Query Steering for Interactive Data Exploration

Ugur Cetintemel*, Mitch Cherniack†, Justin DeBrabant*, Yanlei Diao±, Kyriaki Dimitriadou†, Alex Kalinin*, Olga Papaemmanouil†, Stan Zdonik*
{ugur,debrabant,akalinin,zdonik}@cs.brown.edu, {mfc,kiki,olga}@cs.brandeis.edu, yanlei@cs.umass.edu
Brown University*, Brandeis University†, University of Massachusettes, Amherst±

ABSTRACT

Traditional DBSMs are suited for applications in which the structure, meaning and contents of the database, as well as the questions to be asked are already well understood. There is, however, a class of applications that we will collectively refer to as Interactive Data Exploration (IDE) applications, in which this is not the case. IDE is a key ingredient of a diverse set of discovery-oriented applications we are dealing with, including ones from scientific computing, financial analysis, evidence-based medicine, and genomics. The need for effective IDE will only increase as data are being collected at an unprecedented rate.

IDE is fundamentally a multi-step, non-linear process with imprecise end-goals. For example, data-driven scientific discovery through IDE often requires non-expert users to iteratively interact with the system to make sense of and to identify interesting patterns and relationships in large, amorphous data sets. To make the most of the increasingly available complex and big data sets, users would need an "expert assistant" who would be able to effectively and efficiently guide them through the data space. Having a human assistant is not only expensive but also unrealistic. Thus, it is essential that we automate this task.

We propose database systems be augmented with an automated "database navigator" (*DBNav*) service that assists as a "tour guide" to facilitate IDE. Just like a car navigation system that offers advice on the routes to be taken and display points of interest, DBNav would similarly steer the user towards interesting "trajectories" through the data, while highlighting relevant features. Like any good tour guide, DBNav should consider many kinds of information; in particular, it should be sensitive to a user's goals and interests, as well as common navigation patterns that applications exhibit. We sketch a general data navigation framework and discuss some specific components and approaches that we believe belong to any such system.

1. Introduction

With the advance of auto navigation systems, your days of scribbling down vague directions from the web, struggling with oversized, out-of-date paper maps while driving, and stopping at gas stations to ask for directions are finally over. Now you can relax and enjoy the view as the navigation system guides you to your destination with turn-by-turn directions. It shows your location on a graphical map along with various classes of points of interest such as restaurants, gas stations, rest areas, or touristic attractions. If you are up for sushi, it can suggest and offer directions to the nearby restaurants that serve sushi.

Taking this service one step further, if we are planning a trip to an unfamiliar part of the world, you might seek the advice of a travel agent or a tour guide who can ask you a series of questions, and based on your answers, will suggest an itinerary that is best suited for your interests. We also expect the guide to accompany you on your trip, and to dynamically make adjustments to your itinerary based on your reactions on what you have seen, what it knows about you, and its experience with other tourists. To facilitate IDE, we seek to provide an automated service, which we refer to as *DBNav*, similar to the auto navigator or the tour guide.

6th Biennial Conference on Innovative Data Systems Research (CIDR '13)
January 6–9, 2013, Asilomar, CA, USA.

In IDE applications, users try to make sense of the underlying data space typically by navigating through it, focusing (zooming in) on parts of the data as they identify interesting "stuff", defocusing (or zooming out) on data that is of little or no interest, jumping to related to "stuff", and repeating this process as much as necessary, typically all through a visual interface. IDE remains as a resource- and labor-intensive task despite its growing importance as current DBMSs cannot effectively support such interactive, multi-step tasks with imprecise goals. In particular, DBMSs fall short when providing the following key functionality:

Interactive performance: DBMSs should support online query processing, which is critical for human-in-the-loop analysis and exploration. Interruptible queries and progressive, anytime results coupled with result-quality estimations can make exploration much faster, more effective and engaging, especially in the presence of big data.

Navigation help: DBMSs should assist users with easy navigation through the data space. Such help can come in the form of allowing users to readily express exploratory query sequences, i.e., "query sessions", automatically generate query sessions with little input from the user, or provide recommendations that guide users towards interesting parts of the data.

Visualization: DBMSs should provide effective presentations of the underlying data space to allow users to quickly grasp the data "landscape" and identify interesting features. The form of such presentations can vary according to what the system knows about users requirements.

Personalization and customization: DBMSs should develop and leverage models of users interests, goals, and database interaction styles to provide user- and application-centric, customized query steering and data visualization support.

Although much of the functionality above has been explored and point solutions have been proposed (e.g., online and approximate query processing [HH07], query recommendations [CE09, SM10], collaborative databases [NB09], and user profiles [CF01]), offering integrated support for this collection implies fundamental changes in the design and architecture of DBMSs. Furthermore, in addition to providing the existing features at scale, DBMSs need to provide better support for:

1. Query sessions: Just like how automobile navigation systems help with the planning of "trips", DBMSs should do so with *query sessions*, which we define as a *sequence* of *related* queries. Exploration is data driven: each query typically serves as a jumping-off point for the next. As such, IDE rarely involves independent, entirely ad hoc query sequences. Thus, DBMSs should be aware of this session-oriented usage pattern and provide primitives to express sessions and optimize their execution.

2. User and application profiles: DBMSs provide generic, one-size-fits-all behavior for all users. Rich user models (e.g., profiles) that capture users' interests and goals can be used to offer personalized services. Such user models can be manually specified and/or automatically learned by the system. In either case, interfaces that allow users to provide feedback to the system need to be provided. Likewise, while many large datasets have a relatively small set of sensible "trajectories" through them, DBMSs are agnostic about the overarching applications. Having a model of applications (e.g., possible operations, common navigational patterns) would allow a DBMS to further customize its operation per application.

In the sequel, we focus primarily on how DBNav can provide these services to facilitate online IDE, highlighting the primary challenges and offering initial directions for solutions.

2. Query Steering

We use *query steering* to refer to the process of assisting a user to navigate through a complex data space. Steering typically results in a query session that is either generated by the user (perhaps with some prompting by the system), or entirely by the system (e.g., based on past user and application profiles).

2.1 Steering Modes

We envision the following steering modes for generating query sessions (in increasing order of system involvement):

Manual Steering: The user would ask a sequence of queries from the same or related templates, tweaking the parameter values until she is content with the accumulated results (or runs out of time). We refer to this base usage mode as *manual query steering*, as the user manually specifies the queries in the session one by one.

Power Steering: Alternatively, such a query session can be expressed through a *steering policy* applied on a *steering template*. The steering policy here indicates the range of values and the order each template parameter will be instantiated to generate the query sequence. We refer to this mode as the *power steering*, as the user can specify an arbitrarily long, prioritized query sequence at once, thus has the ability to explore large swaths of the data space easily. At the same time, the system can well optimize this known query sequence (e.g., [AC06, BS07]). Power steering is motivated by *parameter sweeping* tasks commonly used in finance [HS01] and scientific domains (e.g., climate modeling) in backtesting over historical data to identify the optimal model/parameter combinations.

The user can also specify *steering goals* that include additional predicates and functions to identify a subset of queries, this time as a function of the underlying data or query results as opposed to the parameter values. For example, in a session over astronomical data, a user may ask for "the largest region in the sky where the average blue surface brightness value is higher than 24". While the ability to ask such a query is useful for exploration, it can neither be easily expressed nor optimized using standard GroupBy or windowing constructs, especially for multidimensional data.

Auto Steering: The system takes control of steering by leveraging user profiles. Users may directly define (ir)relevant objects and features, and/or provide feedback on the relevance of system. The system builds the *user profile*, investigates the data space on behalf of the user in the background and *automatically* recommends queries that retrieve matching data.

Integrated: All of the above steering techniques, manual, power and auto, should be tightly integrated into a single framework.

2.2 Profiles and Interaction Histories

DBNav relies on models of user interaction behavior and interests for personalization and optimization. We distinguish two profiles types, *user profiles* that characterize the interests and navigation actions of users [CF01] and *application profiles* that characterize the application-database interactions. Application profiles can be seen as summaries of how multiple users of the same application interact with the database.

In Figure 1, we illustrate a simplified application profile created from a pan-and-zoom style visual front-end. This is a hierarchical model that layers query template models and query parameter models. The template model shown is an order-1 discrete-time Markov chain where the states (circles) correspond to query templates that are executed in response to pan-and-zoom user actions. Query transitions (arcs) are labeled using steering operations (e.g., zoom in/out, etc) and transition probabilities. Steering operations correspond to the steering algebra operators described in Section 2.3. For each query template, we have a separate probabilistic query parameter model that can be used to predict with which parameter values the template will be instantiated next (e.g., for the "Pan" template, we show probabilistic association rules that model to which "logical data cells" the next pan operation will take us; a pan operation on data cell 2 will move the focus to cells 3 and 5 with probability 0.8).

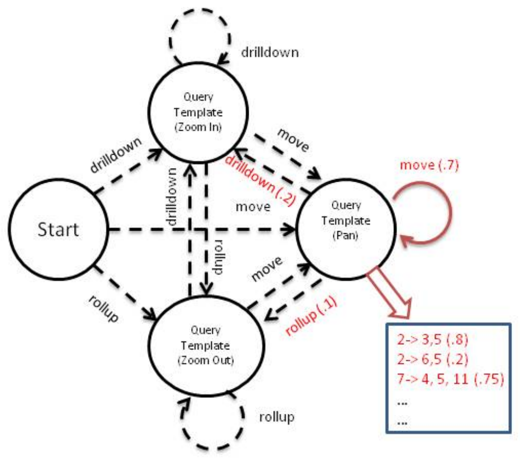

Figure 1: A navigational profile for a pan-and-zoom app

Furthermore, DBNav automatically logs and stores all user-database interactions. In addition to helping with the creation and maintenance of profiles, these *interaction histories* enable a number of useful navigation operations, including: (i) (*time-travel*) going back to any point within an interaction history; (ii) (*parallel worlds*) branching off to explore alternative query paths; (*versions*) materializing the accumulated results; (*replay*) applying a subsequence of interaction histories on other data sets or the same data sets after revisions; (*reuse*) searching for others' interaction histories to identify "workflows" that they can apply on their own of task after customization; and (*suggestions*) receiving suggestions about what new query branches they can explore next (i.e., a "forward" button).

Similar navigation operations have been explored in the past, primarily in the context of visualization workflows [CF06]. The challenge here involves generalizing these ideas to exploratory query processing. Interaction histories and user logs have also been used in the context of security and auditing applications. However, our focus is on the optimization of query sessions based on predictions of user behavior. Finally, OLAP systems support similar user navigational activities (e.g., roll-up, drill-down/up, etc) with our data steering framework. In these applications, query optimization relies on data locality techniques (e.g., prefetching of surrounding data cells) instead of user interaction profiles and application navigational models, which is the focus of our work.

2.3 Steering Algebra

DBNav uses for the manual steering mode a *steering algebra* to represent and reason about query sessions. This algebra is data centric: it defines popular navigational idioms on the underlying data space without being tied to a specific query language. As such as it can formally describe query-to-query transitions and facilitate custom optimizations when processing query sequences.

Our current algebra contains operators that represent common actions such as zooming in/out on attributes, asking for an overview or a more detailed view of query results, and relating (i.e., joining) query results with other relations. Examples operators (on a query *Q*) include (but are not limited to):

- **NARROW(P)**: Restricts the result set by adding the predicate "AND P" in the WHERE or HAVING clause of Q.
- **DRILLDOWN(B_1,...,B_m,[A_1=F_1(E_1),...,A_m=F_m(E_m)])**: Provides a more detailed view by: (a) replacing non-aggregate expressions in the SELECT clause of Q with B_1,...,B_m, (b) adding list of aggregate expressions, "F_i(E_i) AS A_i" to the SELECT clause of Q, and (c) adding B_1,...,B_m to the GROUP BY clause of Q.
- **MOVE(P, V_1,...,V_m)**: Replaces the parameters values of predicate P with the new list of values.
- **RELATE(R)**: Implements a join operation by: (a) adding R to the FROM clause of Q, (b) adding R.A = S.A to the WHERE clause of Q for each S relation in the FROM clause of Q that has a common attribute A with the R relation, and (c) adding unique attributes in R to the SELECT clause of Q.

DRILLDOWN (age, avgD=avg(dosage) on Q_0	Q_1: SELECT **age, avg(dosage) AS avgD** FROM trials C WHERE disease = "diabetes" **GROUP BY age**
RELATE (hospitals) on Q_1	Q_2: SELECT age, avg(dosage) AS avgD, **H.hospital_name, H.hospital_id** FROM trials C, **hospitals H** WHERE disease = "diabetes" **AND C.hospital_id= H.hospital_id** GROUP BY age
NARROW (hospital_name="MGH") on Q_2	Q_3: SELECT age, avg(dosage) AS avgD, H.hospital_name, H.hospital_id FROM trials C, hospitals H WHERE disease = "diabetes" AND C.hospital_id= H.hospital_id **AND H.hospital_name= "MGH"** GROUP BY age

Table 1: Example steering operations

Example. Let us assume an IDE query session in a clinical trial database searching for treatments related to a patient's condition. An initial user query, Q_0, could be "SELECT * FROM trials C WHERE disease= "diabetes"", where trials is the table with the clinical trials information. In Table 1 we show the queries obtained by applying a sequence of steering operations that (1) return a more detailed view of the trials by evaluating the average insulin dosage per age, (2) relate the result trials with the hospital they took place and (3) zoom into the trials conducted at MGH.

Note that the steering algebra operators are *context sensitive*: they cannot be interpreted in isolation. They describe how to get from a given query to a successor query, and thus, require a context. The steering algebra can be used to manually define a query session. Leveraging the algebra to capture system-assisted steering operations (i.e., in power and auto steering modes) is one of the research challenges towards providing an integrated steering framework for IDE applications.

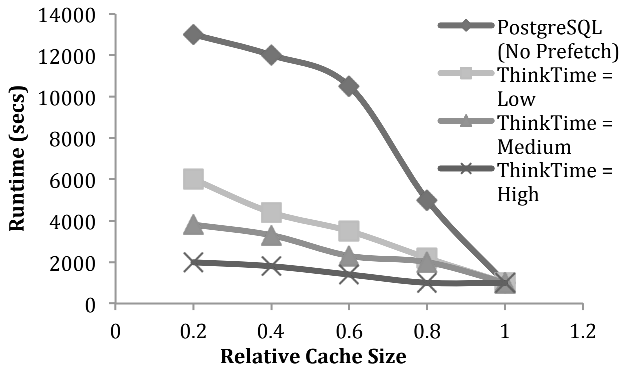

Figure 2: Impact of profile-driven prefetching and caching

2.4 Steering Optimizations

Non-determinism is a salient aspect of query steering and arises due to two main reasons: First, in manual steering *query transitions* may not be known in advance. Second, a user can *interrupt* query execution at any time. Effectively dealing with such non-deterministic behavior requires novel profile-driven modeling and planning for reuse, which we briefly discuss below.

Profile-driven prefetching and caching. Given an application profile (as shown in Figure 1) that shows probabilistic query transition sequences, we can prefetch more data than what Q_i needs in anticipation for Q_{i+1}. Given the probabilistic branching, there will often be multiple likely next queries (and parameter values), so the challenge is to decide what to prefetch under such uncertainty. Various tactics are possible here, including prefetching depth first, breadth first, or using a hybrid approach while pruning branches with low probability, or prefetching data that would benefit multiple branches. A complementary problem is that of profile-driven caching, where the profile information can be used to decide what data are least likely to be accessed in the future and thus can be evicted.

In Figure 2, we quantify the potential benefits of profile-driven prefetching/caching by running synthetically generated query sessions that mimic common analytic tasks in a stock-data analysis exploration application (e.g., fast upward/downward trending). For a pan-heavy workload, we learned a hierarchical application profile similar to the one depicted in Figure 1, for which we combined a variable-order Markov model for template modeling and sequential association rules per template. The figure compares the performance of DBNav (running on top of PostgreSQL) with prefetching and caching for varying think time values (i.e., average time between consecutive queries) vs. vanilla PostgreSQL (no prefetch/LRU), all running the same workload over a 50GB NYSE dataset on an Amazon EC2 m2.2x large instance. We see that the profile-driven approaches are effective in reducing query execution latencies especially for small caches (when smart use of the cache becomes more important) and long think times (that allows more data to be prefetched).

Query checkpointing to facilitate reusable progress. The "overlap" of queries within the same session introduces opportunities for the *reuse* of the execution "progress". We can devise execution strategies that would promote and maximize reuse. One approach that we introduced toward this end is *query checkpointing* in which we convert a query *Q* to one or more *checkpoint queries*, $Q = f(C_1, C_2 \ldots, C_n)$, to be executed in turn. When a user transitions from Q_i to Q_{i+1}, the optimizer strives to express Q_{i+1} in terms of the already materialized checkpoints of Q_i. In the example of Table 1, Q_1's checkpoint queries, C_i

evaluate the sum and count of dosage for each district age value, X, (instead of the average):

C_i: SELECT age, sum(dosage) AS sum, count(dosage) as count
FROM trials C WHERE disease = "diabetes" AND age=X

Assuming C_i is materialized, RELATE(hospitals) operation can use these checkpoints and can be implemented as:

SELECT * FROM C_i, hospitals H
WHERE C_i.hospital_id= H.hospital_id

A challenge is to design these checkpoints to maximize reusability in consideration of the predicted future queries. This can be seen as an online materialized view selection problem. We can leverage the knowledge of the semantics of steering operators and profiles to predict upcoming query transitions, e.g., attributes likely to be involved in grouping, aggregations, relate operations and predicates. The order of executing and materializing the checkpoints is also of concern as the user can interrupt the current query any time. Finally, more aggressive reuse across queries is possible at the level of transient data structures, such as hash buckets, which are created per query.

Query sequence optimizations. Power steering will benefit from well-studied multi-query optimization to more specialized and less studied techniques for query workloads as sequences [AC06, BS07]. The salient aspect of steering sequences is that they consist of correlated queries and thus aggressive reuse-oriented techniques such as those sketched earlier would apply. Power steering with goal functions, on the other hand, requires fundamentally new query optimization techniques that are, in the most general case, similar to basic search algorithms used in AI. Query relaxation techniques that use lattice-based search structures to find queries with desired properties (e.g., [CK09]) are partially applicable but need to be significantly generalized so that a meaningful collection of constraints can be supported. This is an open area that would benefit from novel sampling and function indexing techniques.

Efficient query learning. In auto steering, users engage in a conversation with the system and simply characterize data samples (objects, features, etc.) as relevant or irrelevant to their interests. Based on the user's profile, the system *automatically* formulates "classification" queries that retrieve matching data. The user continues to approve or disapprove of new samples until new iterations receive mostly positive votes.

Through this interactive process, the system must identify query attributes and predicates that match the user's interests. While existing feature selection algorithms can identify relevant attributes, many often translate attributes to a lower dimensional space from which there is no straightforward reverse mapping to the original attribute space. One alternative is to use easier to interpret classification techniques, such as decision trees, that make it easier to (i) identify the classification attributes and values that maximize the *information gain* and (ii) translate back to query expressions (e.g., map each tree branch to a WHERE clause).

Sample selection has a significant impact on auto steering. Initial samples should capture the diversity of the data space aiming to early on eliminate non-interesting data. This is essentially an online learning problem that involves striking a balance between *exploration* (of uncharted "territory") and *exploitation* (of current profile information). We believe that rich multi-dimensional histograms that characterize the entire data space would provide useful when making such tradeoffs.

Another challenge is to minimize the number of iterations required to converge to a result of an acceptable quality. An idea is to analyze past application and user profiles to "predict" a user's interests and steer her along semantic "trajectories" of users with similar interests.

Customized data visualization. Profiles and interaction histories can be used to customize visualizations to further aid steering support. For example, objects can be *non-uniformly* scaled and drawn so that all the "important" objects are clearly visible. This approach has long been successfully used in hand-designed tourist maps. DBNav should support algorithms to set the appropriate resolution for each object based on *profiles and locality principles* and adapt the resolution dynamically as navigation progresses. The visualization system should also support various interaction history operations (e.g., time-travel, replay, forward). Finally, the visualization environment should make it easy for the user to interact with the system, including the ability to input profile information and provide feedback.

3. Conclusions and Ongoing Work

We sketched our vision for how an automated navigation assistant service, DBNav, for interactive exploration of large datasets. DBNav would help a user quickly navigate through a complex large data spaces. We argued that such a service must be able to (1) suggest next steps in a query sequence that methodically move the user through the data in a meaningful way. We call this query steering. We have further argued that (2) query recommendations must be produced quickly.

Profiles of both user interest and application characteristics are the key ingredients to realizing this vision. Profiles can be either supplied by the user or learned from interaction logs. We sketch three query steering modes: (1) manual steering, (2) power steering, and (3) auto steering. We give some simple examples suggestive of where this technology could go, but should not be interpreted in any way as the end point.

A powerful visual front-end is critical IDE applications. The visual component would also use profiles to identify how to map visualizations to pixels in a way that best serves the user's needs. We believe that a data navigation system such as the one sketched in this paper is essential to deriving insight from the huge and complex datasets that one encounters in modern applications such as those in the sciences. A modern DBMS must be a full participant in this endeavor and thus must supply a DBNav-like system alongside a traditional query interface.

4. Acknowledgements

This work is funded in part by NSF Grant IIS-1049974, IIS-1111423, IIS-0905553, and a gift from Intel.

References

[AC06] S. Agrawal, E. Chu, and V. Narasayya, "Automating Physical Database Design: Workload as a Sequence," in SIGMOD'06.
[BS07] I. T. Bowman and K. Salem, "Semantic Prefetching of Correlated Query Sequences," in ICDE'07.
[CE09] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Query Recommendations for Interactive Database Exploration," in SSDBM'09.
[CF01] M. Cherniack, M. Franklin and S. Zdonik, "Expressing User Profiles for Data Recharging," in IEEE Personal Communications, 2001.
[CF06] S. P. Callahan, J. Freire, E. Santos, et al., "VisTrails: Visualization meets Data Management," in SIGMOD'06.
[CK09] C. Mishra and N. Koudas, "Interactive query refinement," in EDBT'09.
[HH07] J. Hellerstein, P. Haas, and H. J. Wang, "Online aggregation," in SIGMOD'97.
[HS01] H. Hochheiser and B. Shneiderman, "Interactive exploration of time series data," in DS'01.
[KW04] A. Kadlag, A. V. Wanjari, J. Freire and J. R. Haritsa, "Supporting Exploratory Queries in Databases," in DASFAA'04.
[NB09] N. Khoussainova et al, "A Case for a Collaborative Query Management System," in CIDR'09.
[SM10] E. Sadikov, Jayant Madhavan, Lu Wang, A. Y. Halevy, "Clustering Query Refinements by User Intent," in WWW'10.