Scalable Spreadsheets for Interactive Data Analysis

Vijayshankar Raman Andy Chou Joseph M. Hellerstein

University of California, Berkeley
{rshankar,achou,jmh}@cs.berkeley.edu

Abstract

Interactive responses and natural, intuitive controls are important for data analysis. We are building ABC, a scalable spreadsheet for data analysis that combines exploration, grouping, and aggregation. We focus on interactive responses in place of long delays, and intuitive, direct-manipulation operations in place of complex queries. ABC allows analysts to interactively explore the data at varying granularities of detail using a spreadsheet. Hypotheses that arise during the exploration can be checked by dynamically computing aggregates on interactively chosen groups of data items.

In this paper we describe our vision for ABC. We give examples that illustrate the need for interactivity in query processing and query formulation, the advantages of dynamic group formulation, and the usefulness of exploration in discovering hypotheses about data that can then be verified by aggregation. We briefly discuss the systems issues involved in building ABC, and mention our progress so far.

1 Introduction

Data analysis is a complex task. Users often do not issue a single, perfectly chosen query that extracts the "desired information" from a database; indeed the idea behind analysis is to extract heretofore unknown information, and in most situations there is no one perfect query [F$^+$96]. User studies have found that information seekers naturally work in an iterative fashion, starting by asking broad questions, and continually refining them based on feedback and domain knowledge (see for e.g., [OJ93]). Hence it is important that the interface to data analysis be interactive. Analysts should be given continual feedback for queries, and a simple, intuitive interface for specifying and refining them.

1.1 Problems with current data analysis methods

Lack of interactivity: Many current mechanisms for data analysis such as decision support queries and data mining are not designed for this continual mode of interaction with the analyst. They are optimized for a batch-style interaction where the user submits a query, waits a long time, and gets a complete, exact answer. The resulting long delays between successive queries disrupts the concentration of the analyst and hampers interactive analysis. Delays are an issue not only for aggregation tools, but also for exploration tools such as spreadsheets. Although spreadsheets provide interactive responses to "point-and-click" operations like sorting and scrolling, they do this by sacrificing scalability. For example, MS Excel has an upper limit of 65536 rows. Likewise, OLAP tools provide fast responses by restricting queries to those that can be answered using precomputed aggregates.

Difficult to find the right questions: Traditional approaches to data analysis such as data mining and OLAP are fundamentally abstraction or aggregation mechanisms: statistical properties of the data are computed to prove hypotheses. This design begs the question of how the hypotheses can be formed in the first place; often the hypotheses are not well-formed, or are based on information external to the data.

Example 1: Consider a person analysing student grades at Berkeley. Will she find evidence of discrimination against people with names that sound like those from a certain ethnic group? This is an imprecise hypothesis that is difficult to express formally, and it will be difficult for the analyst to find this through aggregations.

Example 2: Conversely, consider a person exploring flight delay information. Suppose that flights originating from San Francisco, Seattle, Portland, and Los Angeles have high delays because of a strike in an airline with heavy traffic from the west coast. This is not an imprecise filter and could have been captured with a SQL query. However, the connection between the items was based on some information external to the database (geography, and the domain knowledge of the analyst). Hence this hypothesis is unlikely to crop up in any abstraction from the data.

Noise in data: A related problem with aggregation-based approaches is that we only look at the big picture, which may be skewed by irrelevant noise. For example, with flight statistics, the analyst may not be interested in early arrivals. If these are represented as negative delays, the aggregates can be skewed. Similarly, a person studying access patterns to an organization's web-site from different kinds of users may not be interested in hits from web crawlers. Such irrelevant records cannot be removed without apriori knowledge of their existence.

Hard to use: Data analysis tools are hard to use by analysts, who are likely to be domain experts than database experts. First, many tools either require inputs to be in complex query languages, or — when they have a GUI — constrain the types of queries that can be posed. Second, many tools have no support for query refinement; typically, even adding a filter involves issuing a new query. This is annoying because the analysis process often involves running a query several times with minor changes. Third, with current analysis tools there is no way to specify ad-hoc groups of disparate items for aggregation, without using a query language. In the flight statistics example, an analyst may want to group and compute aggregates on the four cities that he suspects as sources of delays based on his domain knowledge. An OLAP tool will only allow aggregation in a static hierarchy — typically at the level of a city or a state.

1.2 The scalable spreadsheet approach to data analysis

We are attempting to tackle the problems mentioned above with a new data analysis tool based on the idea of a scalable spreadsheet. The principal design goals are as follows.

Easy, Simple Query Formulation: We want to let users specify and refine queries dynamically, in a direct-manipulation manner. To support query refinement, requests for filtering should be specifiable not only based on schemas but also based on *values* in previous results. Finally, we want the interface to allow analysts to combine disparate items into groups and compute aggregates on them in an ad-hoc fashion so that items can be grouped based on information external to the database.

Direct-Manipulation Data Exploration: Exploration of detail is an important complement to abstraction for data analysis. As we describe in Section 2.1, it allows users to build intuition about the data and form hypotheses which are often difficult to find by abstraction (unless one asks for the "right" aggregate). Hence, in addition to traditional aggregation facilities, we want to allow users to interactively explore the data sorted along any dimension. We would like to give the analysts some mechanism by which they can focus on regions of interest and get a broader overview of other regions, and some way to add filters to weed out irrelevant records and concentrate on information of interest.

Interactive Request Processing: We desire that exploration operations like sorting, filtering and scrolling, as well as abstraction operations like aggregation and grouping, give interactive responses. Clearly, doing any of these to completion is intrinsically time-consuming on even moderate databases. For aggregations, we want to show intermediate results and continually improve them as more data is processed. Sorting, scrolling and filtering should be supported in a dynamic way that presents some records to the user as quickly as possible, and allows her to access more results as they become available.

1.3 Interplay between HCI goals and System requirements

A recurring lesson in the CONTROL project (*http://control.cs.berkeley.edu*) has been that work on interfaces and work on systems design must go together; new modes of interaction need new system functionalities which in turn need new algorithms. So far, little of the research in data analysis has dealt with data exploration. Making operations such as sorting, grouping and filtering interactive for large data sizes involves carefully designed algorithms under the hood. We need to mask the time to execute the query at the data source, sort the results, and ship them over the network to the client. The traditional DBMS approach to exploration via percentile or top N queries [CK97, CG96] is not only disturbing to the user because new queries have to be issued each time, but is also intrinsically time-consuming unless there is an index on the sort column.

In this paper we mainly deal with the motivation and description of ABC. In the next two sections we discuss the main features of ABC, and how they solve the problems of Section 1.1. We briefly describe the algorithmic and implementation aspects in Section 4 to illustrate that this functionality is achievable.

2 Interactive exploration

Our basic metaphor for exploration is a spreadsheet: a vertical list of records sorted on some attribute of the data. The analyst can explore the data by scrolling — this can viewed as a fuzzy request for tuples in a range defined on the sort column. Clicking on a different column heading changes the sort order (one can also specify lexicographic multi-column sort orders).

We use sampling to allow users to view the data at varying levels of detail. When an analyst requests to "zoom out" in a region (regions correspond to buckets in a histogram of the data based on the sort column values, as we describe in Section 4), we sample a fraction of items from that region and display this reduced set on the screen. This allows the user to explore the data in full detail, or explore only a sample.

To give the analyst further control over the set of values being explored, ABC provides *dynamic filters*. Dynamic filters can be specified in two ways. First, they can be specified declaratively by entering predicates for different columns. Second, they can be specified based on previously returned results by right-clicking on a value and choosing an operator (for example, one may click on a value 73 and choose > from a pop-up menu to select tuples having values for that column greater than 73).

2.1 Advantages of Exploration

Ease of use: The obvious advantage of exploration is that it is a more natural way of looking at data than query-specification. Users are more comfortable using spreadsheets than DBMSs because all requests are specified through direct-manipulation operations rather than complex queries. For example, moving the scrollbar is more natural than issuing a percentile query. Similarly, refining a query by right clicking on a value and choosing a dynamic filter is easier than reissuing a new query. The analysis process typically involves a lot of query refinement, and therefore we want to make our interface for refinement as natural and simple as possible.

Infer hypotheses from details: Exploration allows users to form hypotheses about the data from specific details. As described in Section 1.1, these hypotheses could not always have been arrived at by abstraction mechanisms such as aggregation. Returning to the ethnicity example from Section 1.1, the analyst would naturally explore student records sorted by GPA because she is trying to analyze student grades. While scrolling in the low GPA regions, she may find by "eyeballing" that many of the students have names that sound like those from a particular ethnic group. Similarly, with the flight delay example, the analyst can find upon sorting by delay and scrolling to the bottom that most of the high delay entries originate at San Francisco, Los Angeles, Seattle, and Portland. The analyst can now form a hypothesis about the cause of the delay based upon his knowledge of geography and the airline strike. We discuss in Section 3.1 how ABC's grouping facilities allow analysts to test these hypotheses by collecting such entries into groups and compute aggregations on them.

Weed out noise: Often, looking at detailed data can reveal potential "noise" that may skew the results of an aggregation, as described in Section 1.1. The analyst can find such irrelevant records by exploring the data and he can add dynamic filters to eliminate them as and when he finds them.

Focus on information of interest: Conversely, while exploring the data one may find specific portions of interest. Returning to the flight delay example, after suspecting that some cities have high delays, the analyst may want to look at these in more detail by adding a filter to exclude other data and then zooming down to a finer granularity. An exploratory interface allows analysts to study different portions of the data at different levels of detail, without reissuing queries. This facility is especially useful for skewed data where no single aggregation granularity may be appropriate.

3 Interactive Aggregation, Grouping

As a complement to interactive exploration, ABC provides support for abstracting information from the data in the same interface. We believe that a direct manipulation mechanism for combining these two operations is useful because a hypothesis formed during exploration will typically be tested by abstracting out aggregates or other statistical quantities from the data.

3.1 Interactive, Direct Manipulation Group Specification

3.1.1 Eyeball Grouping

As we have seen in the ethnicity discrimination example, an important advantage of exploration is that otherwise unnoticeable features of the data may be spotted. When this happens the user should be able to group entities of this kind together and compute aggregates on them. In essence, the user is specifying groups by example based on returned values, rather than based on columns in the schema.

After the user highlights the sample tuples that fall into the desired category and drags them to a "grouping region", the system pops up an interface to formally specify a filter (or more generally, a user defined grouping function) that captures values that are similar to those highlighted. This is akin to using a training set to develop a classification algorithm, and we believe that standard classification or machine learning techniques can be incorporated into this framework. For data such as names, even simple filters such as common substrings and similar sounds are useful for capturing patterns. Note that these filters will likely be fuzzy and not capture the desired category exactly, but we believe that even such an approximate grouping will be useful – such groups can not easily be captured via traditional querying.

3.1.2 Grouping Legos

ABC's exploratory interface encourages analysts to collect apparently disparate data items into groups, based on connections that are external to the data (such as a strike in the example from Section 1.1). Since the analyst may not be able to specify all the values in a group at once, ABC allows him to merge groups, and to append items to existing groups. For instance in the flight statistics example, he first sees San Francisco and

Oakland and groups them to look at their aggregate delays. He then notices Portland and Los Angeles which were also affected by the strike and adds them to the existing group. Alternatively, he may want to experiment with collecting items into groups in many different ways based on different connecting factors, and we want to allow this without requiring him to issue a new SQL query each time.

3.2 Online Aggregation

After choosing the group, the user specifies aggregates to be computed on items in that group. These aggregates are dynamically computed via Online Aggregation [HHW97] – continually improving estimates of the aggregates are provided as more and more data is fetched from the source. Continual estimation is essential here because the interface encourages submission of ad-hoc queries and groups.

4 A peek at systems issues

Clearly, making filtering, sorting, scrolling, and zooming interactive is not easy for large data sets. We exploit the fact that the user can see only a small number of items on the screen at a given time. On a sort request, we give her the illusion that the sorting was instantaneous by preferentially retrieving a sample of items from the region around the current scrollbar position. While she is exploring the data fetched so far, we fetch more items (from the source) and sort them in the background. We use a (dynamically computed) equi-depth histogram over the data to decide which items need to be returned at a given scrollbar position, and Online Reordering [RRH99] for sorting the tuples while the user is looking at the rest of the data. Briefly, we prefetch items from the source and build an approximate hash index on disk based on the histogram buckets while the user is scanning the data that has been fetched so far. To avoid confusing the user, we do not update the screen with newly fetched items unless she scrolls, or explicitly asks to refresh. Even so, because items are fetched dynamically, we have had to carefully design the reordering and display so that the user does not see conflicting information on the screen while scrolling.

Similarly when the user adds a dynamic filter, we do not explicitly issue a new query to the data source. Instead we apply the filter at the client side on the items that have already been fetched from the source. This allows us to reuse all the items that have been fetched so far, and avoids an initial delay. We have found that this simple approach is highly effective when the main usage pattern is an analyst issuing a broad query, and then gradually adding filters to refine the query. A natural extension of this is a generic caching strategy which decides whether to reissue a query or to apply the filters on a cached version based on estimated costs.

Another optimization is to exploit the small number of tuples seen by the user at any given time to execute the query in two steps. We can first retrieve only a key column and the sort column, extract the items to be displayed on the screen, and then issue a semi-join query to fetch the remaining columns for these items alone.

5 Implementation Status

We have been building ABC as a C++ client application with a Java (Swing) user interface. We use ODBC to talk to data sources and have tried to avoid making any assumptions about the data source (about its filter capabilities, aggregation capabilities etc.) because we feel that ABC will be useful over non-DBMS data sources such as search engine results, text files, and sensor feeds too. We have so far developed capabilities for interactive exploration based on Online Reordering as briefly described in Section 4. We have also developed a scheme for dynamic filters which always tries to apply the filters on previously cached results, thereby eliminating starting delays when queries are refined. Our studies over web access logs and flight statistics data show that even our untuned initial prototype is able to provide sub-second response times to mouse gestures for fairly large (about 100MB) data sets.

6 Looking Ahead

We have outlined our vision for ABC, a new interface to data analysis based on the notion of a scalable spreadsheet. The key differences are an emphasis on interactive responses and direct-manipulation controls for specifying and refining queries, explicit support for dynamic, ad-hoc grouping, and the combination of exploration and aggregation. We have given several examples to motivate these features. Ultimately their utility lies in the way they allow users to identify several plausible hypotheses and quickly test them with natural, intuitive operations, instead of issuing complex queries and waiting till they complete.

We intend to conduct user-studies to see which features are useful and which are annoying. A useful extension of ABC is to develop an interactive transformation tool for structured or semi-structured data. ABC can be used to explore data that is in one format, and interactively generate transformation functions to another format. This will be an extension of eyeball grouping to eyeball structure detection and transformation.

Acknowledgments

We would like to thank everyone in the database group at Berkeley for many useful discussions. Conversations with Adam Bosworth, the database group members, and the decision theory group members at Microsoft Research helped us understand the pros and cons of a scalable spreadsheet. We thank the anonymous referees for their valuable comments. Computing and network resources were provided through NSF RI grant CDA-9401156. This work was supported by a grant from Informix Corporation, a California MICRO grant, NSF grant IIS-9802051, and a Sloan Foundation Fellowship.

References

[CG96] S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. In *SIGMOD*, 1996.

[CK97] M. Carey and D. Kossman. Processing Top N and Bottom N queries in SQL. In *IEEE Data Engineering Bulletin*, 1997.

[F+96] U. Fayyad et al. The KDD process for extracting knowledge from volumes of data. *CACM*, 39, 1996.

[HHW97] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *SIGMOD*, 1997.

[OJ93] V. O'day and R. Jeffries. Orienteering in an information landscape: How information seekers get from here to there. In *INTER-CHI*, 1993.

[RRH99] V. Raman, B. Raman, and J. Hellerstein. Online dynamic reordering for interactive data processing. In *VLDB*, 1999. (to appear).