for SeeDB is only conceptual and need not happen in that order.)*

Given a database and a query $Q$, SeeDB considers a space of views $Q'_1, \ldots, Q'_n$ that are generated by adding to $Q$ additional relational operators, such that the results can be readily visualized (via the visual engine, depicted in Figure 2). For instance, one possibility in our Staplers example was to add a group by and an aggregation giving a view corresponding to total sales of Staplers per region. We call these views $Q'_1, \ldots, Q'_n$ the *discriminating views*. To decide if a discriminating view is interesting, SeeDB compares it to the equivalent view obtained from the full database, using the same operators. For comparing the two views, SeeDB uses a set of functions that captures how much the first view *deviates* from the second. Prior work in the visualization community has identified several functions for this purpose; however an important issue is the feasibility of computing such functions on large databases. These functions capture the deviations illustrated in our initial example (e.g., different slopes), and draw from existing functions that compare value distributions (e.g., earth movers function). Of course, SeeDB is not tied to any particular function(s), and allows the analyst to override the defaults using their own functions. We say that SeeDB computes the *utility* of discriminating view when it compares the discriminating view against the same view on the database.

Our next example is a more detailed version of the previous example, and clarifies the terms introduced.

Example 1.1. *We focus on a database with a traditional star schema. We operate on a single fact table D, containing information about sales. The schema of D comprises three dimension attributes: Product, Location, and Year, and one measure attribute: Sales.*

*Let us assume our analyst has entered a query with a single selection predicate: $Q \equiv \sigma_{(Product\ =\ Staplers)}$. The result contains too many tuples to examine individually, and hence the analyst has to rely on some appropriate visualization in order to glean interesting insights about the overall query result.*

SeeDB *searches over all possible discriminating views that can be obtained by adding a single aggregate and group by operator. We initially focus on these two-attribute discriminating views because they are easy to visualize using histograms or line plots. (*SeeDB *also considers more general views.) One of these queries is $Q'_1 = R_1(Q)$ where $R_1 \equiv \gamma_{Location,\ sum(Sales)}$. This query tracks the sum of Sales over Location. A possible result, $R_1(Q(D))$, is the discriminating view shown in the top part of Table 2. Another possible query is $Q'_2 = R_2(Q)$, where $R_2 \equiv \gamma_{Year,\ sum(Sales)}$, tracking the sum of Sales over Year. The bottom part of Table 2 shows a possible result of this second view.*

*The next step is to score each view based on its utility, i.e., its ability to show an interesting property of the query result. For this purpose,* SeeDB *obtains aggregate statistics for Sales for Location and Year for the original full database. (As we will see later, there are interesting optimization opportunities if we can integrate this step with the processing of Q.) Table 1 shows the full database aggregates, i.e., $R_1(D)$ and $R_2(D)$ corresponding to our sample views. (Note the missing Q.) For the Location attribute, both $R_1(D)$ and $R_1(Q(D))$ have similar distributions. (The fact that sales are uniformly lower in $R_1(Q(D))$ is not surprising since $R_1(Q(D))$ only considers a fraction of the data.)*

| Location Aggregates: $R_1(Q(D))$ | | | |
|---|---|---|---|
| Boston: 30 | Seattle: 40 | New York: 40 | San Francisco: 90 |
| Year Aggregates: $R_2(Q(D))$ | | | |
| 2009: 50 | 2010: 40 | 2011: 60 | 2012: 50 |

*Table 1: Aggregates for Product = 'Staplers'*

| Location Aggregates: $R_1(D)$ | | | |
|---|---|---|---|
| Boston: 300 | Seattle: 300 | New York: 300 | San Francisco: 700 |
| Year Aggregates: $R_2(D)$ | | | |
| 2009: 100 | 2010: 200 | 2011: 500 | 2012: 800 |

*Table 2: Original Aggregates*

*However, notice that the distribution across Years is very different for tuples satisfying Product = 'Staplers'. That is, demand for 'Staplers' seems to have not gone up, unlike the other products. This unexpected behavior will be detected by* SeeDB *when it computes the utility of $R_2(Q)$, and hence $R_2(Q)$ (and not $R_1(Q)$) will be suggested to the analyst for further human evaluation.*

Thus, there are several technical challenges that need to be addressed:

- For a given query, $n$, the total number of discriminating views, is likely to be very large to explore exhaustively and precisely. Even if we restrict ourselves to views that append a group-by and an aggregation, the number of choices depends on the number of aggregation methods and group-by attributes. Generating each of $R_1(Q(D)), \ldots, R_n(Q(D))$, scoring them on utility, and then picking the best one is certainly not feasible for most databases. Thus, we need mechanisms to prune the space of views and compute their utility approximately. This approach is reminiscent of how a query optimizer costs and prunes candidate execution plans, except that the objectives are different and hence may require different cost models and data statistics. In addition, given that the end result is consumed by an analyst, it may be preferable to recommend a visualization with lower utility but also lower cost to generate. This option creates a bi-criterion optimization problem, where possible visualizations may trade off between utility and cost.
- Generating and scoring the discriminating views $R_i(Q(D))$ one-by-one may miss interesting optimization opportunities: First, we may share computation between discriminating views. For example, the results of two views with different aggregates but the same group-by may be computed together in one query, followed by projecting out to reveal the two individual views. Second, by evaluating the discriminating views in a deliberate order, we may be able to prune views with low utility (without evaluation) that are definitely not going to be recommended to the analyst.
- Since visualizations tend to convey approximate information, e.g., a trend in a line plot may be more important than knowing the exact coordinates of each point, we can introduce approximations as part of SeeDB. Thus, the utility of a discriminating view may be computed approximately but efficiently, and the recommended discriminating views can be populated with approximate results, based on synopses of the base data or of the query result, that can be generated much more efficiently.

Over the past few years, there has been a significant effort from the visualization community to provide interactive tools for data analysts. In particular, tools such as ShowMe, Polaris, and Tableau [16, 10] provide a canvas for data analysts to manipulate and view data, tools such as Wrangler [7] allow data analysts to transform and clean data, and tools such as Profiler [8] allow users to visualize simple anomalies in data. However, unlike SeeDB, these tools have little automation; in effect, it is up to the analyst to generate a two-column result (like the result of the discriminating view) to be visualized.

In this paper, we present our goals formally, and then present our initial design for SeeDB, along with the underlying challenges.

## 2. SYSTEM OBJECTIVES

We now state the goals of SeeDB more formally, to provide a blueprint for the system design sections that follow. When doing so, we deliberately focus on a simple setting to ground our discussion.

However, the setting we consider is an important use-case that occurs often in practice, and was the focus of our illustrative example. We will consider advanced variants in Section 4.

**Concrete Goals:** We consider a database $D$ with a snowflake schema, with dimension attributes $A$, measure attributes $M$, and potential aggregate functions $F$ over the measure attributes. We limit the class $\mathcal{Q}$, of queries posed over $D$, to be those that select a horizontal fragment of the fact table. The selection of the fragment can be done with selection predicates on the fact table, or on dimension tables through key/foreign-key joins. Intuitively, the idea is that the analyst specifies their interest in examining facts that satisfy specific conditions. SeeDB will identify visualizations that show some interesting properties of these facts.

Given a query $Q$ in $\mathcal{Q}$, we define $\mathcal{R}_Q$, the set of all discriminating views, to be the set of views that perform a group-by and some aggregation over the results of $Q$. For simplicity, we assume that a discriminating view $R$ in $\mathcal{R}_Q$ performs a group-by on a single attribute $a \in A$, and applies an aggregation function $f \in F$ on a single measure attribute $m \in M$. A view in this class corresponds to a two-column table that shows how the value of $f(m)$ varies with values of attribute $a$. This table can be directly visualized using a histogram, a bar chart or a line plot. (We consider generalizations in Section 4.)

We also assume the existence of a function $U(R)$ that can characterize the utility of each view $R(Q)$ in $\mathcal{R}_Q$ (higher is better). For now, we focus on picking discriminating views that optimize $U(R)$ with latency as low as possible: we return to more general objectives in Section 4. Thus, our concrete goal is:

GOAL 2.1. *Given $Q \in \mathcal{Q}$ and a positive integer $K$, find $K$ discriminating views $R_i \in \mathcal{R}_Q$, such that the $R_i$ have the largest values of $U(R_i)$ among those in $\mathcal{R}_Q$, and the total latency is minimized.*

**Operationalizing Utility:** One of the key challenges behind SeeDB is formalizing the utility function $U(R)$ for a discriminating view $R$. There are many choices for $U$ and we expect SeeDB to recommend views that score high on several metrics. As discussed previously, the proposed metric tries to capture the idea of "deviation" between distributions, i.e., a view has high utility if its contents show a trend that deviates from the corresponding trend in the original database.

We first define some notation. For any discriminating view $R_i$ in the class defined above, we note that $R_i(D)$ and $R_i(Q(D))$ are both two column tables. A two-column table can be represented using a weight vector. We let the weight vector $W_{a,f(m)}$ represent the result of $R_i(D) = \gamma_{a,f(m)}(D)$, i.e., distribution of the aggregate function $f$ on the measure quantity $m$ across various values of the attribute $a$. Going back to our example, it follows that

$$W_{\text{Year, sum(Sales)}} = (2009 : 100, 2010 : 200, 2011 : 500, 2012 : 800)$$

Here $m$ is Sales, $f$ is the sum, and $a$ is the Year attribute. Then, we let $W^Q_{a,f(m)}$ represent the (changed) distribution of $R_i(Q(D))$, the aggregated quantity $m$ across values of the attribute $a$, when restricted to the result of the query $Q$. Thus,

$$W^{\sigma_{\text{Prod = 'Staplers'}}}_{\text{Year, sum(Sales)}} = (2009 : 50, 2010 : 40, 2011 : 60, 2012 : 50)$$

The utility $U$ of a discriminating view $\gamma_{a,f(m)}$ is defined to be the distance between $W^Q_{a,f(m)}$, and $W_{a,f(m)}$: $U(\gamma_{a,f(m)}) = S(W^Q_{a,f(m)}, W_{a,f(m)})$ where $S$ is a distance metric. The higher $S$ is, the more useful a discriminating view is. Common distance metrics used in visualization literature include K-L divergence [19], Jenson-Shannon distance [18, 17], and earth mover distance [20]. Wang [17] provides a good overview of the metrics used in scientific visualizations, while [20] provides a summary of probability-based distance metrics. As discussed earlier, we do not prescribe any specific distance metrics, instead, we plan to support a whole range of distance metrics, which can be overridden by the data analyst.

We show in our extended paper [11] that in our example, we get the right discriminating view (i.e., that of Year) when we consider the metrics mentioned above: the utility for Year is significantly higher than that for Location for each of the metrics considered.

# 3. INITIAL DESIGN

Our initial design for SeeDB is as a simple wrapper over an existing database system. One straightforward workflow is as follows:

- *Step 1:* Enumerate all discriminating views $R \in \mathcal{R}_Q$ and evaluate utility $U(R)$ (i.e., compute $W^Q_{a,f(m)}$ and $W_{a,f(m)}$) by issuing the corresponding counting queries to the DBMS. Select the $K$ views with highest $U(R)$.
- *Step 2:* Compute the results of these $K$ views using the DBMS, then forward the results to the visual engine.

It is clear that this workflow suffers from several inefficiencies. We now discuss potential optimizations, some of which require new query processing schemes specialized for the problem at hand.

**Approximate Utility Computation:** We can speed up Step 1 by computing the utilities of discriminating views approximately. Naturally, we would want the approximations to be accurate enough to select a good set of views for Step 2.

Sampling is one possible approximation method: We construct a sample of the query result $Q(D)$, and the underlying data $D$, and use these samples to compute approximate weight vectors $W^Q_{a,f(m)}$ and $W_{a,f(m)}$. (In fact, the latter, which does not depend on $Q$, can be computed before any queries are issued.) The question of how large a sample of $Q$ is necessary to enable the weight vectors $W^Q_{a,f(m)}$, for all $a, f, m$, to have high accuracy is, to the best of our knowledge, still open. Techniques from sampling for aggregation [3], and more generally, approximate query processing [1, 2] may be relevant here.

Ideally, we would want error bounds on the resulting approximate utilities, in order to enable SeeDB to select views of provably high utility and avoid views of provably low utility. These guarantees may depend on the specific metric used. For instance, approximation guarantees on the magnitudes of the weight vectors may directly translate to guarantees on the earth mover distance (a simpler metric), but not the Jenson-Shannon distance (a complex metric).

**Searching the Space of Discriminating Views:** The space of discriminating views may be too large to search exhaustively in an efficient manner, particularly if SeeDB relies on exact utility computation. Instead, it may be possible to prune the search space by leveraging relationships between discriminating views in terms of utility. For instance, functional dependencies among grouping attributes can help us infer that certain views will have the exact same utility by virtue of having the same groups. Furthermore, the search strategy that navigates the space of views may also take into account inter-dependencies: for instance, investing computational resources to determine that a view has provably low utility would be useful, if this determination will lead to the pruning of several other views correlated with the specific view.

**Multi-Query Optimization:** Step 2 comprises the evaluation of $K$ queries and so raises opportunities for multi-query optimization [15]. For instance, if we are recommending several discriminating views with the same group-by attribute, we may combine the computation of the views into a single group-by query with multiple aggregations (one per view). We expect that the opportunities to share computation will increase with more complex queries and views.

Multi-query optimization may also be used to optimize utility computation (Step 1). For instance, multi-query optimization may reveal that $W^Q_{a,f_i(m)}$ may be computed together for all $i$. Additionally, since we make repeated calls to evaluate $W^Q_{a,f(m)}$ for different $a, f, m$, we can instead first materialize the query result $Q(D)$ and

then compute the weight vectors $W^Q_{a,f(m)}$ by issuing queries on the materialized result. Materializing $Q(D)$ can also help in evaluating $R(Q(D))$ for each selected discriminating view $R$ in Step 2.

**Fusing the Two Steps:** Up to this point we considered view selection and view computation as two separate stages inside SeeDB. Alternatively, we may fuse the two stages in order to share work between the computation of utilities (Step 1) and the evaluation of view queries (Step 2), thus reducing end-to-end latency.

Computing $U(R)$ requires knowledge about the contents of $R$, and therefore, we compute $U(R)$ and $R(Q(D))$ together. Since it may be prohibitively expensive to compute these quantities for each $R$, and since the end goal is to recommend only $K$ views, we may employ approximate utility computation coupled with a pruning rule. Specifically, suppose that is it possible to schedule the computation of $Q(D)$ and receive its output in a random order (e.g., as described in [6]). SeeDB will then observe a sample of increasing size as it consumes the output of $Q(D)$. Processing the output involves two tasks: (a) updating a running estimate of the utility of each view (leveraging the fact that the observed output is a sample of $Q(D)$), and (b) updating the current contents of unpruned views using hash-based aggregation. SeeDB can use the running utility estimates to prune views of low utility. Overall, as SeeDB processes $Q(D)$ it can make progress towards both selecting the top-$K$ views and computing their contents. Thus, this specialized query-processing strategy can reduce SeeDB's end-to-end latency but it comes with higher resource requirements (since many group-by queries need to be processed concurrently).

# 4. ENHANCEMENTS AND EXTENSIONS

We now present some enhancements that may further improve the analyst's user experience beyond those suggested previously.

**Reducing Perceived Latency:** To reduce perceived latency, SeeDB can first produce the discriminating view result in the top-K that takes the least amount of time to execute. That way, the user can peruse the first visualization as soon as possible. Then, SeeDB can generate the remaining views. Further, SeeDB can, in the background, compute the result of the current top-$K$ views while considering other views. If a view is no longer in the current top-K, then it is replaced with another (better) view, which begins executing.

**Latency Threshold:** We may wish to incorporate a user-specified overall latency threshold in our goal (i.e., find top-$K$views such that total latency is bounded), so that the analyst does not have to wait too long to see visualizations. Dealing with a latency threshold is certainly more challenging, and will require new techniques. One simple heuristic is to discard any views (without computation) whose cost is estimated to be large. Also, we may be able to leverage interdependencies between views for further pruning based on cost.

**General Settings:** SeeDB can be generalized to handle more elaborate discriminating views with little or no change. For instance, SeeDB can easily handle multiple group-by attributes, resulting in multi-column views that can be visualized as stacked bar-charts. Our discriminating views could include additional selection predicates (in addition to a group-by and an aggregation); for instance, in our staplers example, perhaps the trend line of total sales of staplers in California is an interesting visualization, because it differs from the total sales of staplers in the rest of the country. Overall, there is a rich space of general discriminating views we can consider.

**Refinement of Visualizations:** Since analysts are rarely interested in absolute values in their visualizations, we may be able to leverage ideas similar to those used in online aggregation [5] to produce visualizations that become more accurate over time.

**Selecting Diverse Views:** Our goal simply selects the K views with the highest utility, ignoring the fact that the discriminating views are related. For instance, an analyst may prefer one visualization each of sales and revenue, instead of two visualizations of sales, since the former covers more measure attributes. Incorporating such personalized preferences requires new models of discriminating view-diversity (leveraging metrics from recommendation systems) and hence new computation methods.

# 5. RELATED WORK

Previous work related to SeeDB falls under three themes: visualization tools (covered in Section 1), OLAP, and database visualizations:

**OLAP:** There has been some work on browsing data cubes, allowing analysts to variously find "explanations" for why two cube values were different, to find which neighboring cubes have similar properties to the cube under consideration, or get suggestions on what unexplored data cubes should be looked at next [12, 14, 13].

**Database Visualization Work:** Fusion tables [4] allows users to create visualizations layered on top of web databases; they do not consider the problem of automatic visualization generation. Devise [9] translated user-manipulated visualizations into database queries.

# 6. CONCLUSIONS

We outlined our vision for SeeDB: a system that provides analysts with visualizations highlighting interesting aspects of the query result. We defined several concrete problems in architecting SeeDB, relating to areas ranging from multi-query optimization and approximation to multi-criteria optimization. We believe that the general area of bringing visualizations closer to the DBMS is a challenging, yet, important direction for database research in the future.

# 7. REFERENCES


[1] K. Chakrabarti et al. Approximate query processing using wavelets. In *VLDB*, pages 111–122, 2000.

[2] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[3] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.

[4] H. Gonzalez et al. Google fusion tables: web-centered data management and collaboration. In *SIGMOD Conference*, pages 1061–1066, 2010.

[5] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In J. Peckham, editor, *SIGMOD 1997*, pages 171–182. ACM Press, 1997.

[6] C. Jermaine et al. Scalable approximate query processing with the dbo engine. *ACM Trans. Database Syst.*, 33(4), 2008.

[7] S. Kandel et al. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[8] S. Kandel et al. Profiler: integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554, 2012.

[9] M. Livny et al. Devise: Integrated querying and visualization of large datasets. In *SIGMOD Conference*, pages 301–312, 1997.

[10] J. D. Mackinlay et al. Show me: Automatic presentation for visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1137–1144, 2007.

[11] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. SeeDB: Visualizing Database Queries Efficiently. *Stanford Infolab*, 2013.

[12] S. Sarawagi. Explaining differences in multidimensional aggregates. In *VLDB*, pages 42–53, 1999.

[13] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.

[14] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, pages 531–540, 2001.

[15] T. K. Sellis. Multiple-query optimization. *ACM TODS*, 13(1):23–52, 1988.

[16] C. Stolte et al. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, 2008.

[17] C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy*, 13(1):254–273, 2011.

[18] Wikipedia. Jensen shannon divergence — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].

[19] Wikipedia. Kullback leibler divergence — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].

[20] Wikipedia. Statistical distance — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].