Mind your dependencies for semantic query optimization

Eduardo H. M. Pena[1,3], Eric Falk[2], Jorge Augusto Meira[2], Eduardo Cunha de Almeida[3]

[1] Federal University of Technology - Paraná, Brazil
eduardopena@utfpr.edu.br
[2] University of Luxembourg
eric.falk, jorge.meira@uni.lu
[3] Federal University of Paraná, Brazil
eduardo@inf.ufpr.br

Abstract. Semantic query optimization uses dependencies between attributes to formulate query transformations and revise the number of processed rows, with direct impact on performance. Commercial databases present facilities to define dependencies as not enforced constraints. The goal is to help the query optimizer in cases where the database is denormalized or simply lost dependencies in the design. However, feeding these facilities is a manual task which is tedious and error-prone. An attractive alternative is the automatic discovery of dependencies, but the cost of finding dependencies increases with the number of rows and attributes in the dataset. In this paper, we stick to the automatic discovery approach, but to reduce the cost we focus on dependencies matching the current queries in the pipe (ie., workload). Initially, we rely on a large set of functional dependencies computed in batch with state of the art algorithms in the literature. Over time our focused dependency selector (FDSel) chooses exemplars to feed the query optimizer. Therewith we eliminate further manual interactions. The automatically selected exemplars exhibit statistical properties that resemble those of the initial dependency set. This demonstrates the effectiveness of our proposed approach. In the best case scenario, by applying the FDSel for join elimination on a real-world database, we reduce query response time by more than one order of magnitude.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.2.8 [**Database Applications**]: Miscellaneous

Keywords: Data profiling, Functional dependencies, Integrity constraints, Query optimization

1. INTRODUCTION

The ongoing explosion of data-intensive scenarios and the urgency to store and analyze large amounts of data has brought new challenges to the data management community. From scientific to business domains, modern organizations hold a wide variety of data provisions with datasets constantly increasing their size at exponential rates. Thus, mechanisms to help IT professionals or scientists understand their data became crucial. These mechanisms provide valuable insights to be used for better data exploitation, outcome refinement, and even for query performance improvement. On this subject, several data profiling tasks have been studied [Abedjan et al. 2015]. They typically aim to efficiently analyze large data sets, to expose relevant metadata, such as patterns among records and correlation between attribute sets.

One of the most important data profiling task is dependency discovery, particularly, the discovery of the functional dependencies (FDs). While FDs are defined as integrity constraints in database design phases, manually updating them as the application and data evolve becomes an error-prone task which may even be left behind in denormalized databases (e.g., data warehouses). In turn, automatic dependency discovery does not rely exclusively on schema information, but considers the data tuples

of the database as well.

Many studies have shown that dependency discovery is quite challenging. On complexity level, a review by Liu et al. [Liu et al. 2012] has shown that finding FDs is in $\mathcal{O}(n^2(\frac{m}{2})^2 2^m)$ for a relation with m attributes and n records. Demetrovics et al. [Demetrovics et al. 2006] investigated the possible number of dependencies as a combinatorial problem and partially proved an upper bound of $2^m - 1$ for relations with m attributes. Nevertheless, discovery algorithms have been proposed over the years [Papenbrock and Naumann 2016]. They have devised several techniques to handle a larger number of records and columns. Alternatively, the theoretical and practical aspects of approximate functional dependencies (AFDs) [Kivinen and Mannila 1995], that is, functional dependencies that partially hold in a relation, have also been considered in order to reduce the discovery complexity.

The number of FDs radically increases with the number of columns in the dataset. This number may increase drastically as the number of columns goes up, e.g., in the region of millions for datasets with hundreds of columns and thousands of records [Papenbrock et al. 2015]. The main problem is that selecting which of the dependencies are most relevant for a given task is left for human analysis. It is particularly difficult to understand the relationships among hundreds, or even thousands, of dependencies spread across multiple relations. Therefore, the selection process should be adjusted in accordance with the use context of the dependencies. This should not only prune the unnecessarily large number of results but it should also provide more meaningfulness to the selected dependencies.

Interesting measures have been proposed to score FDs and other types of constraints. These measures are primarily based on the statistical properties of the data and have shown good potential to filter dependencies for tasks such as FD evolution [Mazuran et al. 2016], data cleansing [Chiang and Miller 2008] and normalization [Papenbrock and Naumann 2017]. However, those measures may produce inconclusive recommendations to be explored by semantic query optimization [Ilyas et al. 2004]. As observed in [Kimura et al. 2009], data dependencies should be exploited with caution. They may impose additional performance penalties in planning phases as the number of dependencies increases.

We present the focused dependency selector (FDSel), a data-driven, query-aware tool to effectively select relevant FDs for semantic query optimization. We hypothesize that the information from the workload of the application (e.g., selection filters in SQL statements) is a powerful asset to narrow the large number of FDs discovered in the datasets. First, FDSel associates summaries of application workloads with the set of discovered FDs in application data. Then, it can use different strategies to recommend sets of FDs that offer the best trade-off between a reduced number of FDs and best gains in query execution time with semantic query optimization. We refer to these sets of FDs as exemplar FDs. The FDSel is also responsible for setting and triggering the semantic optimizations, acting as a middle-ware between the user applications and the database.

Our contributions are as follows. We present a novel mechanism to combine the semantic information found in functional dependencies with workload awareness to help semantic query optimization. We formulate effective procedures to select exemplar FDs from the large sets of FDs returned by automatic discovery algorithms. We present two schemes in which the exemplar FDs could help in semantic query optimization, namely, join elimination (JE) and order optimization (OO). We provide an experimental evaluation of our tool, using real and synthetic datasets, which shows that our tool is able to effectively select exemplars with adequate statistical properties, and improve query performance without any human interaction.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 gives an overview of the FDSel usage scenario. Section 4 details the properties and procedures of FDSel. Section 5 outlines our experimental evaluation of FDSel. Finally, Section 6 concludes this paper and presents future directions.

2. RELATED WORK

Functional dependencies (FDs) have been thoroughly studied in the domain of relational databases. They express relationships between relation attributes to help a wide range of applications, e.g., schema design [Romero et al. 2009], data integration [Amavi and Halfeld Ferrari 2014], data cleaning [Abedjan et al. 2015], and query optimization [Ilyas et al. 2004; Paulley and Larson 2010]. Liu et al. [Liu et al. 2012] present an exhaustive review of dependency discovery. They provide both algorithmic and theoretical perspectives of the main methods for discovering FDs. More recently, the authors of [Papenbrock et al. 2015] presented a experimental evaluation and comparison of the most important FD discovery algorithms. They also provided technical details on the implementation of those algorithms.

The discovered dependencies may become numerous and complex. In this regard, some studies consider scoring schemes to rank their dependencies [Chiang and Miller 2008; Chu et al. 2013; Mazuran et al. 2016; Abedjan et al. 2015]. [Chiang and Miller 2008] evaluates a set of metrics to measure the meaningfulness of an extension of FDs called conditional functional dependencies (CFDs). Similarly, Chu et al. [Chu et al. 2013] use a function based on succinctness and coverage of denial constraints to rank results for user validation. Confidence and goodness measures are associated with FDs in [Mazuran et al. 2016], to support the repairing of FD violations. The Aetas system [Abedjan et al. 2015] ranks AFDs according to association measures to assist users in the validation of temporal rules. Unfortunately, none of the above studies consider ranking strategies for query optimization.

The benefits of using correlations and dependencies of the data for query optimization have been studied for decades [Hammer and Zdonik 1980]. The correlation detection via sampling (CORDS) [Ilyas et al. 2004] recommends sets of attributes for which query optimizers should maintain additional statistics. To do so, CORDS discovers AFDs with a sample based approach which refines sets of candidate attribute pairs, chosen from the catalog statistics and the sampled attribute values. CORDS is limited to two-attribute AFDs, but the results shown in [Ilyas et al. 2004] clearly demonstrate the benefit of using dependencies to improve query execution. EXORD is a three-phase framework for exploiting attribute correlations in big data query optimization [Liu et al. 2018]. It considers source-to-target attribute mappings as correlations. The first phase of EXORD is responsible for validating an initial user-defined set of correlations. It works on simple statistics (e.g, the number of records violating a correlation) to only keep correlations that fall under user-defined thresholds. The second phase uses a cost model to select correlations for deployment. The authors look into an interesting optimization problem: how to select a subset of correlations with the objective of maximizing the total benefit (i.e, correlation applicability). The exploitation phase is responsible for rewriting the queries so that they exploit more efficient access plans.

In this paper, we consider semantic query optimizations, particularly, how to use FDs to modify queries so that performance is enhanced but semantics preserved. Some commercial optimizers (e.g, [Zaharioudakis et al. 2000]) incorporate rewriting strategies into the planning phases. In [Szlichta et al. 2013], the authors investigate the use of order dependencies (a variant of FDs) for order optimization. [Cheng et al. 1999] study variations of join elimination and predicate introduction. Unfortunately, most of the studies on semantic query optimization require the user to specify a set of constraints. In contrast, in the next sections we show that FDSel eliminates this manual interaction by employing automatic discovery and selection of dependencies.

3. OVERVIEW

In this section, we present a high-level description of the focused dependency selector (FDSel). Given the high number of FDs discovered in real-world data, the main question we seek to answer is how to effectively use inherent semantic information to help in specific scenarios. Thus, we design FDSel as a data-driven, query-aware mediating tool that autonomously leverages semantic query optimizations

by exploiting patterns in the data (FDs) and applications (query workload).

Figure 1 illustrates the control flow between the components of FDSel. The input of FDSel is a database along with its catalog, and a representative query workload. The first component of FDSel is the *FDs extractor*. It uses an efficient algorithm to find all FDs in the database tables (1). These FDs determine relationships between groups of attributes and can provide valuable semantic information from the data. FDSel stores all discovered FDs in a buffer for further analysis.

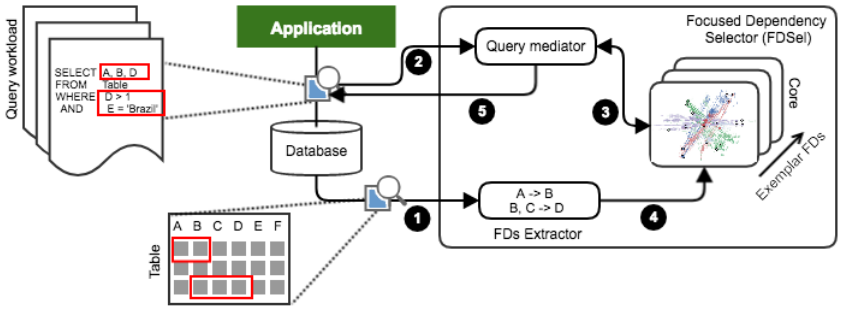


Fig. 1: FDSel Workflow: (1) Finding FDs, (2) Workload Selection, (3) FDs Buffer Lookup, (4) Semantic Optimization, and (5) Query Rewriting.

The second component of FDSel is the *query mediator*. It progressively intercepts the database queries to form a batch (2). In addition, the *query mediator* performs semantic optimizations for each query, if any optimization is available. These optimizations are set by the *core* component, described later. FDSel considers the application workload to be lists of queries that are expected to be executed by the application. Once the *query mediator* has processed sufficient queries, it calls the *core* component of FDSel for updates.

The *core* component receives the workload characterization from the *query mediator* (3), scans the FDs buffer, and starts selecting FDs for semantic query optimization (4). It first counts attribute frequencies from FDs and queries. Then, it combines this frequency information to build occurrence matrices, which form the input for the selection procedures. The *core* component can use three strategies for the selection task. In two of these strategies, the occurrence matrices are used to sort FDs according to their structures and their proximity to the query workload. In short, the structure of a FD is defined by which set of attributes define each other, and the proximity of an FD measures how many attributes it has in common with the workload characterization. The strategies based on ranking are tied to two different interest metrics: distrust, which considers the redundancy of attribute values; and *Mahalanobis* distance, which considers correlations found in the occurrence matrices. The *core* component iterates over the set of ranked FDs to find FDs with appropriate values for these interest metrics. Finally, the third strategy is an adaptation of the affinity propagation clustering algorithm [Frey and Dueck 2007] which works with the occurrence matrices.

The *core* component is also responsible for setting the rewriting strategies for the *query mediator* (5). It only considers optimizations that preserve semantics, that is, there is no change in the output of the rewritten queries. The FDSel sits between user applications and data processing platforms and it is completely decoupled from the internals of any specific database system.

4. FOCUSED DEPENDENCY SELECTOR

In this section, we detail the operation of the focused dependency selector (FDSel). We first review functional dependencies and FD discovery. Then, we describe data structures to combine FDs with

workloads, and the procedures to select FDs. Finally, we describe how to employ the selected FDs for semantic query optimization.

4.1 Functional Dependencies

We follow [Liu et al. 2012] to define functional dependencies (FDs). Let $\mathcal{R}$ be a relation schema, $X \subseteq \mathcal{R}$, and $Y \in \mathcal{R}$. An FD f over $\mathcal{R}$ has the form $f : X \rightarrow Y$, and requires that X functionally determines Y. An FD f is satisfied in r, denoted as $r \models f$, if and only if for all tuple pairs $t_1, t_2 \in r$, the satisfaction of the expression $t_1[X] = t_2[X]$ implies the satisfaction of the expression $t_1[Y] = t_2[Y]$. In other words, the right-hand side (rhs) Y of f is functionally determined by the left-hand side (lhs) X. We use $f.lhs$ to denote the left-hand side, and $f.rhs$ to denote the right-hand side of an FD f.

A set of FDs F is non-trivial and minimal if for all $f \in F$, then f has only a single attribute on its rhs, does not have any redundant attribute (i.e. $Y \notin X$), and there is no Z such that $(X - Z) \rightarrow Y$ is a valid FD. Additionally, the set of FDs F must be reduced, that is, for all $f \in F$ the set $F - f$ is not equivalent to F. The set of non-trivial and minimal FDs can be generalized by inference rules [Beeri et al. 1984] to obtain the sets of all FDs satisfied in r.

FDSel discovers all the non-trivial and minimal FDs holding in the database tables. Several algorithms for FD discovery have been proposed, and many of them have evolved over different versions in the literature. We refer to [Liu et al. 2012] and [Papenbrock et al. 2015] for further details on FDs discovery. In practice, FDSel could use any FD discovery algorithm that, given an instance r, returns the set of non-trivial and minimal FDs over r. FDSel uses an algorithm called HyFD [Papenbrock and Naumann 2016]. The algorithm combines series of row-based and column-based optimizations, such as sampling and compression, that allows HyFD to scale for large datasets. Until the writing of this paper, HyFD was the most efficient FD discovery algorithm with the best results in terms of runtime and scalability.

4.2 Attribute Occurrence Matrices

The query workload provides valuable information to support query optimization. In general, the query workload presents strong access patterns which, either in horizontal level (individual tuples) or vertical level (individual attributes), points out to specific database areas that are more frequently accessed than others. Functional dependencies express semantic consistency requirements for data through sets of dependent attributes. FDSel leverage this characteristic of FDs to reduce the number of attribute sets that should be addressed in semantic query optimization. Thus, the combination of appropriate semantic information and query workload information is a potential asset to help finding alternative execution strategies and, therefore, improve query processing.

FDSel measures the binary relationship between the attributes in a relation and the frequencies in which the incoming queries are touching the attribute values. This binary relationship is also applied to FDs by simply considering their lhs and rhs attributes. The information about the occurrence of attributes in the queries or FDs is initially stored in a $m \times n$ binary matrix O, called the attribute occurrence matrix (AOM). As a first step, the operations for AOMs regarding queries or FDs are the same, thus, we define all the operations in a single AOM. Throughout the definitions, we distinguish how to adjust each AOM to FDs or queries.

Consider a set of queries $Q = \{q_1, ..., q_m\}$, which is expected to run on database relation instances r. For simplicity, we assume there is only one relation r in the database. For each query q_i, FDSel collects the attributes in the operators of q_i (e.g, projection and selection) to compose a set of attributes s. Furthermore, for each FD $f : X \rightarrow Y$ discovered in r, FDSel composes two attribute sets s. The first set is formed with the attributes in X, and the second one is formed with the attribute Y.

Consider a relation and its attributes $\mathcal{R}(A_1, ..., A_n)$, and a collection of sets $S = \{s_1, ..., s_m\}$ such

that $s_i \subseteq \mathcal{R}$. For each $s_i \in S$, and for each $A_j \in \mathcal{R}$, FDSel assigns a binary occurrence value for AOM, as in Function 1:

$$o_{ij} = \begin{cases} 1 \text{ if } s_i \text{ has attribute } A_j, \\ 0 \text{ } otherwise. \end{cases} \quad (1)$$

Each entry o_{ij} indicates whether or not an attribute of $\mathcal{R}$ is touched by one of the elements of s_i.

FDsel estimates three AOMs: for a set of queries (O^q); and, given a set of FDs, for their *lhs* (O^{lhs}); and for their *rhs*(O^{rhs}).

Example 4.1. Let $F = \{A \rightarrow B, BC \rightarrow D\}$ be the set of FDs discovered in an instance r of relation $\mathcal{R} = (A, B, C, D)$. Additionally, consider a set of queries $Q = \{\pi_{(A,D)}(\sigma_{B=10}(\mathcal{R})), \pi_{(A,B,C)}(\sigma_{C=20}(\mathcal{R})), \pi_{(A,D)}(\sigma_{D>1}(\mathcal{R}))\}$ running over r. The AOMs O^q, O^{lhs}, and O^{rhs} are defined as follows:

$$O^q = \begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array},$$

$$O^{lhs} = \begin{array}{c} \\ f_1.lhs \\ f_2.lhs \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array},$$

$$O^{rhs} = \begin{array}{c} \\ f_1.rhs \\ f_2.rhs \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}.$$

The row sum vector of AOM O is given by $\sum_j^m o_{ij}$, and is denoted by $\rho(O)$. Furthermore, let $\gamma(O)$ denote the column sum vector of AOM O, which is given by $\sum_i^n o_{ij}$.

FDSel requires some additional operations on AOMs O. Notice that each AOM can also be represented as a sequence of rows $O = [o_1, ..., o_m]$. Function **elems**(O) returns the set of elements from O, such that for any o_i and o_k of **elems**(O), then $o_i \neq o_k$. In addition, function **count**(O, o_i) returns how often element o_i occurs in the sequence O. Finally, function **length**(o_i) returns the number of o_{ij} such that $o_{ij} \neq 0$.

Given AOMs O and O', FDSel incorporates the number of accesses to the attributes of $\mathcal{R}$ of AOM O' into AOM O with a weighted AOM $\mathbf{O}$ given by Equation 2.

$$\mathbf{O} = \sum_i^m O(i)\rho(O') \quad (2)$$

Notice that it would be possible to integrate the attribute weights from the workload into the AOMs of FDs, and vice-versa. However, FDSel only requires the former type of integration because its goal is to enhance the semantic information of the discovered FDs.

Consider a weighted AOM $\mathbf{O}$ estimated with Equation 2, either $O = O^{lhs}$ or $O = O^{rhs}$, and $O' = O^q$. Function **skewed_sort**$(\mathbf{O})$ returns a sorted version of **elems**$(\mathbf{O})$ that satisfies the following:

$$\text{for any } \mathbf{o}_i \text{ and } \mathbf{o}_{i+1} \text{ from } \mathbf{elems}(\mathbf{O}) \text{ then } \gamma_i(\mathbf{O})\mathbf{count}(\mathbf{O}, \mathbf{o}_i) \quad \leq \quad \gamma_{i+1}(\mathbf{O})\mathbf{count}(\mathbf{O}, \mathbf{o}_{i+1}). \quad (3)$$

The result of **skewed_sort**(**O**) is a sequence of rows sorted according to their frequencies in **O** times the weight of their target attributes. Each entry $\mathbf{o}_{ij}$ of **skewed_sort**(**O**) can be converted into the attributes of FDs, *lhs* or *rhs*, by mapping the equivalent attributes A_j of $\mathcal{R}$ for each element of $\mathbf{o}_{ij}$ other than zero.

4.3 Quality Measures for FDs

Many quality measures have been proposed to measure dimensions of data correlations [Chiang and Miller 2008; Chu et al. 2013; Mazuran et al. 2016]. Unfortunately, most of them only work for approximate FDs and, therefore, they cannot help in our context because the optimizations in FDSel require FDs holding in the entire dataset. As a further matter, estimating those quality measures is time-consuming because it typically requires projections over the entire relation.

An interesting metric for FDs is redundancy, that is, how often sets of equal values for the *lhs* or *rhs* attributes of FDs jointly appear in the dataset. We use a metric from related work [Mazuran et al. 2016] to measure the distrust of an FD $f : X \rightarrow Y$ in r, given as in Equation 4.

$$d = \sqrt{\left(\frac{|\pi_X(r)|}{|r|} - \frac{|\pi_Y(r)|}{|r|}\right)^2} \tag{4}$$

The difference is minimal when the projections over *lhs* and *rhs* approximate in the number of duplicates. In this case, an FD f is less likely to have been discovered by chance, which reduces the level of distrust of f. As an example, consider the simple relation in Table I, and two FDs, $f_1 : AB \rightarrow C$ and $f_2 : D \rightarrow E$, satisfied by the data. The distrust level of f_1 is given by $d(f_1) = \sqrt{(4/6 - 3/6)^2} = 0.16$, and the distrust level of f_2 is given by $d(f_2) = \sqrt{(5/6 - 1/6)^2} = 0.66$.

Table I: A simple relation.

A	B	C	D	E
b	g	5	1	y
b	g	5	2	y
b	g	5	3	y
b	m	6	4	y
b	q	7	5	y
c	g	7	5	y

Notice that the distrust of an FD f does not take any workload characteristic into consideration. The studies on workload characterization typically investigate many parameters, such as I/O throughput, temporal locality, and data variance aspects. A comprehensively report on the subject can be seen in [Basak et al. 2016]. FDSel uses a second quality measure called *Mahalanobis* distance (MD) [Hazewinkel 1987] to combine data instances and workload characterization.

MD works as a similarity measure between the attribute access pattern in the workload and the structure of attributes in the FDs (i.e., *lhs* and *rhs*). We have chosen MD rather than classical measures, such as Pearson correlation or Euclidean distance, because MD is suitable for side comparisons. For example, MD agrees with the intuition that "$A \rightarrow B$ is closer to $A \rightarrow C$" than "$A \rightarrow B$ is to $C \rightarrow D$", disjointly. The same applies to comparisons between FDs and query workload because they account for the same set of attributes.

MD uses multi-dimensional analyses of unequal variances and correlations between the weighted attributes of AOMs to adjust the geometrical distribution. Thus, FDSel estimates the MDs straightforward from AOMs. Assume $u = \rho(O^q)$ and v to be any $\mathbf{o}_i$ from $\mathbf{O}^{lhs}$ (weighted over another O^q).

FDSel estimates the MD as in Equation 5:

$$md(u, v) = \sqrt{(u - v)V^{-1}(u - v)^T} \tag{5}$$

where V^{-1} is the inverse of the covariance matrix. By using MDs, the difference between query patterns ($\rho(O^q)$) and weighted FDs ($\mathbf{O}^{lhs}$) can be considered in terms of the difference between the vectors of u and v relative to their variance.

4.4 Selecting FDs

Instead of using all possible rewrite strategies from the large set of FDs, FDSel uses the properties previously described to focus on meaningful FDs, which we call exemplar FDs. Because FDSel uses FDs for semantic query optimization, FDSel focus on exemplars that integrate as much coalescence of attributes as possible, while producing the best gains in query optimization. We formulate three different strategies for selecting exemplars, described next.

4.4.1 *Selecting FDs based on their rank.* FDSel first sorts the set of discovered FDs F using Algorithm 1. The algorithm requires as input a set of FDs, and two weighted AOMs: $\mathbf{O}^{lhs}$ and $\mathbf{O}^{rhs}$. As pointed out earlier, entries $\mathbf{o}_{ij}$ of **skewed_sort**($\mathbf{O}$) are converted into the attributes of FDs, *lhs* or *rhs*, by mapping the attributes A_j of $\mathcal{R}$ for each element of $\mathbf{o}_{ij}$ other than zero. In other words, entries $\mathbf{o}_{ij}$ represent valid *lhs* and *rhs* in F. Thus, for each distinct *lhs* in F (Line 3), the algorithm iterates over each distinct *rhs* (Line 4) to find a combination that builds a valid f in F (Line 5). The combination $lhs \rightarrow rhs$ is appended to the ranked sequence of FDs F' (Line 7) only if such combination is a valid FD in F (Line 5). Notice that because the iteration over F is based on the weighted AOMs, with the **skewed_sort** function, the first FDs to be appended in the result F' are those in which the target attributes are the most accessed by the application query workload.

Algorithm 1: Ranking FDs

```
   Data: Set of discovered FDs F, weighted AOM O^lhs, and weighted AOM O^rhs
   Result: Ranked FDs (F')
1  begin
2      F' ← { }
3      foreach lhs ∈ skewed_sort(O^lhs) do
4          foreach rhs ∈ skewed_sort(O^rhs) do
5              if (lhs, rhs) build a valid FD f in F then
6                  f = lhs → rhs
7                  F' ← {F'} + f
```

Algorithm 1 returns the same number of FDs as in the initial set F. Because the result F' is a sorted sequence, FDSel can iterate through F' until the FDs f in F' stop meeting some desired criteria. We noticed that the quality measures of FDs degrade as this iteration occurs. Thus, FDSel estimates the distrust and MD against the current workload of each FD f, and builds two sets of exemplar FDs. FDSel outputs the first set of exemplars by considering the following criterion: (1) iterate through F' until there is a harsh increasing in distrust. The second set of FDSel is built with the following criterion: (2) iterate through F' until there is a harsh increasing in MD against the current query workload. We consider that there is harshness when an element in F' shows a quality measure that is higher than the double of the median of previous elements seen in the iteration up to that point. Finally, the set of exemplars are extended with inference rules [Beeri et al. 1984] before they are used in semantic query optimization.

4.4.2 *Clustering FDs with affinity propagation algorithm.* FDSel uses clustering in the third strategy to select FDs. To this purpose, we adapted the affinity propagation (AP) clustering algorithm [Frey and Dueck 2007] to work with AOMs, and cluster FDs based on their weighted structures. Unlike other clustering algorithms (e.g. k-means), AP does not require the number of clusters to be specified a priori. In addition, AP clustering algorithm can be applied for data that does not lie in a continuous space or data with non-symmetric similarities. The AP clustering algorithm identifies the most representative elements in a set by recursively transmitting messages between pairs of elements until convergence. An acceptable set of exemplars (corresponding clusters) is selected when the message-passing procedure is done. That happens when a fixed number of iterations is reached or after the changes in the messages either fall below a threshold or remain constant for some iterations.

The inputs of the AP algorithm are measures of similarity between pairs of data points, which FDSel extracts from the weighted AOMs. Consider two elements $\mathbf{o}_i$ and $\mathbf{o}_j$, $\mathbf{o}_i \neq \mathbf{o}_j$, of AOM $\mathbf{O}^{lhs}$. FDSel uses MD as the similarity measure for AP inputs, and estimates the MD between the *lhs* structures of pairs of FDs. There are two categories of messages exchanged between pairs $[\mathbf{o}_i, \mathbf{o}_j]$. The first message is called responsibility $r(\mathbf{o}_i, \mathbf{o}_j)$, which measures the accumulated evidence that $\mathbf{o}_j$ should be the exemplar for $\mathbf{o}_i$. Formally, the responsibility is given as in Equation 6.

$$r(\mathbf{o}_i, \mathbf{o}_j) \leftarrow md(\mathbf{o}_i, \mathbf{o}_j) - \max_{\forall \mathbf{o}'_j \neq \mathbf{o}_j} \left\{ \alpha(\mathbf{o}_i, \mathbf{o}'_j) + md(\mathbf{o}_i, \mathbf{o}'_j) \right\} \tag{6}$$

The availability α of an element $\mathbf{o}_j$ to be the exemplar of $\mathbf{o}_i$ is given as in Equation 7.

$$\alpha(\mathbf{o}_i, \mathbf{o}_j) \leftarrow \min \left\{ 0, r(\mathbf{o}_j, \mathbf{o}_j) + \sum_{\mathbf{o}'_i \text{ s.t. } \mathbf{o}'_i \notin \{\mathbf{o}_i, \mathbf{o}_j\}} \max \{0, r(\mathbf{o}'_i, \mathbf{o}_j)\} \right\} \tag{7}$$

Responsibility r and availability α are initially zero, and all $\mathbf{o}_i$, $\mathbf{o}_j$ equally represent a potential exemplar FD. At any time of AP, measures r and α can be combined to identify exemplars FDs. Responsibility iteration lets all elements $\mathbf{o}_i$ compete for ownership of another $\mathbf{o}_j$, and availability iterations choose evidences for every other element $\mathbf{o}_i$ as to whether each candidate exemplar would make a satisfying exemplar FD.

FDSel iterate through F to find the corresponding *rhs* of the AP output. Additionally, this set of exemplars FDs is also extended with inference rules [Beeri et al. 1984].

4.5 Semantic Query Optimization

Once we have detailed the FDs selection process of FDSel, we present a potential scenario where the selected FDs can improve the overall query performance. We use the approach presented in [Simmen et al. 1996] and [Chakravarthy et al. 1990], nevertheless, our tool can be easily extended to work with others dependency-aware optimization schemes like [Laurent and Spyratos 2011] and [Paulley and Larson 2010]. A semantic query optimization must rewrite incoming queries into syntactically different, yet semantically equivalent queries, which are expected to produce a more efficient execution plan. The rewritten queries are semantically equivalent if and only if their results are the same as the original query, regardless of the state of the database [Chakravarthy et al. 1990].

Rewritings could be blocked for particular queries according to the trade-off between optimization time and the quality of the execution strategies. As noted by [Shekhar et al. 1988], semantic optimization increases the search space of possible plans and, as a result, depends upon efficient searching techniques to keep optimization costs within reasonable bounds. FDSel is totally decoupled from any RDBMS query optimizer, and its first and foremost goal is to select suitable FDs for optimization. Thus, the incoming queries are only rewritten when they fall into two well-defined classes. FDSel uses the exemplars FDs to carry out two classes of semantic query optimization commonly discussed in

the literature [Simmen et al. 1996; Chakravarthy et al. 1990; Laurent and Spyratos 2011; Paulley and Larson 2010]: join elimination (JE) and order optimization (OO).

The JE technique iterates over the set of FDs to find residual clauses in the query. In this case, residuals clauses are joins for which the result is known a priori (empty or redundant joins) and, therefore, could be removed from the query. Consider a relation $\mathcal{R} = \{X, Y, Z\}$, and a FD $f : X \rightarrow Y$ holding in an instance r of $\mathcal{R}$. The relation $\mathcal{R}$ can be decomposed as $\mathcal{R}' = \pi_{(X,Y)}(\mathcal{R})$, and $\mathcal{R}'' = \pi_{(X,Z)}(\mathcal{R})$. This lossless-join decomposition is used to target queries where no attributes are selected or projected from the $\mathcal{R}''$ relation. The JE optimization is already implemented in some commercial RDBMS [Cheng et al. 1999]. However, these implementations require the users to explicit declare the set of constraints. Thus, automating this task may be beneficial in environments where users access views defined over a large number of joins (e.g., a star schema in a data warehouse). Further details and more complex JE optimizations can be found in [Chakravarthy et al. 1990] and [Cheng et al. 1999].

The goal of OO is to find optimal interesting sorting orders, that is, the sequence of the attributes in the order specification. Interesting sorting orders are carefully considered by query planners to reason about operations that accept hashing or ordering. They may emerge when tables are joined or when tuples are ordered, grouped or distinguished. The reduce order algorithm presented in [Simmen et al. 1996] takes as input a set of FDs, a set of applied predicates, and sorting orders specifications to return an optimized sorting order specification. Consider an FD $f : X \rightarrow Y$ holding on relation instance r, and a query $q = \tau_{X,Y}\left(\pi_{(X,Y)}(\mathcal{R})\right)$ running on r. The query q can be rewritten as $q' = \tau_{X}\left(\pi_{(X,Y)}(\mathcal{R})\right)$ because there is only one value of Y for each X. More examples and details on OO can be found in [Simmen et al. 1996] and [Szlichta et al. 2013].

5. EXPERIMENTAL STUDY

In this section, we present an experimental study to evaluate the effectiveness of FDSel.

5.1 Scenario

In preliminary experiments, we found that using FDs for semantic query optimization provides compelling gains in environments where very large relations are vertically partitioned (e.g., column-stores in data warehouses). In practice, there are many reasons why partitioning may be required. For example, database administrators might fragment a relation into a set of smaller relations to reduce maintenance costs, or to cope with distributed designs where applications use some fragments more frequently than others (e.g., invisible joins in column-stores [Abadi et al. 2008]). Another example is automatic normalization (e.g, to Boyce-Codd Normal Form)[Papenbrock and Naumann 2017], which uses FDs to eliminate redundancies and anomalies introduced as the dataset grow. Regardless the reasons for table partitioning, typical mechanisms to reconstitute information from partitions include *views*. Views can be defined using arbitrarily complex queries that blindly join partitions in order to present the user with a representation of the original table, with potential restrictions. The users access may be limited to the defined views (maybe through a query manager interface), therefore, redundant joins or residual sorting order operations are likely to occur. We stick to this view scenario to present our experimental evaluation.

5.2 Datasets and Implementation Details

Datasets. We use both synthetic and real-world datasets, which come from different domains. Table II lists these datasets with their number of attributes, number of records, number of discovered FDs, and number of exemplars selected according to the three selection strategies of FDSel. The datasets *Abalone* and *Adults* have been used for FD discovery evaluation in [Papenbrock et al. 2015]. The

Table II: Description of the datasets, number of FDs, number of exemplars FDs with FDSel.

Dataset	#Columns	#Records	#FDs	#FDs with FDSel - Criterion 1	#FDs with FDSel - Criterion 2	#FDs with FDSel - AP
Abalone	9	4,177	137	6	6	10
Adults	14	48,842	78	9	3	5
SIMMC	12	2m	32	5	3	8
Lineitem	16	6m	4k	8	101	23

Adults dataset is based on census data for US citizen salaries. The *Abalone* dataset consists of clinical data about patients and diseases. In addition, we use a 2-week snapshot of data extracted by *SIMMC*, a brazilian project from the Ministry of Communications [Possamai et al. 2014] [1]. SIMMC dataset has about 2M records with a total size of nearly 300MB. Finally, we use the *lineitem* relation of the business oriented synthetic TPC-H dataset, set for 1GB scale.

Implementation details. Our experiments were executed on a single Linux machine with a 2.60 GHz Quad Core i7-3720QM processor, 8GB of RAM, 500GB 7200rpm SATA II disk, and Java 1.8. Our prototype was implemented as a Java client connecting to a PostgreSQL server via JDBC.

FDSel discovers the set of FDs holding on each dataset and stores the results in a buffer. After the discovery, we use a tool called *Normalize* [Papenbrock and Naumann 2017] to decompose the original dataset into BCNF-conformed set of tables. We supervise the results of *Normalize* to avoid semantically incorrect partitions. During our experiments, this partitioning step generated between three and six tables for each dataset. These tables are randomly joined to build the set of views in which queries run.

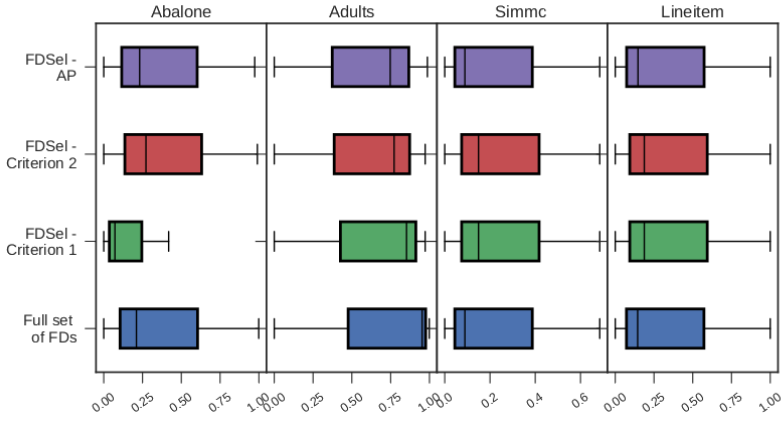
We execute select-project-join (SPJ) queries and select-project-join with group by (SPJGB) queries over the views, which are chosen at random. To choose the range of filter predicates, we equally divide the domain of each attribute according to the number of queries N to be executed. If the number of distinct values in the attribute domain is less than N, we assume the sequence of the closest pairs of values in the domain. For these cases, overlapping query predicates is required. Predicate ranges are chosen using a zipfian distribution [Barahmand and Ghandeharizadeh 2013] on the number of queries N . We also follow a zipfian distribution to choose attributes for selections, projections, and grouping. We vary the number of attributes in each of these operations according to the number of attributes in the view. Finally, we use the same distribution configuration to generate a thousand queries for the training workload and a hundred queries for performance evaluation with semantic query optimization. We use the above procedures to run FDSel set for either JE or OO, and we report the JE and OO performance results separately.

The training workload and the FDs buffer comprise the input of FDSel. The *core* component selects the exemplars of FDs and prepares the query mediator for optimizations. The *query mediator* is conditioned to the semantics of each query. If the set of operations and attributes required to evaluate the query fall into rules conforming to JE or OO (based on the set of exemplars FDs), it rewrites the query, otherwise, it bypasses the rewriting process.

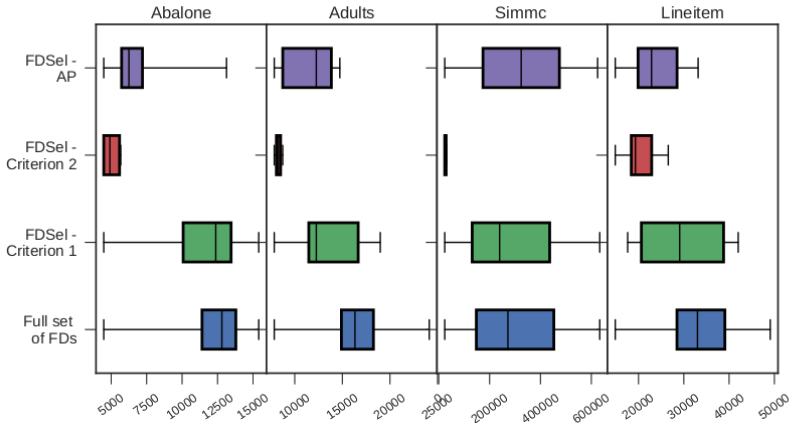
5.3 Effectiveness

Selecting FDs is subjective to a combination of factors (e.g., application, schema-level structures, and instance-level information), and the number of discovered FDs are usually too large for manual inspection. Thus, we cannot properly define a gold standard containing a complete set of absolute FDs. Instead of reporting recall and precision measures, we evaluate the exemplar FDs based on

[1] http://simmc.c3sl.ufpr.br/



(a) Levels of distrust



(b) *Mahalanobis* distances between FDs and workload

Fig. 2: Quality of exemplar FDs: the bottom boxes represent the distributional trends for the initial set of FDs; the remaining boxes represent the distributional trends of the exemplar FDs returned by FDSel.

their quality according to the measures described in Section 4, and their suitability for semantic query optimization.

We estimated the quality measures for the sets of exemplars FDs returned by: Algorithm 1, pruned with Criterion 1 (increase in distrust); Algorithm 1, pruned with Criterion 2 (increase in MD); and AP clustering algorithm. We refer to this results as FDSel - Criterion 1, FDSel - Criterion 2 and FDSel - AP, respectively. In addition, we estimated the quality measures for the initial set of discovered FDs to form a baseline. For each FD, we estimated its *distrust* level and its MD from the query workload. Figure 2 shows the distributional characteristics of the quality measures in a box-and-whisker plot. Each box divides the measures estimated for a set of FDs into quartiles to illustrate their degree of concentration and range. The bottom boxes and whiskers (we refer to them as bases) show the concentration and range of the measures for the set of initial FDs and serve as reference point for assessing which way the results from FDSel sway.

In general, the results from FDSel procedures were fairly close to that of the bases for *distrust* (Figure 2a). Particularly, FDSel - Criterion 1 presented more pronounced gains only for Abalone, where distribution measures are thicker and closer to the lowest values. For other datasets and strategies, FDSel causes slightly alterations at the center quartiles. In a deeper analyses, we have found that

many FDs exhibit similar levels of *distrust*. These FDs form groups that are easily distinguished by their attributes (e.g, FDs with many attributes in common at their *lhs*). Because we rank the set of FDs regarding structural frequencies (*lhs* and *rhs*) weighted over the workload, similar FDs are likely to be sorted into close spots at the sequence. However, the lack of a single attribute at their structure may cause *distrust* to dramatically change.

As can be seen in Figure 2b, the distributions for MD reveal much more pronounced variations. That is because the initial set of FDs present different levels of correlation to the query workload, and, surely, because FDSel uses different strategies to select exemplars FDs. For Criterion 1, FDSel may start discarding relevant FDs sooner than other procedures (e.g, contrast between *distrust* and MD in Abalone).

As expected, FDSel - Criterion 2 produced the best MD distributions. It softens the *distrust* barrier from FDSel - Criterion 1 and focus on the MD of each FD. FDSel - Criterion 2 was able to produce distance measures that concentrate towards lower values (all quartiles groups spread themselves to the first half of the distribution). Notably, it was the most effective procedure when the number of original FDs was relatively small. For SIMMC, all exemplars exhibit distance measures that are close to the lower tail of the distribution. Nevertheless, if the number of FDs is higher, the distances for the set of original FDs approximate normal distributions (e.g, Lineitem). Because FDSel - Criterion 2 selects exemplars that are more likely to fall closer to minimum values for MD, it may disregard groups of FDs with higher distances but also higher semantics (e.g, high number of correlated attributes at the *lhs*). This might occur if FDs have higher number of attributes. Because of their weighted equivalence, FDs that are structurally wider may increase the likelihood of larger MDs.

Criterion 1 and 2 may become over-judicious for some base distributions and discard relevant FDs. It is important for the selection task to achieve a parsimony between the number of exemplars and the semantics they expose because such characteristic is compelling in the optimization phase. The distributions for FDSel - AP suggests that the exemplars have good levels of agreement with the workload, leaning reach and distributions toward the first half of the base (except by SIMMC dataset). Interestingly, the exemplars for the SIMMC produced more uniform distributions if compared to the base. Though the original set of FDs had just few distance measures concentrated at the fourth quartile, FDSel - AP was able to select exemplars from it. This was only possible because of the intrinsic characteristic of the AP algorithm in combination with the *Mahalanobis* distance. As described in Section 4, the AP algorithm simultaneously considers any FD in the original set as a possible exemplar. Because AP refines this large set by exchanging similarity messages between its elements (FDs), it was crucial to choose a distance measure that could capture the semantic aspects of an FD along with the workload closeness. With MD, the similarities between pairs of FDs in the space are defined by the weighted attributes.

Because MD accounts for unequal variances and correlations between the weighted attributes, it estimates the distances by assigning different influence factors to the attributes in each FD. Differently from Criterion 2, the selection with AP not only considers distance values, but also considers how much set of attributes are correlated. Figure 3 represents the overall behavior of AP applied over the FDs of *lineitem* relation. Notice that it is possible for exemplars to be responsible for representing distant data points. That is why AP was able to select exemplars from spread locations in the distribution but at the same time shortening the range of distances with the workload.

Table II reports the number of all discovered FDs, and the number of exemplars selected by FDSel. As we shall see in the next experiment, a high number of exemplars does not necessarily mean better optimizations and, therefore, does not guarantee higher gains in query performance.

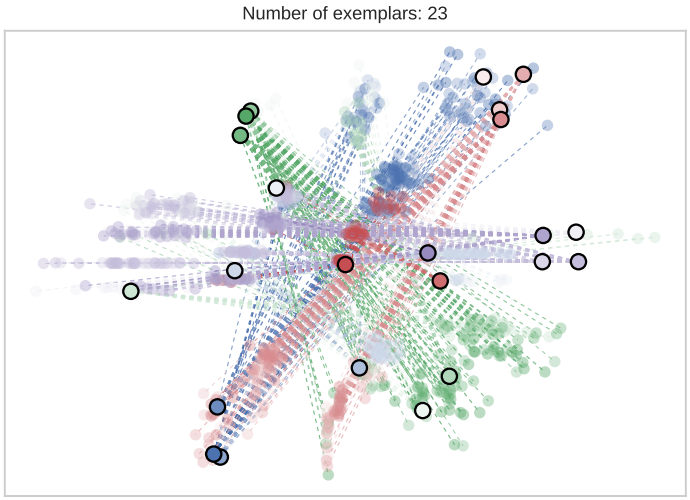


Fig. 3: Behavior of FDSel - Affinity Propagation over Lineitem dataset. Dimensions were reduced with Principal Component Analysis for better visualization.

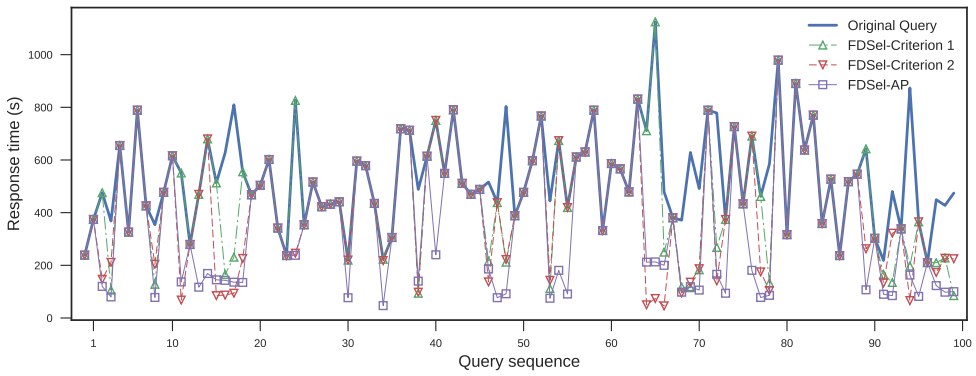
5.4 Performance improvement with semantic query optimization

In this experiment, we investigate the performance improvements of using FDSel for semantic query optimization. Figures 4a and 4b illustrate the implication of JE and OO optimizations for queries running over *lineitem*. The execution time remains unchanged for some queries because FDSel could not find any rewrite strategy for them. However, improvements of more than an order of magnitude can be viewed for many queries.

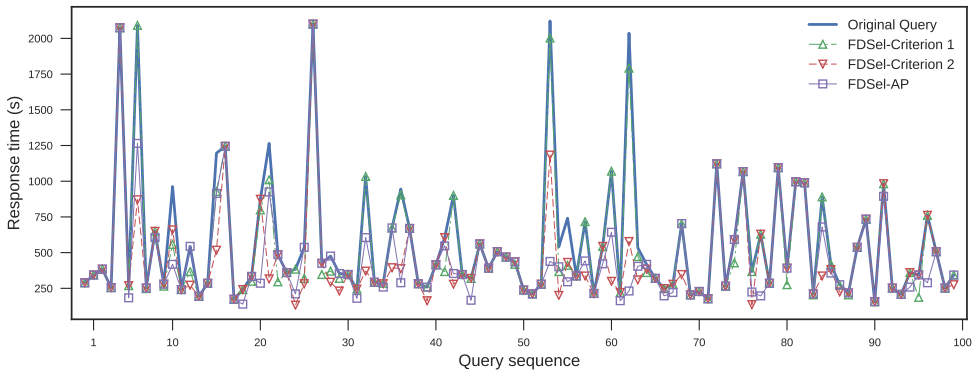
As expected, JE showed the most significant reductions in execution time for best-cases. For example, a particular query over *lineitem* reached a 12-fold improvement with FDSel - AP. Particularly, FDSel - AP presented higher improvements for larger datasets (SIMMC and Lineitem) because for some predicates the queries produced intermediary results that does not fit in main memory. Although OO produced more moderate improvements, the number of queries that benefited from rewriting was more consistent. For example, there were many queries in SIMMC that reached 4 to 6-fold improvements. Tables III and IV details the average improvements with JE and OO, respectively. On average, approximately one-third of the workload (for all datasets) was able to take advantage of some rewriting rule. Compared with the normal execution, FDSel was able to reduce the average execution time by nearly half in many cases. Notably, some of the best improvements occurred when the number of exemplars available was among the smallest (SIMMC and FDSel - Criterion 2). This strongly confirms our hypothesis that focusing on the information implied by the context usage (e.g, query workload) is much more effective than necessarily considering the largest number of FDs to obtain the best results. For example, FDSel - AP over the queries in *lineitem* presented the best performance even though it relied on less than a quarter of the number of exemplars of FDSel - Criterion 2.

6. CONCLUSION AND FUTURE WORK

Correlations and dependencies structures among data and attributes permeates databases, and, whenever possible, should be exploited in data management tasks. Although several commercial solutions present facilities to unite not enforced constraints (such as FDs) into planning phases, we can not



(a) Join Elimination



(b) Order Optimization

Fig. 4: Example of improvements in query execution time with FDSel over lineitem.

Table III: Performance improvements with FDSel set for join elimination (JE)

Dataset	Normal Execution (Avg)	FDSel - Criterion 1 (Avg)	FDSel - Criterion 2 (Avg)	FDSel - AP (Avg)
Abalone	22ms	18ms	17ms	15ms
Adults	209ms	152ms	136ms	123ms
SIMMC	22.90s	15.79s	12.03s	13.44s
Lineitem	531.33s	383.51s	344.75s	297.10s

Table IV: Performance improvements with FDSel set for order optimization (OO)

Dataset	Normal Execution (Avg)	FDSel - Criterion 1 (Avg)	FDSel - Criterion 2 (Avg)	FDSel - AP (Avg)
Abalone	40ms	38ms	24ms	24ms
Adults	367ms	320ms	220ms	237ms
SIMMC	62s	54s	35s	29s
Lineitem	571s	487s	387s	360s

expect them to be wisely exploited in query plans without human supervision. In this paper, we

presented FDSel, an effective automatic tool for selecting FDs in relational databases. FDSel is based on the idea of matching FDs with the current workload to boost query optimization. First, we model attribute occurrence matrices (AOMs) with the FDs and the workload information. We provide operations over the AOMs to estimate weights over each matching. Then, we present strategies to investigate this matching: (1) ranking FDs that match most of the projections/selections in the query stream (i.e., workload); and (2) clustering FDs on their *lhs* structure, with only the most representative matching elements set as exemplars. Next, we compute the distance between binary relationships of FDs and workload to focus on well ranked FDs (by the ranking strategy) or similar ones (by the clustering strategy). Finally, we indicate the focused exemplars in hand to help with semantic query optimizations. The results of both ranking and clustering strategies shown that, with respect to dependency selection, FDSel was able to choose sets of FDs that produced *distrust* distributions reasonably similar to those produced by the exhaustive FDs discovery approaches. The results also demonstrated the effectiveness of FDSel at discovering FDs on different datasets (one of them running in production) for query optimization, frequently reducing query response time in up to 1 order of magnitude in join elimination.

Despite new efforts in automatic discovery of dependencies and new ways to explore query optimization, lots of ground remains unexplored. The future work shall concentrate on making the selection process more generic, thus, exploring other types of dependencies and other workload-related use for them. In addition, we intend to use focusing strategies to improve the dependency discovery because it is distinctly the most time consuming step in FDSel.

REFERENCES

Abadi, D. J., Madden, S., and Hachem, N. Column-stores vs. row-stores: how different are they really? In *SIGMOD 2008, Vancouver, BC, Canada*. pp. 967–980, 2008.

Abedjan, Z., Akcora, C. G., Ouzzani, M., Papotti, P., and Stonebraker, M. Temporal rules discovery for web data cleaning. *Proc. VLDB Endow.* 9 (4): 336–347, Dec., 2015.

Abedjan, Z., Golab, L., and Naumann, F. Profiling relational data: A survey. *The VLDB Journal* 24 (4): 557–581, Aug., 2015.

Amavi, J. and Halfeld Ferrari, M. pp. 83–113. In A. Hameurlain, J. Küng, and R. Wagner (Eds.), *Maximal Set of XML Functional Dependencies for the Integration of Multiple Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 83–113, 2014.

Barahmand, S. and Ghandeharizadeh, S. D-zipfian: A decentralized implementation of zipfian. In *Proceedings of the Sixth International Workshop on Testing Database Systems*. DBTest '13. ACM, New York, NY, USA, pp. 6:1–6:6, 2013.

Basak, J., Wadhwani, K., and Voruganti, K. Storage workload identification. *Trans. Storage* 12 (3): 14:1–14:30, May, 2016.

Beeri, C., Dowd, M., Fagin, R., and Statman, R. On the structure of armstrong relations for functional dependencies. *J. ACM* 31 (1): 30–46, Jan., 1984.

Chakravarthy, U. S., Grant, J., and Minker, J. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.* 15 (2): 162–207, June, 1990.

Cheng, Q., Gryz, J., Koo, F., Leung, T. Y. C., Liu, L., Qian, X., and Schiefer, K. B. Implementation of two semantic query optimization techniques in db2 universal database. In *25th VLDB*. VLDB '99. pp. 687–698, 1999.

Chiang, F. and Miller, R. J. Discovering data quality rules. *Proc. VLDB Endow.* 1 (1): 1166–1177, Aug., 2008.

Chu, X., Ilyas, I. F., and Papotti, P. Discovering denial constraints. *Proc. VLDB Endow.* 6 (13): 1498–1509, Aug., 2013.

Demetrovics, J., Katona, G. O. H., Miklós, D., and Thalheim, B. On the number of independent functional dependencies. In *4th FoIKS*. FoIKS'06. Springer-Verlag, Berlin, Heidelberg, pp. 83–91, 2006.

Frey, B. J. and Dueck, D. Clustering by passing messages between data points. *Science* vol. 315, pp. 972–976, 2007.

Hammer, M. and Zdonik, S. B. Knowledge-based query processing. In *Proceedings of the Sixth International Conference on Very Large Data Bases - Volume 6*. VLDB '80. VLDB Endowment, pp. 137–147, 1980.

Hazewinkel, M. *Encyclopaedia of Mathematics (1)*. Springer, 1987.

Ilyas, I. F., Markl, V., Haas, P., Brown, P., and Aboulnaga, A. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*. SIGMOD '04. ACM, New York, NY, USA, pp. 647–658, 2004.

Kimura, H., Huo, G., Rasin, A., Madden, S., and Zdonik, S. B. Correlation maps: A compressed access method for exploiting soft functional dependencies. *Proc. VLDB Endow.* 2 (1): 1222–1233, Aug., 2009.

Kivinen, J. and Mannila, H. Approximate inference of functional dependencies from relations. *Theoretical Computer Science* 149 (1): 129 – 149, 1995.

Laurent, D. and Spyratos, N. Rewriting aggregate queries using functional dependencies. In *International Conference on Management of Emergent Digital EcoSystems*. ACM, New York, NY, USA, pp. 40–47, 2011.

Liu, J., Li, J., Liu, C., and Chen, Y. Discover dependencies from data - a review. *IEEE Trans. on Knowl. and Data Eng.* 24 (2): 251–264, Feb., 2012.

Liu, Y., Liu, H., Xiao, D., and Eltabakh, M. Y. Adaptive correlation exploitation in big data query optimization. *The VLDB Journal* 27 (6): 873–898, Dec, 2018.

Mazuran, M., Quintarelli, E., Tanca, L., and Ugolini, S. Semi-automatic support for evolving functional dependencies. In *Proceedings of the 19th EDBT, Bordeaux, France, 2016*. pp. 293–304, 2016.

Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.-P., Schönberg, M., Zwiener, J., and Naumann, F. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proc. VLDB Endow.* 8 (10): 1082–1093, June, 2015.

Papenbrock, T. and Naumann, F. A hybrid approach to functional dependency discovery. In *SIGMOD*. SIGMOD '16. ACM, New York, NY, USA, pp. 821–833, 2016.

Papenbrock, T. and Naumann, F. Data-driven schema normalization. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*. pp. 342–353, 2017.

Paulley, G. N. and Larson, P.-A. Exploiting uniqueness in query optimization. In *CASCON First Decade High Impact Papers*. CASCON '10. IBM Corp., Riverton, NJ, USA, pp. 127–145, 2010.

Possamai, C. L. B., Pasqualin, D., Weingaertner, D., Todt, E., Castilho, M. A., de Bona, L. C. E., and de Almeida, E. C. Proinfodata: Monitoring a large park of computational laboratories. In *Open Source Software: Mobile Open Source Technologies*, L. Corral, A. Sillitti, G. Succi, J. Vlasenko, and A. I. Wasserman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 226–229, 2014.

Romero, O., Calvanese, D., Abelló, A., and Rodríguez-Muro, M. Discovering functional dependencies for multidimensional design. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*. DOLAP '09. ACM, New York, NY, USA, pp. 1–8, 2009.

Shekhar, S., Srivastava, J., and Dutta, S. A formal model of trade-off between optimization and execution costs in semantic query optimization. In *14th VLDB*. VLDB '88. San Francisco, CA, USA, pp. 457–467, 1988.

Simmen, D., Shekita, E., and Malkemus, T. Fundamental techniques for order optimization. *SIGMOD Rec.* 25 (2): 57–67, June, 1996.

Szlichta, J., Godfrey, P., Gryz, J., and Zuzarte, C. Expressiveness and complexity of order dependencies. *PVLDB* vol. 6, pp. 1858–1869, 2013.

Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H., and Urata, M. Answering complex sql queries using automatic summary tables. *SIGMOD Rec.* 29 (2): 105–116, May, 2000.