SeeDB: Automatically Generating Query Visualizations

ABSTRACT

Data analysts operating on large volumes of data often rely on visualizations to interpret the results of queries. However, finding the right visualization for a query is a laborious and time-consuming task. We propose SeeDB, a system that partially automates this task: given a query, SeeDB explores the space of all possible visualizations, and automatically identifies and recommends to the analyst those visualizations it finds to be most “interesting” or “useful”.

1. INTRODUCTION

Data analysts must sift through very large volumes of data to identify trends, insights, or anomalies. Given the scale of data, and the relative ease and intuitiveness of examining data visually, analysts often use visualizations as a tool to identify these trends, insights, and anomalies. However, selecting the “right” visualization often remains a laborious and time-consuming task.

We illustrate the data analysis process using an example. Consider a dataset containing sales records for a nation-wide chain of stores. Let’s say the store’s data analyst is interested in examining how the newly-introduced heating device, the “Laserwave Oven”, has been doing over the past year. The results of this analysis will inform business decisions for the chain, including marketing strategies, and the introduction of a similar “Saberwave Oven”.

The analysis workflow proceeds as follows: (1) The analyst poses a query to select the subset of data that she is interested in exploring. For instance, for the example above, she may issue the query:

Q = SELECT * FROM Sales WHERE Product = “Laserwave″

Notice that the results for this query may have (say) several million records each with several dozen attributes. Thus, directly perusing the query result is simply infeasible. (2) Next, the analyst studies various properties of the selected data by constructing diverse views or visualizations from the data. In this particular scenario, the analyst may want to study total sales by store, quantity in stock by region, or average profits by month. To construct these views, the analyst can use operations such as binning, grouping, and aggregation, and then generate visualizations from the view. For example, to generate the view ‘total sales by store’, the analyst would group each sales record based on the store where the sale took place and sum up the sale amounts per store. This operation can easily be expressed as the familiar aggregation over group-by query:

Q′ = SELECT store, SUM(amount) FROM Sales WHERE
Product = “Laserwave” GROUP BY store

The result of the above query is a two-column table that can then be visualized as a bar-chart. Table 1 and Figure 1 respectively show an example of the results of this view and the associated visualization. To explore the query results from different perspectives, the analyst generates a large number of views (and visualizations) of the form described above. (3) The analyst then manually examines each view and decides which ones are “interesting”. This is a critical and time-consuming step. Naturally, what makes a view interesting depends on the application semantics and the trend we are comparing against. For instance, the view of Laserwave sales by store, as shown in Figure 1, may be interesting if the overall sales of all products show the *opposite* trend (e.g. Figure 2). However, the same view may be uninteresting if the sales of all products follow a similar trend (Figure 3). Thus, we posit that a view is *potentially “interesting” if it shows a trend in the subset of data selected by the analyst (i.e., Laserwave product-related data) that deviates from the equivalent trend in the overall dataset*. Of course, the analyst must decide if this deviation is truly an insight for this application. (4) Once the analyst has identified interesting views, the analyst may then either share these views with others, further interact with the displayed views (e.g., by drilling down or rolling up), or start afresh with a new query.

Of the four steps in the workflow described above, the ones that are especially repetitive and tedious are steps (2) and (3), where the analyst generates a large number of candidate views, and examines each of them in turn. The goal of our system, SeeDB, is to automate these labor-intensive steps of the workflow. Given a query Q indicating the subset of data that the analyst is interested in, SeeDB automatically *identifies and highlights to the analyst the most interesting views of the query results using methods based on deviation*. Specifically, SeeDB explores the space of all possible views and measures how much each view deviates from the corresponding view on the entire underlying dataset (e.g. Figure 1 vs. Figures 2 or 3.) By generating and scoring potential views automatically, SeeDB effectively eliminates steps (2) and (3) that the analyst currently performs. Instead, once SeeDB recommends interesting views, the analyst can evaluate this small subset of views using domain knowledge and limit further exploration to these views.

Manasi: Add second example with medical data?

We described our vision for SeeDB, along with the associated research challenges in a companion vision paper [**?**]. In this demon-

Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 4
Copyright 2013 VLDB Endowment 2150-8097/13/12.

Table 1: Data: Total Sales by Store for Laserwave

Store	Total Sales ($)
Cambridge, MA	180.55
Seattle, WA	145.50
New York, NY	122.00
San Francisco, CA	90.13

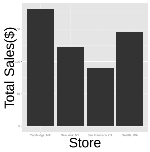


Figure 1: Visualization: Total Sales by Store for Laserwave

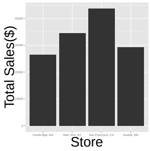


Figure 2: Scenario A: Total Sales by Store

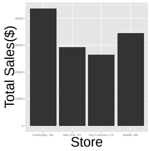


Figure 3: Scenario B: Total Sales by Store

stration proposal, we present our first SEEDB prototype addressing some of the challenges listed in that vision paper. In particular, our current prototype of SEEDB is built as a “wrapper” that can be overlaid on any relational database system. Given any query, SEEDB leverages special optimization algorithms and the underlying DBMS to generate and recommend interesting visualizations. To do so efficiently and accurately, we must address the following challenges: (a) We must determine metrics that accurately measure the “deviation” of a view with respect to the equivalent view on the entire database (e.g., Figure 1 vs. 2), while simultaneouly ensuring that SEEDB is not tied to any particular metric(s); (b) We must intelligently explore the space of candidate views. Since the number of candidate views (or visualizations) increases as the square of the number of attributes in a table (we will demonstrate this in subsequent sections), generating and evaluating all views, even for a moderately sized dataset (e.g. 1M rows, 100 attributes), can be prohibitively expensive; (c) While executing queries corresponding to different views, we must share computation as much as possible. For example, we can compute multiple views and measure their deviation all together in one query. Independent execution, on the other hand, will be expensive and wasteful; (d) Since analysis must happen in real-time, we must trade-off accuracy of visualizations or estimation of “interestingness” for reduced latency. Section 4 describes how we address these challenges.

Manasi: Add list of contributions

2. PROBLEM STATEMENT

Given a database D and a query Q, SEEDB considers a number of views that can be generated from Q by adding relational operators. For the purposes of this discussion, we will refer to views and visualizations interchangeably, since it is straightforward to translate views into visualizations automatically. For example, there are straightforward rules that dictate how the view in Table 1 can be transformed to give a visualization like Figure 1. Furthermore, we limit the set of candidate views to those that generate a two-column result via a single-attribute grouping and aggregation (e.g. Table 1). However, SEEDB techniques can directly be used to recommend visualizations for multiple column views (> 2 columns) that are generated via multi-attribute grouping and aggregation.

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 of queries Q posed over D to be those that select one or more rows from the fact table, and denote the results as D_Q.

Given such a query Q, SEEDB considers all views V_i that perform a single-attribute group-by and aggregation on D_Q. We represent V_i as a triple (a, m, f), where $m \in M, a \in A, f \in F$, i.e., the view performs a group-by on a and applies the aggregation function f on a measure attribute m. We call this the *target view*.

$$\texttt{SELECT}\ a, f(m)\ \texttt{FROM}\ D_Q\ \texttt{GROUP BY}\ a$$

As discussed in the previous section, SEEDB evaluates whether a view V_i is interesting by computing the deviation between the view applied to the selected data (i.e., D_Q) and the view applied to the entire database. The equivalent view on the entire database $V_i(D)$ can be expressed as shown below that we call the *comparison view*.

$$\texttt{SELECT}\ a, f(m)\ \texttt{FROM}\ D\ \texttt{GROUP BY}\ a$$

The results of both the above views are tables with two columns, namely a and $f(m)$. We normalize each result table into a probability distribution, such that the values of $f(m)$ sum to 1. For our example in Table 1, the probability distribution of $V_i(D_Q)$, denoted as $P[V_i(D_Q)]$, is: (Jan: 180.55/538.18, Feb: 145.50/538.18, March: 122.00/538.18, April: 90.13/538.18). A similar probability distribution can be derived for $P[V_i(D)]$.

Given a view V_i and probability distributions for the target view ($P[V_i(D_Q)]$) and comparison view ($P[V_i(D)]$), the *utility* of V_i is defined as the distance between these two probability distributions. Formally, if S is a distance function,

$$U(V_i) = S(P[V_i(D_Q)], P[V_i(D)])$$

The utility of a view is our measure for whether the target view is “potentially interesting” as compared to the comparison view: the higher the utility, the more the deviation from the comparison view, and the more likely the view is to be interesting. Computing distance between probability distributions has been well studied, and SEEDB supports a variety of metrics to compute utility, including Earth Movers Distance, Euclidean Distance, Kullback-Leibler (K-L) Divergence, and Jenson-Shannon Distance. In our demonstration, conference attendees can experiment with different distance metrics and examine how the choice of metric affects view quality. Finally, we note that while other definitions of the comparison views and utility metrics are possible, for our initial exploration into visualization recommendations, we chose to focus on the intuitive definitions above.

PROBLEM 2.1. *Given an analyst-specified query Q on a database D, a distance function S, and a positive integer k, find k views $V \equiv (a, m, f)$ that have the largest values of $U(V)$ among all the views that can be represented using a triple (a, m, f), while minimizing total computation time.*

Manasi: Define measure and dimension attributes?
Manasi: Define the various distance metrics?

2.1 Extensions

Manasi: Make this clearer There are two important variations of the problem that can be addressed exactly by the same techniques discussed in the paper. These are:

- **Group-by clauses with multiple dimension attributes**: It is straightforward to extend the SEEDB techniques to group-by clauses with multiple attributes. However, for ease of exposition and visualization, we limit the number of attributes in the group-by clause to one.
- **Comparison of two-queries**: Instead of comparing the views of the input query to the entire underlying dataset, it may be more appropriate to compare them to another subset of the data (e.g. sales of “Staplers” vs. sales of “Printers”). This simply involves replacing the dataset parameter D with a second query Q'. This variation is important since it can help users find interesting differences in data. Our techniques apply to this variation unchanged and we show experimental results for this variation.

3. STATE-OF-THE-ART APPROACHES

Related Work: Over the past few years, the research community has introduced a number of interactive data analytics tools such as ShowMe, Polaris, and Tableau [**?**, **?**] as well as tools like Profiler allow analysts to detect anomalies in data. Unlike SEEDB, which recommends visualizations automatically, the tools place the onus on the analyst to specify the visualization to be generated. Similar visualization specification tools have also been introduced by the database community, including Fusion Tables [**?**] and the Devise [**?**] toolkit. There has been some work on browsing data cubes in OLAP, allowing analysts to find explanations, get suggestions for next cubes to visit, or identify generalizations or patterns starting from a single cube [**?**, **?**, **?**]. While we may be able to reuse the metrics from that line of work, the same techniques will not directly apply to visualizations.

3.1 Viz papers and tools

3.2 Data Cubes

3.3 General ML and stats references

3.4 Multi-query optimization

4. SEEDB DESIGN

In this section, we present the SEEDB architecture, starting with an overview followed by a detailed discussion of its components.

4.1 SEEDB architecture overview

Our SEEDB prototype is designed as a layer on top of a traditional relational database system. While optimization opportunities are restricted by virtue of being outside the database, our design permits SEEDB to be used in conjunction with a variety of existing database systems. SEEDB is comprised of two parts: a frontend and a backend. The frontend is a "thin client" that is used to issue queries and display visualizations. The backend, in contrast, performs all the computation required to generate and select views to be recommended. Figure 4 depicts the architecture of our system.

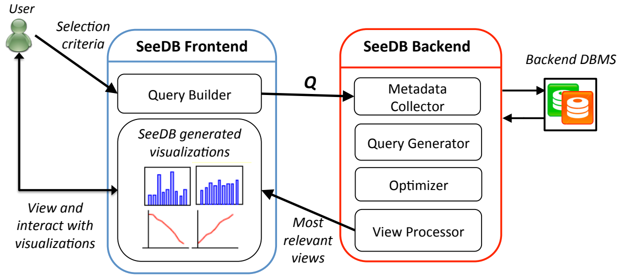


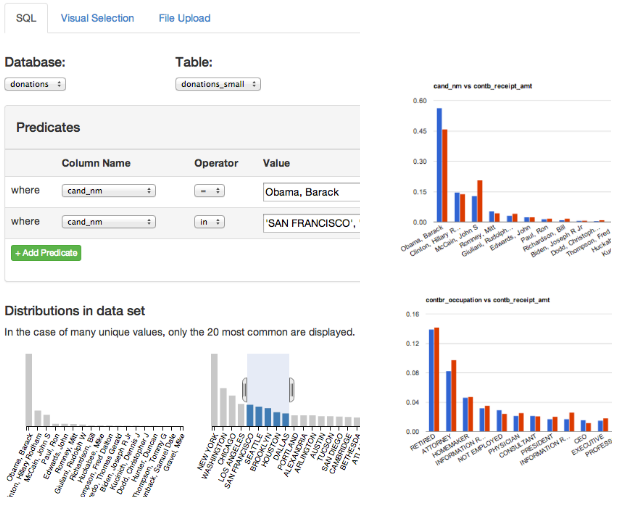
Figure 4: SeeDB Architecture

An analyst uses the frontend to issue queries to SEEDB. We provide three mechanisms for the analyst to issue queries (further discussion in Section 5). Once the analyst issues a query via the frontend, the backend takes over. First, the Metadata Collector module queries metadata tables (a combination of database-provided and SEEDB specific tables) for information such as table sizes, column types, data distribution, and table access patterns. The resulting metadata along with the analyst's query is then passed to the Query Generator module. The purpose of the Query Generator is twofold: first, it uses metadata to prune the space of candidate views to only retain the most promising ones; and second, it generates target and comparison views for each view that has not been pruned. The SQL queries corresponding to the target and comparison views are then passed to the Optimizer module. We refer to these queries collectively as *view queries*. Next, the Optimizer module determines the best way to combine view queries intelligently so that the total execution time is minimized. (We discuss optimizations performed by SEEDB in Section 6.) Once the Optimizer module has generated the optimized queries, SEEDB runs them on the underlying DBMS. Results of the optimized queries are processed by the View Processor in a streaming fashion to produce results for individual views. Individual view results are then normalized and the utility of each view is computed. Finally SEEDB selects the top k views with the highest utility and returns them to the SEEDB frontend. The frontend generates and displays visualizations for each of these view. We now discuss SEEDB modules in detail.

5. SEEDB FRONTEND

TThe SEEDB frontend, designed as a thin client, performs two main functions: it allows the analyst to issue a query to SEEDB, and it visualizes the results (views) produced by the SEEDB backend. To provide the analyst maximum flexibility in issuing queries, SEEDB provides the analyst with three mechanisms for specifying an input query: (a) directly filling in SQL into a text box, (b) using a query builder tool that allows analysts unfamiliar with SQL to formulate queries through a form-based interface, and (c) using pre-defined query templates which encode commonly performed operations, e.g., selecting outliers in a particular column.

Once the analyst issues a query via the SEEDB frontend, the backend evaluates various views and delivers the most interesting ones (based on utility) to the frontend. For each view delivered by the backend, the frontend creates a visualization based on parameters such as the data type (e.g. ordinal, numeric), number of distinct values, and semantics (e.g. geography vs. time series). The resulting set of visualizations is displayed to the analyst who can then easily examine these "most interesting" views at a glance, explore specific views in detail via drill-downs, and study metadata for each view (e.g. size of result, sample data, value with maximum change and other statistics). Figure 5 shows a screenshot of the SEEDB frontend (showing the query builder) in action.



Database: donations

Table: donations_small

Predicates

	Column Name	Operator	Value
where	cand_nm	=	Obama, Barack
where	cand_nm	in	'SAN FRANCISCO',

Distributions in data set

In the case of many unique values, only the 20 most common are displayed.

Figure 5: SeeDB Frontend: Query Builder (left) and Example Visualizations (right)

Manasi: Interactivity. Ship all data to client?

6. SEEDB BACKEND

$T(C_i)$	Data type for column C_i
$\lvert C_i \rvert$	Number of distinct values in C_i
$Var(C_i)$	Variance of values in C_i
$Corr(C_i, C_j)$	Correlation measure for all pairs of columns
$\mathcal{H}_{i\ldots k}$	Hierarchies between columns C_i to C_k
f_{C_i}, f_{C_i,C_j}	Frequency of access for each column and column pair

Table 2: Statistics and Table Metadata

The SeeDB backend is responsible for all the computations for generating and selecting views. To achieve its goal of finding the most interesting views accurately and efficiently, the SeeDB backend must not only accurately estimate the accuracy of a large number of views but also design ways in which the total processing time will be minimized. We first describe the basic SeeDB backend framework and then briefly discuss our optimizations.

Basic Framework: Given a user query Q, the basic approach computes all possible two-column views obtained by adding a single-attribute aggregate and group-by clause to Q. The target and comparison views corresponding to each view are then computed and each view query is executed independently on the DBMS. The query results for each view are normalized, and utility is computed as the distance between these two distributions (Section 2). Finally, the top-k views with the largest utility are chosen to be displayed. The basic approach is clearly inefficient since it examines every possible view and executes each view query independently. We next discuss how our optimizations fix these problems.

Manasi: put microbenchmark pictures here

View Space Pruning: In practice, most views for any query Q have low utility since the target view distribution is very similar to the comparison view distribution. SeeDB uses this property to aggressively prune view queries that are unlikely to have high utility. This pruning is based on metadata about the table including data distributions and access patterns. Manasi: fix following. taken from previous full paper Specifically, no expensive scans of the underlying tables are performed. In addition, we order the execution of view queries so that higher utility views can be computed before those with lower utility, thus permitting early stopping. For each table in the DBMS, we assume that statistics from Table 2 are available or can be computed cheaply. The data type for each column is numeric, categorical, ordinal, geographic, or date_or_time. The data type for column C_i, $T(C_i)$, along with the number of distinct values $|C_i|$ is used to determine whether the column will be treated as a *dimension* attribute or a *measure* attribute (Section **??**). As before, we denote the set of dimension attributes by $\mathcal{D}$ and measure by $\mathcal{M}$.

We employ the following heuristics for pruning and ordering views based on the statistics above.

- *Variance-based pruning*: Dimension attributes with low variance are likely to produce views having low utility (e.g. consider the extreme case where an attribute only takes a single value); SeeDB therefore prunes views with grouping attributes with low variance.
- *Correlated attributes*: If two dimension attributes a_i and a_j have a high degree of correlation (e.g. full name of airport and abbreviated name of airport), the views generated by grouping the table on a_i and a_j will be very similar (and have almost equal utility). We can therefore generate and evaluate a single view representing both a_i and a_j. SeeDB clusters attributes based on correlation and evaluates a representative view per cluster.
- *Bottom-up hierarchy traversal*: Manasi: fix. from full paper draft We observe that for a set of dimension attribute with a hierarchial structure, $H_{C_{i\ldots k}}$, if a view V at hierarchy level h has utility u, then views at hierarchy level $h-1$ will have utility $\leq u$. XXX: is this true for all utility functions?
- *Access frequency-based pruning*: In tables with a large number of attributes, only a small subset of attributes are relevant to the analyst and are therefore frequently accessed for data analysis. SeeDB tracks access patterns for each table to identify the most frequently accessed columns and combinations of columns. While creating views, SeeDB uses this information to prune attributes that are rarely accessed and are thus likely to be unimportant.

View Query Optimizations: The second set of optimizations used by SeeDB minimizes the execution time for view queries that haven't been pruned using the techniques described above. Since view queries tend to be very similar in structure (they differ in the aggregation attribute, grouping attribute or subset of data queried), SeeDB uses multiple techniques to intelligently combine view queries. The ultimate goal is to minimize scans of the underlying dataset by sharing as many table scans as possible. Our strategies include:

- *Combine target and comparison view query*: Since the target view and comparison views only differ in the subset of data that the query is executed on, we can easily rewrite these two view queries as one. This simple optimization halves the time required to compute the results for a single view. Manasi: Effect of indexes?
- *Combine Multiple Aggregates*: A large number of view queries have the same group-by attribute but different aggregation attributes. Therefore, SeeDB combines all view queries with the same group-by attribute into a single query. This rewriting provides a speed up linear in the number of aggregate attributes.
- *Combine Multiple Group-bys*: Since SeeDB computes a large number of group-bys, one significant optimization is to combine queries with different group-by attributes into a single query with multiple group-bys attributes. For instance, instead of executing queries for views (a_1, m_1, f_1), $(a_2, m_1, f_1) \ldots (a_n, m_1, f_1)$ independently, we can combine the n views into a single view represented by $(\{a_1, a_2 \ldots a_n\}, m_1, f_1)$ and post-process results at the backend. Alternatively, if the SQL GROUPING SETS functionality is available in the underlying DBMS, SeeDB can leverage that as well. While this optimization has the potential to significantly reduce query execution time, the number of views that can be combined depends on the correlation between values of grouping attributes and system parameters like the working memory. Given a set of candidate views, we model the problem of finding the optimal combinations of views as a variant of bin-packing and apply ILP techniques to obtain the best solution.
  Manasi: Bin packing forumation details
- *Sampling*: For datasets of large size, an optimization that affects performance significantly is employing sampling: we construct a sample of the dataset that can fit in memory and run all view queries against the sample. However, as expected, the sampling technique and size of the sample both affect view accuracy.
- *Parallel Query Execution*: The final optimization that SeeDB employs is taking advantage of parallel query execution at the DBMS to reduce total latency. We observe that as the number of queries executed in parallel increases, the total latency decreases at the cost of increased per query execution time (Manasi: discuss: thrashing).

Manasi: also from full paper draft It is possible to collect the above statistics at the dataset level too, as opposed to the entire table level. The advantage of table level statistics is that they have to be computed only once per table; however, dataset-level statistics are more accurate since they only consider the specific parts of the table. XXX: we use dataset-level statistics with table statistics do not result in aggressive pruning.

7. HANDLING ATTRIBUTES OF DIFFERENT KINDS

Manasi: Numerical, Temporal, Categorical, Gepgraphical
Manasi: Binning

8. USER STUDY: IS THIS USEFUL?

8.1 Comparison with other tools - Tableau, Spotfire

8.2 Lessons from different datasets

Manasi: Fix – copied form demo paper We propose to demonstrate the functionality of SEEDB through hands-on interaction with a variety of datasets. Our goals are two fold: (1) demonstrate the utility of SEEDB in surfacing interesting trends for a query and (2) demonstrate that we can return high quality views efficiently for a range of datasets. We will use four different datasets in our demonstration:

- **Store Orders dataset** [**?**]: This dataset is often used by Tableau [**?**] as a canonical dataset for business intelligence applications. It consists of information about orders placed in a store including products, prices, ship dates, geographical information, and profits. Interesting trends in this dataset have been very well studied, and attendees will use SEEDB to quickly re-identify these trends.
- **Election Contribution dataset** [**?**]: This is an example of a dataset typically analyzed by non-expert data analysts like journalists or historians. With this dataset, we demonstrate how non-experts can use SEEDB to quickly arrive at interesting visualizations.
- **Medical dataset** [**?**]: This real-world dataset exemplifies a dataset that a clinical researcher might use. The schema of the dataset is significantly complex and it is of larger size.
- **Synthetic data:** We provide a set of synthetic datasets with varying sizes, number of attributes, and data distributions to help attendees evaluate SEEDB performance on diverse datasets.

Scenario 1: Demonstrating Utility. Attendees are provided with three diverse, real-world datasets to explore using SEEDB. For each dataset, attendees can issue ad-hoc or pre-formulated queries to SEEDB. SEEDB will then intelligently explore the view space and optimize query execution to return the most interesting visualizations with low latency. Attendees can examine the returned queries visually, via the associated view metadata, and via drill-downs. To aid the evaluation of visualizations, the demo system will be configured to also show the user "bad" views (views with low utility) that were not selected by SEEDB. Similarly, we provide pre-selected queries (and previously known information about their results) to allow attendees to confirm that SEEDB does indeed reproduce known information about these queries. Attendees will also be able to experiment with a variety of distance metrics for computing utility and observe the effects on the resulting views.

Scenario 2: Demonstrating Performance and Optimizations. This scenario will use an enhanced user interface and synthetic datasets mentioned above. Attendees will be able to easily experiment with a range of synthetic datasets and input queries by adjusting various "knobs" such as data size, number of attributes, and data distribution. In addition, attendees will also be able to select the optimizations that SEEDB applies and observe the effect on response times and accuracy.

Thus, through our demonstration of SEEDB we seek to illustrate that (a) it is possible to automate labor-intensive parts of data analysis, (b) aggregate and grouping-based views are a powerful means to identify interesting trends in data, and (c) the right set of optimizations can enable real-time data analysis of large datasets.

9. PERF STUDY

Manasi: end-to-end testing with various parameter settings