Robust Voice Querying with MUVE: Optimally Visualizing Results of Phonetically Similar Queries

Ziyun Wei
Cornell University
Ithaca, NY, USA
zw555@cornell.edu

Immanuel Trummer
Cornell University
Ithaca, NY, USA
itrummer@cornell.edu

Connor Anderson
Cornell University
Ithaca, NY, USA
ca339@cornell.edu

ABSTRACT

Recently proposed voice query interfaces translate voice input into SQL queries. Unreliable speech recognition on top of the intrinsic challenges of text-to-SQL translation makes it hard to reliably interpret user input. We present MUVE (Multiplots for Voice quEries), a system for robust voice querying. MUVE reduces the impact of ambiguous voice queries by filling the screen with multiplots, capturing results of phonetically similar queries. It maps voice input to a probability distribution over query candidates, executes a selected subset of queries, and visualizes their results in a multiplot.

Our goal is to maximize probability to show the correct query result. Also, we want to optimize the visualization (e.g., by coloring a subset of likely results) in order to minimize expected time until users find the correct result. Via a user study, we validate a simple cost model estimating the latter overhead. The resulting optimization problem is NP-hard. We propose an exhaustive algorithm, based on integer programming, as well as a greedy heuristic. As shown in a corresponding user study, MUVE enables users to identify accurate results faster, compared to prior work.

PVLDB Reference Format:
Ziyun Wei, Immanuel Trummer, and Connor Anderson. Robust Voice Querying with MUVE: Optimally
Visualizing Results of Phonetically Similar Queries. PVLDB, 14(11): 2397 - 2409, 2021.
doi:10.14778/3476249.3476289

1 INTRODUCTION

Voice interfaces are popular, as evidenced by the rise of devices and services such as Google Home, Amazon Alexa, or Apple's Siri. They provide a particularly natural way to interact with computers and enable hands-free interaction. This has recently motivated systems that enable relational databases for voice access, including for instance EchoQuery [19], CiceroDB [34], SpeakQL [31, 32], and approaches for voice-based OLAP [6, 36], among others.

Voice query interfaces (VQIs) typically built on prior work on natural language querying [10, 13, 17, 18, 25, 28, 29, 45]. Here, the goal is to translate natural language text into corresponding SQL queries. Despite significant recent advances, text-to-SQL translation is a hard problem. The intricacies of natural language as well as similarly named database elements lead to ambiguities in query interpretation. This ambiguity translates to VQIs which built on the latter. On top of that, VQIs rely on speech recognition which is notoriously difficult. As established in prior studies [2], this makes query interpretation for VQIs very hard.

Example 1. At the time of writing, issuing the query "OK Google, what's the population in New York?" via voice input on the Web site www.google.com yields the population count for New York City. However, ambiguity between New York City and New York state is not addressed. The approach presented in this paper might show results for both query interpretations (city and state). This avoids the need for repeated queries or additional input if the most likely query interpretation is incorrect.

Prior work on natural language and VQIs typically requests user feedback to resolve ambiguities. For instance, users may provide feedback on candidate queries [17] or select query fragments [2, 7]. Alternatively, users may get asked specific clarification questions [19] (e.g., in case of columns with similar names). All of those methods have in common that users need to provide additional input, costing them time. We explore a complementary approach to resolve ambiguity in voice querying. Instead of resolving ambiguities with the help of the user, we try to display results for all of the most likely query interpretations. In doing so, we hope to reduce disambiguation time for users. Our approach is implemented in the MUVE system (Multiplots for Voice quEries).

MUVE relies on existing components for speech recognition and to translate input text into a probability distribution over queries. The primary research challenge we address in MUVE is the automated design of result multiplots. We formalize the generation of the result output as an optimization problem. Our search space is constrained by the screen resolution and minimal requirements on font sizes and plot space. This means that we can only fit a limited number of plots and data points onto the output screen. The goal of optimization is to maximize the probability that the correct result is shown on the output screen. In addition, the user time for finding the correct query result in the visualization should be minimized.

Accurately estimating time until users find specific results is, of course, challenging. We conduct a user study with crowd workers to obtain a corresponding cost model. In our study, we analyze the impact of visualization features such as color and positioning on user response time. Based on our results, we formulate the visualization optimization problem. The search space is the space of multiplots, distinguished by results shown as well as other properties (e.g., color) of the visualization. The objective function is based on our user model and estimates expected time overheads.

The optimization problem becomes challenging due to constraints between plots and queries. Each plot presents results for a subset

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476289

of query candidates. Each query result is represented as one bar in such a plot (we consider aggregation queries that result in a single numerical result). The y axis of a plot measures the result quantity. The x axis varies a query property. For instance, we may vary the aggregation function on the x axis. Alternatively, we may vary the constant used in one specific query predicate. Besides the query property varied on the x axis, each plot contains results for a fixed query template. In doing so, we avoid having to associate each data point with a complete SQL query (which would require disproportional amounts of space). It motivates however a judicious choice of plots to display. For instance, it may not always be best to display the single, most likely query result. Instead, it may be better to choose a plot that can contain results for a large number of likely queries, whose accumulated probability is dominant.

Multiplot selection problem is NP-hard (as we show in our analysis). Hence, we cannot obtain guaranteed optimal solutions efficiently (unless $P = NP$). MUVE features two solvers for this problem. The first one translates a multiplot selection problem instance into an integer linear program. It uses corresponding solver software to obtain an optimal solution (which is then translated into a visualization). In addition, MUVE features a greedy heuristic. This algorithm does not guarantee an optimal solution. However, it typically generates near-optimal solutions fast.

MUVE processes several alternative query interpretations. For large data sets, this can lead to significant overheads. We describe multiple strategies to mitigate such overheads, either by sharing work between similar queries or by presenting results incrementally. In our experiments, we evaluate different multiplot selection algorithms as well as different approaches to reduce processing overheads. We also perform user studies in which we evaluate user satisfaction and compare MUVE to a baseline.

Our original, scientific contributions are the following.

- We propose the MUVE system, aimed at enabling robust voice querying despite ambiguities due to noisy speech recognition and text-to-SQL translation.
- We introduce the problem of multiplot selection. The goal is to cover alternative query interpretations, each associated with a likelihood, by an optimal result visualization.
- We present and formally analyze two algorithms to solve the multiplot selection problem.
- We present multiple approaches to reduce the impact of overheads when processing multiple query interpretations.
- We compare the performance of our algorithms experimentally and perform user studies, validating our user model and comparing MUVE against a baseline.

The remainder of this paper is organized as follows. We formally introduce our problem model, and associated terminology, in Section 2. Next, we give an overview of the MUVE system in Section 3. We discuss user studies by which we established a user model, as well as the resulting model, in Section 4. The optimization problem, solved by MUVE, is based upon that model. In Section 5, we describe how to map multiplot selection to integer programming. In Section 6, we describe a corresponding greedy algorithm. Then, in Section 7, we analyze our algorithms and properties of our optimization problem. We discuss strategies to mitigate query processing overheads, caused by processing multiple queries, in Section 8. Finally, we report experimental results, comparing our two solvers and comparing MUVE to baselines, in Section 9.

2 FORMAL MODEL

We introduce our problem model and related terminology.

Definition 1. *A **Candidate Query** is a query into which the voice input can possibly translate. It is associated with a probability, indicating the confidence of the system that this query represents the user's intent accurately. In the context of this work, we focus on queries (e.g., aggregates) that produce one single, numerical output.*

The goal of MUVE is to cover alternative query interpretations by a single visualization. This visualization is a multiplot which we define more formally in the following.

Definition 2. *A **Query Group Plot** (or simply **Plot** in the following) visualizes results for a group of similar queries. Those queries are similar in the sense that they instantiate a common query template with placeholders. In the visualization, the plot title references the template while labels on the x-axis reference concrete values for the placeholders. E.g., placeholders may substitute constants in predicates but also operators or aggregation functions. We consider bar plots for which a subset of bars may be highlighted with a markup color (red).*

Definition 3. *A **Multiplot** consists of plots, structured into one or multiple rows. Each row contains one or multiple plots (according to the previous definition).*

Primarily, we are concerned with finding optimal multiplots. Optimality is defined with regards to the following metric.

Definition 4. ***User Disambiguation Time** is the time it takes users to find the result for the correct interpretation of their voice query. For a fixed input, disambiguation time is a function of the selected multiplot. We present a model for disambiguation time, based on user studies, in Section 4.*

Now, we present the problem that this paper focuses on.

Definition 5. *Given a set of candidate queries (with their probabilities), a maximal number of rows and the screen width, the goal of **Multiplot Selection** is to find a multiplot containing results for some of the candidate queries, respecting the dimension constraints (i.e., the number of rows and screen width), while minimizing disambiguation time according to a given model.*

3 SYSTEM OVERVIEW

Figure 1 shows an overview of the MUVE system. MUVE enables voice-based access to a relational database. It answers voice queries with a multiplot, capturing results for the most likely query translations. The system will be demonstrated at the upcoming SIGMOD'21 conference [40]. Next, we discuss components of MUVE (some of which are shown in Figure 1) in more detail.

Voice Query (Input). MUVE supports voice queries on a relational database. Currently, MUVE supports SQL aggregation queries with predicates on a single table that produce a single, numerical result. The result of each such query can be represented as one data point in a corresponding plot. Users formulate their queries in natural language. This means that user input needs to be translated into corresponding SQL queries (a process that leads to ambiguities).

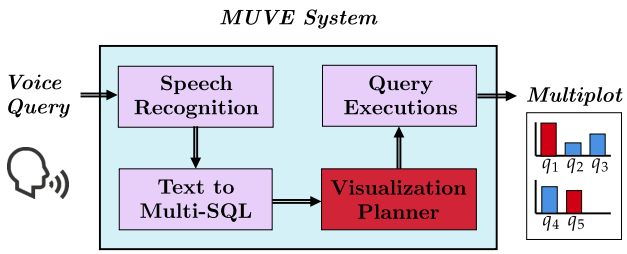

Figure 1: Overview of the MUVE system: voice queries are mapped to a probability distribution over queries, the visualization planner (our research focus) determines an optimal multiplot covering results for the most likely queries

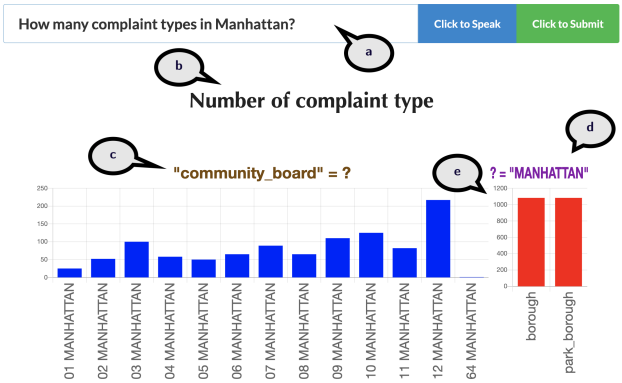

Figure 2: MUVE allows users to speak (or type) natural language queries (a). The resulting multiplot contains results for similar queries whose common elements are outlined in the headline (b), while covering specific templates in specific plots (c, d). Results of likely query interpretations are marked up in red (e).

Multiplot (Output). MUVE answers voice queries by showing a multiplot. A multiplot consists of several bar plots, each of them showing results for different query candidates. We arrange plots in multiple rows. Each plot is associated with a query template that is shown as plot title. The template contains one placeholder (e.g., the value of a constant in a query predicate). Values on the plot x axis are associated with different substitutes for the placeholder (e.g., different values for the predicate constant). Plot data points correspond to results of query candidates, covering different interpretations of the user input. Figure 2 shows an example output of our current prototype.

Speech Recognition. MUVE is targeted at voice queries. In a first step, user voice input needs to be transcribed to text. For that, MUVE uses the browser-based Web Speech API[1].

Text to Multi-SQL. Users describe their queries in natural language. Hence, we need to translate text into corresponding SQL queries. Typically ("text to SQL"), the goal is to translate input into one single query (whose result is displayed). MUVE's output covers multiple alternative query interpretations. Therefore, we translate input into a probability distribution over candidate queries instead ("text to multi-SQL").

We generate candidate queries in multiple steps. First, MUVE uses sequence-to-sequence translation to map text input to a most likely query. More precisely, we use the recently proposed SQLova approach [13]. Next, we take into account uncertainties due to noisy speech recognition and uncertain text to query translation. We generate query variations by replacing query fragments by phonetically similar alternatives. Specifically, we iterate over all schema element names and constants that appear in the query. We use a functionality offered by Apache Lucene[2] to find the k most phonetically similar entries for each query element (typically, we set k to 20). Candidate queries are derived from the original query (raw output of text to query translation) by replacing elements with those alternatives. Finally, we assign probabilities to the different query candidates. The probability of a single replacement is based on a distance function that measures phonetic similarity between text fragments. More precisely, we map query and database elements to a phonetic representation using the Double Metaphone algorithm [24]. Then, we use the Jaro-Winkler [8] distance to calculate similarity. The probability of multiple replacements corresponds to the product of probabilities for single replacements. Note that our algorithms for visualization planning, discussed next, could be used with more sophisticated candidate generation algorithms.

Visualization Planner. Our research focus is on the visualization planner. The goal of visualization planning is to generate a multiplot that optimally covers the set of candidate queries. A candidate query is covered, if one of the result plots contains that query's result. More formally, the visualization planner obtains a set Q of candidate queries with probabilities $r(q)$ for $q \in Q$ as input. The result of a query q can be shown in one or several plots $P(q)$, covering templates with placeholders that match query q. Each plot p is associated with minimal plot dimensions $m(p)$, determined for instance by the plot title. Adding more data points to a plot increases the (horizontal) width proportionally. Furthermore, the visualization planner obtains the screen resolution, together with the desired number of plot rows, as input. The goal of optimization is to select a visualization that minimizes expected overheads for the user (we present a corresponding cost model, based on user studies, in Section 4). Optionally, query processing overheads can be considered during optimization as well (see Section 8).

Query Executions. After selecting queries for visualization, those queries are executed to obtain their results. MUVE does not execute different candidates independently but merges similar queries together, e.g. via group by clauses, to reduce overheads.

4 USER BEHAVIOR MODEL

We conducted a user study to find out how visualization features influence time for finding desired results. We report the study results and propose a simple user model, consistent with those results.

[1]https://wicg.github.io/speech-api/
[2]https://lucene.apache.org/

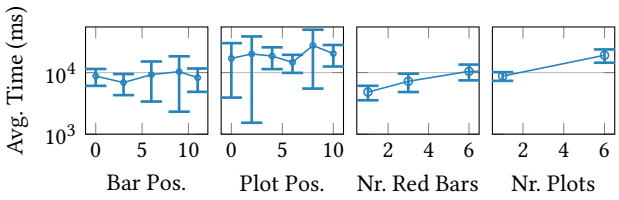

Figure 3: Average user perception time as a function of different multiplot visualization features.

Table 1: Results of Pearson correlation analysis.

Feature	Bar Pos.	Plot Pos.	Nr. Red Bars	Nr. Plots
R^2	0.050	0.079	0.24	0.39
p	0.72	0.6	0.00050	0.000052

4.1 User Study Results

In our study, we verify the following hypotheses. Note that we are interested specifically in linear relationships to keep the complexity of the model (and of the user study) reasonable.

Hypothesis 1. *Disambiguation time grows linearly in the target bar position within a plot (counting positions from left to right).*

Hypothesis 2. *Disambiguation time grows linearly in the target plot position within a multiplot (counting positions from left to right and from top to bottom).*

Hypothesis 3. *If highlighting specific results via color, including the correct result, disambiguation time grows linearly in the number of highlighted results.*

Hypothesis 4. *Disambiguation time grows linearly in the number of plots shown in the multiplot.*

We conducted a user study on the Amazon Mechanical Turk (AMT) crowd-sourcing platform. Each Human Intelligence Task (HIT) links to an online version of the MUVE interface. Each link leads directly to a multiplot (without asking users to submit queries first). Crowd workers were asked to read a query description, stating the aggregate as well as a list of column-value pairs (representing equality conditions), and to identify the associated result in the multiplot as quickly as possible. We generated queries by randomly selecting aggregates and columns and values for two equality predicates (with uniform distribution). We simulated ambiguity by including results of the 11 most phonetically similar queries, according to the metric described in Section 3, for a total of 12 results shown to users. We measure user disambiguation time as follows. The timer starts once the visualization loads in the Web browser. The timer stops once crowd workers click on the bar representing, in their opinion, the result of the target query.

We submitted 520 HITs in total (26 types of tasks, each of which was available to 20 crowd workers), varying the bar position within a plot, the plot position within a multiplot, and the number of bars colored. We paid 10 cents per HIT. We received submissions for 262 out of 520 tasks within the allocated time of six hours.

Figure 3 shows average disambiguation time as a function of multiple visualization features. From left to right, we report time as a function of the correct bar position (within a single plot with 12 bars), time as a function of the correct plot position (within a multiplot containing 6 plots with two bars in two rows), time as a function of the number of red bars (one of the red bars represents the correct results), and time as a function of the number of plots in the multiplot. Table 1 shows the result of an associated Pearson correlation analysis.

Based on p values, assuming the common cutoff of $p = 0.05$, only the number of plots and the number of red bars have a statistically significant relationship with user disambiguation time. This means that Hypothesis 3 and Hypothesis 4 are validated while we cannot find enough evidence to support Hypotheses 1 and 2. Hence, we base our model on the former but not on the latter.

4.2 Derived Time Model

We present a simple model for user disambiguation time, based on the results from the previous subsection. Our model considers three factors. First, we consider which results are visualized. Second, we consider the number of plots in which they are visualized. Third, we consider which of the bars are highlighted with a markup color (red). Note that we do not consider position of bars and plots (i.e., we model the visualization as a set of plots and each plot as a set of bars). This is consistent with our user study results.

Primarily, our model distinguishes three cases. First, the correct result may be present and colored in red. Second, the correct result may be present but not colored in red. Third, the correct result may not be present in the current visualization. Next, we outline how to estimate cost in each of the three cases.

We introduce the following notation. By p and p_R with $p_R \leq p$, we denote the number of plots containing at least one red bar. By b and b_R with $b_R \leq b$, we denote the total number of bars and the number of red bars. We assume that users read plots and bars in random order (we assume that each permutation is equally likely). Also, we assume that users first focus on red bars (which requires reading and understanding their context, i.e. the semantics of the containing plots, as well). If the correct result is not found after studying red bars, users turn to the non-highlighted bars. We denote by c_P the cost of understanding a plot and by c_B the cost of reading a bar (based on our results, it is $c_P > c_B$). We infer the values for those constants from our user study results.

If the correct result is highlighted in red, the expected disambiguation time is time required for reading red bars and associated plots, until the right bar is found. As all permutations are equally likely, we expect to read half of the bars and of associated plots. Hence, if the correct result is marked up in red, we expect a disambiguation cost of $D_R = b_R \cdot c_B/2 + p_R \cdot c_P/2$. Otherwise, if the correct result is visualized but not colored up in red, we expect users to first read all red bars, then read half of the remaining bars in expectation (together with the associated plots). Hence, the associated cost is $D_V = 2 \cdot D_R + (b - b_R) \cdot c_B/2 + (p - p_R) \cdot c_P/2$. Finally, we consider the case that the correct result is not at all present in the visualization. In that case, the user first studies red and remaining bars, then concludes that the desired result is missing. In that case, the user must ask a new query, causing significant overheads. We

generally model the cost of a missing result by a large constant, D_M. Given probabilities r_R, r_V, and r_M with $r_R + r_V + r_M = 1$, representing probability of a highlighted, visualized, or missing result, the expected cost is given as $r_R \cdot D_R + r_V \cdot D_V + r_M \cdot D_M$.

5 INTEGER PROGRAMMING SOLVER

In the following, we show how to transform multiplot selection into integer linear programming. After that transformation, we can apply existing solvers to find an optimal solution.

5.1 Decision Variables

Our cost model does not consider the position of a bar within a plot. Also, it does not consider the position of a plot within a multiplot. Hence, it is sufficient to introduce decision variables, modeling which bars and plots are visualized (but no the precise position). While position does not matter for the cost model, we still need to make sure that plots fit on the screen with a given resolution. For that purpose, we keep track of the row (within the multiplot) in which each plot is placed.

We are given a set of candidate plots. Each plot is characterized by a query template and can display a subset of query results. We introduce binary variables of the form p_i^j, indicating whether the i-th plot is shown in multiplot row number j (if so, $p_i^j = 1$, otherwise $p_i^j = 0$). For each plot, we can choose which query results to show within it. We introduce binary variables of the form $q_{i,j}^k$ to indicate whether the result of query candidate i is shown in plot j in row k. We introduce those variables only for pairs of queries and plots that are "compatible" (i.e., the query instantiates the query template associated with the plot). Furthermore, we have the choice to highlight specific results. We introduce binary variables of the form $h_{i,j}^k$ to indicate whether query i in plot j, shown in row k, is marked up in red.

5.2 Constraints

First, we can only assign query results to plots that are on display. We introduce constraints of the form $q_{i,j}^k \leq p_i^j$ to capture this fact. Also, we can only highlight query results that are on display. This is expressed via constraints of the form $h_{i,j}^k \leq q_{i,j}^k$. It is not useful to display the same result multiple times in a multiplot. We introduce constraints of the form $\sum_{j,k} q_{i,j}^k \leq 1$ to capture this fact.

We use plots of equal height and limit the number of rows, in accordance with the vertical screen resolution. Hence, no additional constraints are required to avoid exceeding the screen height. However, we must avoid exceeding the horizontal dimensions.

We introduce constraints for each row to limit the width. We represent the width of the i-th plot (without any bars) by constants W_i. We assume that each bar has the same width and assume, without restriction of generality, that this width is one (we can scale W_i accordingly). For each row r, we introduce constraints of the form $\sum_i (p_i^r \cdot W_i) + \sum_{i,k} q_{i,k}^r \leq W$ where W is the screen width.

5.3 Objective Function

Our objective function is based on the user model, established in the last section. We assume that exactly one of the query candidates correspond to the correct interpretations. We can express expected cost, E, as a sum $E = \sum_i r_i \cdot E_i$. Here, r_i is the probability that the i-th query candidate is the correct interpretation and E_i is expected cost under that assumption. Note that r_i is a constant while E_i depends on the visualization. Next, we express E_i as a linear function.

There are three possible cases with regards to query i. First, it may not be shown in the visualization. Second, it may be on display but not highlighted. Third, it may be highlighted. We show how to calculate cost E_i for each of those mutually exclusive cases.

First, we consider the case that query i is not on display. In that case, the expected cost equals a constant D_M. We introduce auxiliary variable q_i, representing whether query i is on display in at least one row. Note the reduced set of indices, compared to our decision variable q_i^j (which distinguishes by row). We add the term $(1 - q_i) \cdot D_M$ to the expected cost E_i. Clearly, setting $q_i = 1$ is preferable to minimize the cost function. We must ensure that q_i can only be set to one if that is consistent with the assignments for variables q_i^j. We introduce constraints of the form $q_i \leq \sum_j q_i^j$ to express that fact.

Second, we consider the case that the result of query i is on display and highlighted. In that case, we assume that users read, in expectation, half of the other highlighted bars and half of the associated plots before discovering query i. First, we introduce auxiliary variables, capturing whether specific plots in specific rows contain at least one highlighted bar. We introduce binary variables s_i^j, indicating whether plot i in row j has at least some highlighted bars. For consistency, we add constraints of the form $s_i^j \leq p_i^j$ (capturing that plots must be on display to become eligible). We add constraints of the form $s_i^r \leq \sum_i h_{j,i}^r$ to ensure that plots without highlighted bars cannot qualify. Also, we add constraints of the form $s_i^r \geq \sum_i h_{j,i}^r / n_i$ where n_i represents the number of query results that can be displayed within plot number i. The latter constraint forces variable s_i^r to one if at least one result is highlighted.

Each query is shown (and highlighted) at most once on the screen. We introduce binary variable $h_i = \sum_{j,k} h_{i,j}^k$ to indicate whether query i is highlighted. If query i is correct and highlighted, the probability that users read any other highlighted query j first is 50%. We add terms of the form $\sum_j h_i \cdot h_j \cdot c_B/2$ to E_i. Note that we multiply two variables which is typically inadmissible for linear objectives. However, we multiply two binary decision variables. Such products can be easily linearized by introducing an auxiliary variable (e.g., introduce variable y for the product of binary variables $x_1 \cdot x_2$ and enforce $y \leq x_1$, $y \leq x_2$, and $y \geq x_1 + x_2 - 1$). We use the product notation, exclusively between binary variables and constants, in the following, without explicitly introducing auxiliary variables. Also, the probability that users study another plot, containing some highlighted bars first, is 50%. We add the term $\sum_{j,k} h_i \cdot s_j^k \cdot c_P/2$ to E_i to express that fact (here, c_P is the cost for reading one plot).

Finally, we consider the case that query i is on display but not highlighted. In this case, we assume that users read all red bars first, together with the associated plots. We introduce variable $d_i = q_i \cdot (1 - h_i)$, indicating whether query i is displayed but not highlighted. Now, we add the terms $\sum_j d_i \cdot h_j \cdot c_B$ and $\sum_{j,k} d_i \cdot s_j^k \cdot c_P$ to E_i (representing the cost of reading highlighted bars and plots first). For all other bars that are on display but not highlighted, the

probability that users read them before query i is again 50%. Hence, we add the terms $\sum_j d_i \cdot d_j \cdot c_B/2$ and $\sum_{j,k} d_i \cdot p_j^k \cdot (1-s_j^k) \cdot c_P/2$ to E_1 (accounting for the cost of reading other, non-highlighted bars and their associated plots).

5.4 Incremental Optimization

As we will see in more detail in Section 9, optimization can take non-negligible amounts of time. To reduce latency, MUVE supports incremental optimization for the integer programming approach. Here, optimization time is divided into short sequences. The i-th sequence has duration $k \cdot b^i$ for appropriately chosen constants k and b, i.e., MUVE uses an exponentially increasing timeout scheme (to reduce overheads associated with restarting optimization). After each optimization sequence, the resulting visualization is generated and shown to the user. Hence, users can see a first visualization early. We analyze in Section 9.5 whether this approach increases user satisfaction.

6 GREEDY SOLVER

We present a fast, greedy optimization algorithm as an alternative to the integer programming approach.

6.1 Problem Analysis

For the following analysis, we consider the cost model presented in Section 4. This model approximates user disambiguation time by the formula $r_R \cdot D_R + r_V \cdot D_V + r_M \cdot D_M$ (r_R, r_V, and r_M are probabilities of highlighting, displaying, or missing the correct query, D_R, D_V, and D_M the associated cost of doing so).

The following theorem is useful to restrict the search space for coloring choices in the multiplot. For its proof, we assume that all bars considered for highlighting are non-redundant (i.e., no other plot within the same multiplot is showing the same result).

Theorem 2. *Each plot in an optimal multiplot highlights results of the k most likely queries in it (for some $k \in \mathbb{N}$).*

Proof. We conduct a proof by contradiction. Assume one plot in the optimal multiplot shows results for two queries, q_1 and q_2 with probabilities r_1 and r_2. Assume the result of q_1 is highlighted but not the one for q_2 and that $r_1 < r_2$. We use $\delta = r_2 - r_1$ in the following. Assume we "swap" the coloring of q_1 and q_2 (i.e., q_2 is highlighted but not q_1). The terms in our cost formula will change as follows. The total number of highlighted and non-highlighted bars and plots do not change. Hence, D_R and D_V (which only depend on the latter) do not change either. On the other side, r_R increases by δ while r_V decreases by δ. Hence, overall cost changes by $\delta \cdot D_R - \delta \cdot D_V$. It is $D_V \geq D_R$ (since $D_V = 2 \cdot D_R + (b - b_R) \cdot c_B/2 + (p - p_R) \cdot c_P/2$). Hence, as $\delta > 0$, overall cost cannot increase by that change. Therefore, the initial plot was not optimal, contradicting our assumption. □

Next, we analyze cost savings by adding plots.

Definition 6. ***Cost savings*** *of a multiplot M is the difference in cost between an empty multiplot and M, according to our cost model (i.e., $C(\emptyset) - C(M)$ where $C(M)$ denotes cost of a set of plots).*

We make the following assumption.

Assumption 1. *Cost components D_R and D_V, denoting cost of reading highlighted and non-highlighted results respectively, are smaller than the cost D_M of a miss (i.e., $D_R < D_M$ and $D_V < D_M$).*

The latter assumption seems realistic. If the correct query is missing, users must ask a new voice query which takes significant amounts of time. This assumption implies the following.

Lemma 1. *Cost savings are non-decreasing in the set of plots.*

Proof. Denote by δr_R and δr_V the change in probability, after adding a plot, of having the correct result highlighted or visualized. Denote by ΔD_R and ΔD_V the changes in plot reading costs. Of course, it is $\Delta D_R \leq D_R$ and $\Delta D_V \leq D_V$. Hence, the total change to the cost savings function is at least $(\delta r_R + \delta r_V) \cdot (D_M - \max(D_V, D_R))$ which is positive due to Assumption 1. □

Next, we show that cost savings have diminishing returns. First, we define this property formally.

Definition 7. *A set function $f : S \mapsto \mathbb{R}$ is **submodular** if the following holds. If adding an element $s \in S$ to two sets $S_1 \subseteq S_2 \subseteq S$ then $f(S_1 \cup \{s\}) - f(S_1) \geq f(S_2 \cup \{s\}) - f(S_2)$.*

We analyze cost savings with regards to submodularity. We assume that C_M, the cost of not showing the correct query, is larger than either D_R (cost for reading highlighted bars and plots) or D_V (cost for reading non-highlighted bars and plots).

Theorem 3. *Disambiguation cost savings are sub-modular as a function of the set of selected plots.*

Proof. Consider a multiplot showing plots P_1 and another one showing plots P_2. Further, assume that $P_1 \subseteq P_2$ (i.e., the second multiplot contains a superset of plots, compared the the first one). Now, assume we add the same plot p to both, P_1 and P_2. We will analyze the relative change in cost savings. Denote by $\Delta Q_V(p, P)$ the set of queries whose results are visualized (but not highlighted) in a newly added plot p but not in any of the plots in P. Denote by $\Delta Q_R(p, P)$ the set of newly added query results that are highlighted. Now, we can express changes to different elements of the cost formula as follows. It is $\delta r_R = \sum_{q \in \Delta Q_R(p,P)} \Pr(q)$ (where $\Pr(q)$ denotes the likelihood of query candidate q) and $\delta r_V = \sum_{q \in \Delta Q_V(p,P)} \Pr(q)$. At the same time, it is $\delta r_M = -(\delta r_R + \delta r_V)$ (since all queries that are not highlighted or visualized must be missing).

Next, we compare the changes by adding p to P_1 and P_2. We use superscript 1 (e.g., δb_R^1) to denote parameters related to P_1 and superscript 2 for P_2. The number of added bars and plots does not depend on which plots are already present. Hence, the change in reading cost for highlighted (ΔD_R) and non-highlighted components (ΔD_V) does not depend on prior plots either. When adding p to P_i, the change in cost savings can be expressed as $\Delta C^i = D_M \cdot (\delta r_R^i + \delta r_V^i) - \delta r_R^i \cdot D_R^i - \Delta D_R \cdot (r_R^i + \delta r_R^i) - \delta r_V^i \cdot D_V^i - \Delta D_V \cdot (r_V^i + \delta r_V^i)$. Equivalently, it is $\Delta C^i = \delta r_R^i \cdot (D_M - D_R^i - \Delta D_R) + \delta r_V^i \cdot (D_M - D_V^i - \Delta D_V) - r_R^i \cdot \Delta D_R - r_V^i \cdot \Delta D_V$. However, it is $r_R^2 \geq r_R^1$ and $r_V^2 \geq r_V^1$ (since P_2 visualizes a superset of highlighted and non-highlighted results, compared to P_1). Also, it is $D_R^2 \geq D_R^1$ and $D_V^2 \geq D_V^1$ (since reading cost is monotone in the number of bars and plots). We have $\delta r_R^1 \geq \delta r_R^1$ and $\delta r_V^1 \geq \delta r_V^2$. At the same time, we assume that $D_M > D_R^i + \Delta D_R$ and $D_M > D_V^i + \Delta D_V$ (Assumption 1). Hence, it is $\Delta C_1 \geq \Delta C_2$, proving sub-modularity. □

Algorithm 1 Greedy algorithm for multiplot selection.

```
1:  // Greedily generates multi-plot with n rows for screen width w
2:  // covering results for query candidates Q.
3:  function GreedyViz(Q, n, w)
4:      // Generate uncolored plot candidates
5:      P ←PlotCandidates(Q, w)
6:      // Try different highlighting options
7:      C ←AddColors(P)
8:      // Select plots for multiplot
9:      M ←PickPlots(Q, n, w, C)
10:     // Final cleanup of multiplot
11:     F ←Finalize(M)
12:     // Return generated multiplot
13:     return F
14: end function
```

Algorithm 2 Generate set of plot candidates.

```
1:  // Given queries Q with probabilities, returns plot candidates.
2:  function PlotCandidates(Q)
3:      // Group queries by template
4:      T = ∅
5:      // Iterate over candidate queries
6:      for q ∈ Q do
7:          // Iterate over associated templates
8:          for t ∈ 𝒯(q) do
9:              T(t) ← T(t) ∪ {q}
10:         end for
11:     end for
12:     // Generate plots for query groups
13:     P ← ∅
14:     // Iterate over templates with queries
15:     for ⟨t, Q_t⟩ ∈ T do
16:         // Iterate over subsets of likely queries
17:         for S ⊆ Q_t : ∄s ∈ S, q ∈ Q_t \ S : Pr(q) > Pr(s) do
18:             P ← P ∪ {𝒫(S)}
19:         end for
20:     end for
21:     return P
22: end function
```

6.2 Greedy Algorithm

Algorithm 1 is the main function of our greedy algorithm. As input, it receives a set of candidate queries Q with associated probabilities, the number of multiplot rows, n, and the screen width w. The output is a greedily constructed multiplot (i.e., plots with assigned rows).

The algorithm executes four phases that are discussed in more detail in the following. First, it generates a set of candidate plots without highlighting. Those plots are characterized by the set of queries for which results are shown. Second, for each candidate plot, the algorithm generates multiple versions that differ by the subset of results that are highlighted. Those plots, characterized by a set of queries on display and by a subset of queries highlighted, form the basic building blocks of the multiplot. Typically, screen resolution is too limited to show all candidate plots at once. Hence, in a third step, Algorithm 1 picks a subset of plots to show. Finally, the algorithm "polishes" the multiplot, as explained later, to obtain the final version. Next, we discuss those four steps in more detail.

Algorithm 3 Highlight likely query results in plots.

```
1:  // Generate colored versions of uncolored plots P.
2:  function AddColors(P)
3:      C ← ∅
4:      // Iterate over uncolored plots
5:      for p ∈ P do
6:          // Iterate over sets of likely queries
7:          for S ⊆ p.Q : ∄s ∈ S, q ∈ p.Q \ S : Pr(q) > Pr(s) do
8:              // Add plot version with highlighted results
9:              C ← C ∪ {ColorPlot(p, S)}
10:         end for
11:     end for
12:     return C
13: end function
```

Algorithm 2 implements the first step (generating a set of candidate plots). As input, it obtains a set of candidate queries Q with associated probabilities. The output is a set of candidate plots (without highlighting). All queries within the same plot must instantiate a common query template. This template forms the title of the plot, while query-specific substitutions of placeholders form entries on the x-axis.

First, Algorithm 2 groups queries according to their query templates. Each query may instantiate multiple templates. More precisely, we obtain possible templates for a query by replacing constants or operators within a query with placeholders. For a given query, we introduce placeholders for a limited number of elements. Function $\mathcal{T}(q)$ denotes the templates derived for a query q. We maintain a set of associated queries for each template (denoted by $T(t)$ in Algorithm 2) that is updated while iterating over all queries.

Next, we generate multiple plots for each template with associated queries (we equivalently use T as function as well as a set of $\langle key, value \rangle$ pairs in the pseudo-code). For each template, we iterate over subsets of the most likely queries. Hence, we assume that, given space constraints, we prefer adding more likely queries. This is a heuristic as it neglects interactions with other selected plots. The resulting plot candidates are returned.

Algorithm 3 generates multiple colored versions for each plot candidate. We have proven in Theorem 2 that plots coloring the k most likely queries lead to an optimal solution. Hence, Algorithm 3 generates plot versions by highlighting the k most likely queries (considering different values of k).

Given candidate plots with highlighting, we must select a subset that is compatible with the screen dimensions. Theorem 3 demonstrates that cost savings are submodular in the set of selected plots. Hence, we use greedy algorithms from the domain of submodular maximization. Algorithm 4 describes a variant exploiting greedy algorithms for maximizing monotone, submodular functions, subject to multi-dimensional knapsack constraints (e.g., Yu et al. [42] propose a corresponding algorithm). Algorithm 4 transforms multiplot selection into submodular maximization as follows. It generates a set of items where each item is associated with the combination of a plot candidate and a multiplot row. Each row forms one dimension of the corresponding knapsack problem. Given an item representing a specific plot in a specific row r, the associated weight is zero for each dimension except for the one associated with row r (and weight is proportional to the plot width). Weight bounds are defined

Algorithm 4 Select subsets of plots for multiplot.

```
// Given queries Q, number of rows n, screen width w,
// colored plot candidates C, pick plots for multiplot.
function PickPlots(Q, n, w, C)
    // Generate plot-row combinations
    I ← ∅
    // Iterate over colored plots
    for p ∈ C do
        // Iterate over multiplot rows
        for r ∈ 1, ..., n do
            I ← I ∪ {⟨p, r, p.width · e_r⟩}
        end for
    end for
    // Submodular maximization under constraints
    return M ← arg max_{M⊆I} C(∅) − C(M) s.t. F(M, w)
end function
```

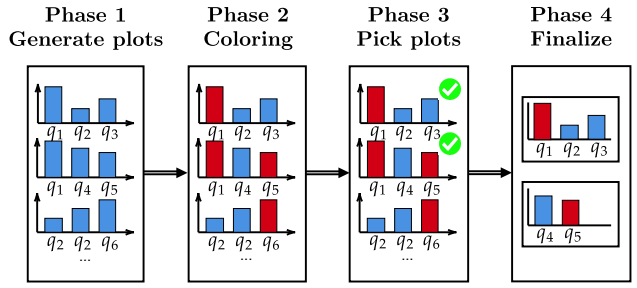

Figure 4: The greedy algorithm generates candidate plots, highlights results of likely queries, selects a near-optimal plot subset, and finally removes redundant query results.

by the screen width (predicate $\mathcal{F}$, used in Algorithm 4, is satisfied if and only if multiplot M is compatible with the screen dimensions). The function to maximize is cost savings, comparing cost of an empty multiplot to the ones of the current selection (represented as $C(\emptyset) - C(M)$ in the pseudo-code).

As a variant of the algorithm presented so far, we can fix the width of generated plot candidates. Then, the multidimensional knapsack constraint reduces to one single cardinality constraint (i.e., we limit the number of plots to select). Then, algorithms for cardinality-constrained, submodular maximization (e.g., the classical algorithm by Nemhauser [21]) can be used for picking plots.

Finally, we "polish" the multiplot resulting from the previous step (pseudo-code omitted). Here, we remove redundant results that appear in multiple plots. After removing a (redundant) result from a plot, we try to fill the resulting gap with the most likely, non-redundant query that fits into the corresponding plot. Figure 4 illustrates the four steps of the greedy algorithm.

6.3 Output Quality Analysis

We heuristically generate plot candidates by selecting the most likely queries from each query group. Relative to the best multiplot that can be constructed from those components, the greedy algorithm achieves the following guarantee (we assume that plots are picked using the algorithm by Yu et al. [42]).

Theorem 4. *The greedy algorithm achieves cost savings within $O(1/(1+2\cdot r)-\varepsilon)$ of the optimum, relative to the initial plot candidates.*

Proof. Algorithm 1 only considers a limited number of possibilities to highlight results in each plot. However, based on Theorem 2, the optimal solution is among them. Cost savings are sub-modular (see Theorem 3) and monotone under Assumption 1 (according to Lemma 1). Hence, the postulated guarantees derive directly from the guarantees offered by Yu et al. [42] (setting r instead of d in their equations). □

7 COMPLEXITY ANALYSIS

First, we analyze the complexity of multiplot selection.

Theorem 5. *Multiplot selection is NP-hard.*

Proof. Consider an instance of the knapsack problem, defined by a set I of items with weights b_i and utility u_i ($i \in I$), as well as a a weight bound B. We reduce to multiplot selection in polynomial time. For each item i, we introduce a plot p_i with one associated query candidate q_i (i.e., the query can be shown in that plot but in no other plots). We set the width of each plot with its bar, $W_i + 1$, proportional to b_i, the screen width W proportional to B. We set the probability of q_i, r_i, proportional to u_i. We assume that the cost of reading bars and plots is negligible (i.e., $c_B = c_P = 0$). We set the cost of a missing result, D_M, to one. No bars are highlighted. Therefore, the probability that the correct bar is highlighted is zero ($r_R = 0$). The probability to display the correct result is the sum of probabilities r_i (summing over query results shown in the selected plots). The optimal solution maximizes the probability sum, given width constraints. This is equivalent to maximizing the utility sum under weight constraints for the original knapsack instance. □

Next, we analyze complexity of the proposed algorithms. We denote by n_q the number of queries, by n_r the number of rows, and by n_p the number of candidate plots. For the ILP approach, we analyze how the ILP problem size evolves as a function of the multiplot selection instance.

Theorem 6. *The number of ILP variables for multiplot selection is in $O(n_p \cdot n_q \cdot n_r + n_q \cdot (n_q + n_p))$.*

Proof. Decision variables $q^k_{i,j}$ and $h^k_{i,j}$ reference combinations of plots, queries, and rows. Their number is in $O(n_p \cdot n_q \cdot n_r)$. We also introduce auxiliary variables, representing the product of two decision variables, for each combination of queries or queries and plots (i.e., $O(n_q \cdot (n_q + n_p))$). □

Theorem 7. *The number of constraints for multiplot selection is in $O(n_p \cdot n_q \cdot n_r + n_q \cdot (n_q + n_p))$.*

Proof. We introduce constraints for each decision variable $q^k_{i,j}$ representing combinations of queries, plots, and rows. We introduce constraints for auxiliary variables connecting query pairs. □

Finally, we analyze time complexity of the greedy algorithm from Section 6. We assume that the algorithm for submodular maximization by Yu et al. [42] is used internally. In addition to the previous notations, we use W to denote the horizontal screen resolution (measured as the number of bars) and ε denotes the tuning

parameter used to trade optimality for optimization time during submodular optimization.

Theorem 8. *Greedy multiplot selection has time complexity in* $O((n_q \cdot n_p \cdot (\log W)/\varepsilon)$.

Proof. Algorithm 2 iterates over combinations of queries and query templates, i.e. the number of steps is in $O(n_q \cdot n_p)$ (since n_p restricts the number of plots and, equivalently, query templates). Algorithm 3 may generate up to $O(n_q)$ colored versions for each uncolored plot. Its complexity is therefore in $I(n_q \cdot n_p)$. The algorithm by Yu et al. has a complexity of $O(\log W/\varepsilon)$ per input element and iterates once over the input [42]. Hence, the complexity of Algorithm 4 is in $O(n_q \cdot n_p \cdot (\log W)/\varepsilon)$. Post-processing can be implemented by iterating at most twice over all combinations of plots and queries (with a complexity in $O(n_q \cdot n_p)$). □

8 QUERY PROCESSING OVERHEADS

MUVE considers different interpretation of a voice input query. To show results for them in a multiplot, it must first process the associated queries. This creates additional processing overheads, compared to executing only the most likely query. Next, we discuss several mechanisms by which MUVE reduces those overheads.

8.1 Query Merging

MUVE processes multiple queries that are all possible interpretations of the same voice input. Those queries tend to be similar, creating opportunities to share processing tasks between them.

For a given multiplot, MUVE generally considers merging similar queries before query processing starts. More precisely, it merges queries on the same table with similar predicates. For instance, it replaces multiple equality predicates on the same column by a corresponding IN condition while adding result columns for each aggregate of the merged queries. To decide whether to merge, we use the cost model of the Postgres optimizer (which is used for data processing in the current MUVE version). Our current implementation does not support merging for more complex queries (e.g. queries with sub-queries). Existing techniques from the area of multiple query optimization [4, 27, 39] could be used to expand the scope of query merging.

In addition, MUVE can already consider processing costs and merging opportunities when selecting plots. For instance, among several multiplots with similar cost estimates according to the user cost model, MUVE can select one minimizing (estimated) processing overheads. Alternatively, MUVE can select plots under constraints on added processing overheads, compared to executing the most likely query alone.

MUVE supports this more proactive approach via its integer programming based solver. MUVE compares candidate queries to find out which of them could be merged into a single query. We introduce binary variables of the form g_i for each of those groups of queries. Those variables indicate whether the corresponding query group is processed ($g_i = 1$) or not. For each query, there may be multiple associated processing groups. We impose constraints of the form $q_i \leq \sum_{j \in G(i)} g_j$ to ensure that queries can only be selected for the multiplot if at least one of their associated groups is processed. Here, q_i indicates whether the i-th query is shown in the multiplot

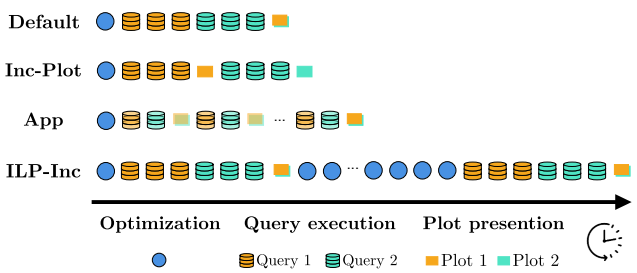

Figure 5: Overview of different processing methods.

and $G(i)$ denotes indices of all groups such that processing the group yields a result for query i. We obtain processing cost estimates for query groups via the Postgres explain statement. Now, we can estimate processing overheads by summing up groups, weighted by their processing cost estimates (i.e., the sum $\sum_i g_i \cdot c_i$ represents processing costs with c_i representing overheads for each group).

8.2 Progressive Presentation

We can reduce the impact of large processing overheads (rather than processing overheads themselves) by showing users visualizations of partial results early. We demonstrate that this approach can increase user satisfaction in Section 9.5.

In our scenario, partial results can be defined in two ways. First, a partial result may contain a subset of plots. Second, a partial result may contain approximate results for all plots in the multiplot. The first kind of partial results requires generating a subset of plots (and processing the associated queries). The second kind requires processing a data sample.

MUVE supports both strategies. The first strategy, called incremental plotting in the following, generates single plots sequentially. After each newly generated plot, the visualization is updated. Hence, users see plots appear sequentially on the screen (and may thereby obtain the result for the correct query even before the entire multiplot is generated). The second strategy, called approximate processing in the following, presents results for a data sample first. While users consider the approximate visualization, processing continues in the background on the full data set. The visualization is updated once the precise result is available. Figure 5 illustrates the different processing methods. It includes default processing (visualization after all queries are processed) and incremental optimization, described in more detail in Section 5.4.

9 EXPERIMENTAL EVALUATION

We evaluate the proposed methods experimentally.

9.1 Experimental Setup

All algorithms are implemented in Java 1.8. We run queries on the Postgres database system (version 13.1). We used Gurobi version 9 as integer programming solver, implementing the ILP described in Section 5[3]. For the following experiments, we use four data sets.

[3]For ease of exposition, we use slightly different auxiliary variables in Section 5, compared to our actual implementation. The asymptotic number of variables and constraints is however equivalent.

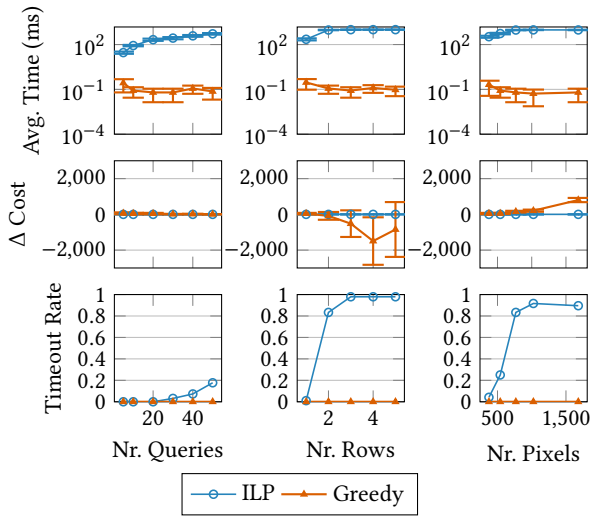

Figure 6: Solver performance on 311 request data.

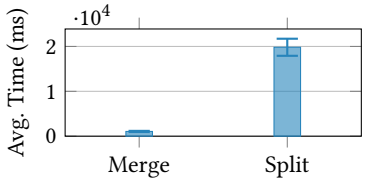

Figure 7: Impact of query merging on execution costs.

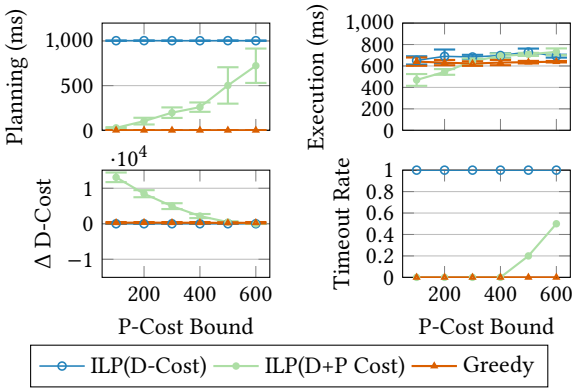

Figure 8: Disambiguation cost versus processing cost when varying processing cost bound.

The first one contains contacts for advertisement, provided by a partner in industry. The second one contains data of the department of buildings in NYC (DOB)[4]. The third one contains data describing NYC's 311 service requests[5]. Finally, we use a data set on flight delays[6], the largest one with a size of 10 GB. All of the following experiments are executed on a MacBook Pro with Intel Core i9 2.9 GHz CPU and 32 GB of main memory, running MacOS 10.15.7. We show 95% confidence bounds for all plots showing arithmetic averages in the following (confidence bounds are not applicable for plots showing counts and ratios).

9.2 Visualization Planning

We compare the greedy planner (described in Section 6) against the integer programming approach (described in Section 5). For each of our three data sets and each setting of the parameters we vary in the following, we generate 100 aggregation queries, randomly generating up to five equality predicates by randomly picking columns and constants. Then, we search for phonetically similar queries and plan a multiplot for the resulting candidates. We vary the number of candidate queries considered, the number of multiplot rows, and the number of pixels. For the latter parameter, we consider common screen resolutions, ranging from phones over tablets to typical computer screens. We use one row, 20 candidate queries, and a resolution of the iPhone as defaults. We set a timeout of one second for all compared approaches (as we target interactive data analysis). We measure optimization time, the ratio of timeouts, as well as the delta between cost of the proposed solutions, according to our cost model (the cost unit is estimated milliseconds of user disambiguation time).

Figure 6 shows results for the 311 service request data set. Clearly, the greedy algorithm is significantly faster and does not incur any timeouts. Integer programming, on the other side, generates better solutions over a range of scenarios. As the ratio of timeouts increases, the solutions of the greedy algorithm become preferable. Scalability is particularly limited in the number of rows. Already for three rows, the ratio of timeouts reaches nearly 100%. Note that, in case of a timeout, the ILP approach still produces a solution (which is however not guaranteed to be optimal anymore). On the other side, the ILP approach seems to scale quite well in the number of query candidates. Results for the other two data sets show the same tendencies and can be found in the extended technical report [40].

Altogether, ILP generates better multiplots for problems of relatively small dimensions. ILP scales quite well in the number of query predicates but less well in the number of rows. For larger problem instances, the greedy algorithm becomes preferable in terms of run time as well as in terms of solution quality.

9.3 Processing Cost

MUVE tends to execute many similar queries to cover alternative interpretations of the user's voice input. MUVE tries to reduce processing overheads by merging similar queries for execution, as opposed to executing them in separation. We performed a micro-benchmark using our largest data set (DOB with a size of 1 GB). We generated 10 queries randomly and retrieved the 50 phonetically most similar candidate queries for each of them. We executed those 50 queries once separately and once merged together. Figure 7 shows corresponding average results. Clearly, merging queries can reduce query processing overheads significantly.

Next, we analyze possibilities to influence processing overheads during visualization planning. We presented an extension of the ILP

[4]https://data.cityofnewyork.us/Housing-Development/DOB-Job-Application-Filings/ic3t-wcy2
[5]https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9
[6]http://stat-computing.org/dataexpo/2009/

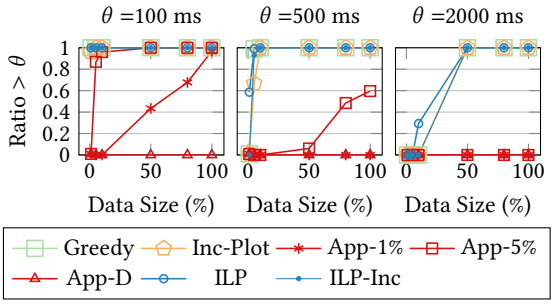

Figure 9: Non-interactive queries for presentation methods under different thresholds (θ) when varying data size.

approach that allows taking into account processing overheads. We can set a constraint on the added overheads of processing multiple query interpretations (as opposed to only the most likely one). In Figure 8 we vary that constraint and compare against the other planning approaches ("ILP(D-Cost)" and the greedy algorithm only consider user disambiguation cost). Each data point in Figure 8 represents an arithmetic average over 10 randomly generated queries (we set a resolution of 900 pixels).

First, we observe that tightening the constraint can indeed reduce execution costs by around 35.7%. This shows that MUVE can effectively limit overheads by multi-query execution. On the other side, disambiguation costs increase, the more visualization planning is restricted by processing cost considerations. Interestingly, optimization time increases as well for the processing-cost aware ILP approach. Tight processing cost constraints restrict the search space significantly, thereby making it easier to explore it.

9.4 Scaling Up Data Size

We test scalability of all proposed methods in the data size. We use samples from the data set on flight delays, scaling up sample size up to the full size of 10 GB. We generate queries by randomly selecting one aggregation column and one equality predicate (i.e., a random column and a random value with uniform distribution). As candidates, we use the 20 most similar queries according to our matching method. Figure 9 links combined optimization and processing time to the data size. The x-axis shows data size (as percentage of the full data size) while the y-axis shows the ratio of test cases (of 100 test cases per data point) for which the specified time threshold θ was reached. We use multiple values for θ.

We compare the following baselines. Greedy is the default approach (i.e., one single visualization is generated) with greedy optimization and reactive query merging. ILP is the default approach with ILP optimization, integrating processing cost into the optimization objective to favor multiplots with low estimated processing overheads among the ones minimizing the user cost model. ILP-Inc is the ILP approach with incremental optimization with $k = 62.5ms$ and $b = 2$ (see Section 5.4). Inc-Plot visualizes plots incrementally (see Section 8.2 while App visualizes an approximation first. App-1% and App-5% visualize results for a fixed sample size of 1% and 5% respectively. App-D dynamically estimates the sample size to use in order to meet the current interactivity threshold. For all methods generating a sequence of visualizations, we report time until the

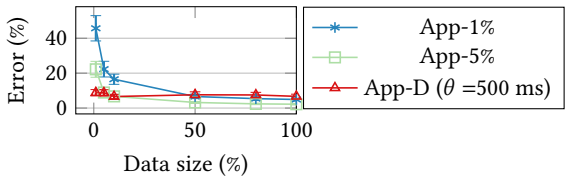

Figure 10: Relative error of initial multiplot for approximate processing methods.

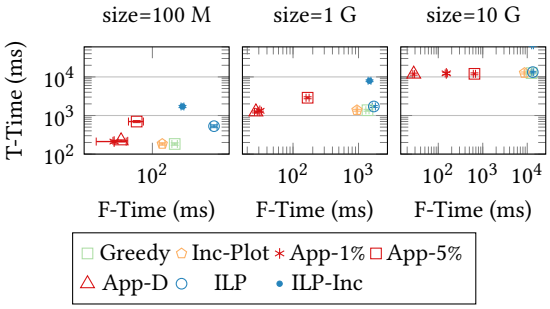

Figure 11: Time until correct result appears first (F-Time) versus total multiplot generation time (T-Time).

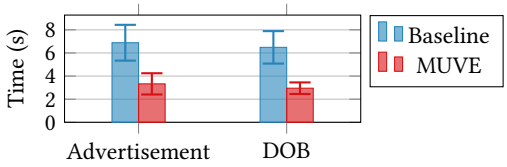

Figure 12: Average disambiguation time for users with MUVE, compared to baseline.

correct query results becomes visible (at least as an approximation) in Figure 9. The result quality, according to our user disambiguation cost model, was near-optimal for all compared methods (cost within 0.9% of the minimum for each test case).

Clearly, the ratio of test cases for which the interactivity threshold was reached increases for increasing data sizes and decreases with increasing threshold θ. While incremental plot visualization improves slightly over the other methods, only approximation can meet interactivity thresholds for large data sets.

Figure 10 analyzes relative error (average over all test cases) of the initial visualization for approximate processing. Clearly, the error is limited in particular for large data sizes. Figure 11 compares time until the correct result is shown first (F-Time) to time until the final visualization is generated (T-Time). For small data sizes, approximation increases T-Time noticeably. For large data sizes, the effect is negligible. This can be explained by per-query overheads that do not depend on data size (e.g., query parsing) but become dominated as data size grows. ILP-Inc has highest overheads for large data sizes as it implies repeated processing.

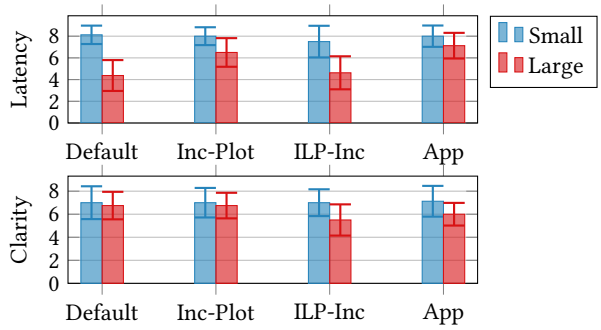

Figure 13: Average rating for latency and clarity by users in user study on small and large datasets.

9.5 User Studies

We conducted two user studies (both focused on comparing methods and therefore exempt from IRB requirements [3]). The majority of our participants are students outside of computer science (thereby representative for non-experts who may benefit most from natural language query interfaces). We assume that users issue voice queries in the browser (a feature offered by Web sites such as www.goooogle.com) via a desktop computer.

The first study compares MUVE (generating one multiplot via greedy optimization) to a baseline. We used Zoom and recruited 10 participants, nine of them college students with four CS students. Our baseline lets users resolve ambiguities by choosing correct columns and constants via a drop down menu (showing likely alternatives), inspired by systems such as DataTone [7]. We measured time after the voice query was processed and until the user verbally reports the query result. Each user issued 30 queries, 10 on each of the three aforementioned datasets, alternating between MUVE and the baseline (half of participants started with MUVE). We specified queries for users, generating each query by randomly selecting column and value for one equality predicate and an aggregation column. We discard the first 10 queries per participant (on the 311 request data) as warmup and report arithmetic averages for queries on advertisement and DOB data in Figure 12. Clearly, visually identifying the desired result is faster than resolving ambiguities by clicking buttons.

We conducted a second study comparing MUVE variants. We recruited 10 graduate students, two of them in computer science. For one large (flight delays) and one small (311 requests) data set, using one randomly generated query with one predicate per data set, we asked users to rate the approaches illustrated in Figure 5 on a scale from one to ten. Users were shown all visualization variants for the same query and data before submitting their rating. Figure 13 shows arithmetic averages for "latency" as well as "clarity". For latency, user satisfaction clearly decreases for the default approach once data size increases. Approximation receives statistically significantly better ratings, compared to the default approach, for large data sets. On the other side, the 95% confidence bounds overlap for all approaches when it comes to clarity (while ILP-Inc has the lowest average, likely due to a sequence of changing plots shown to the user). Overall, approximation seems preferable for large data sets.

10 RELATED WORK

Our work connects to prior work on natural language query interfaces [1, 17, 28]. Recently, advanced natural language processing processing methods, exploiting for instance sequence-to-sequence models [41, 45] or pre-trained language models [13, 16, 38], based on the Transformer architecture [37], have enabled significant improvements for text-to-SQL benchmarks such as WikiSQL [45] or SPIDER [43]. Nevertheless, text to query translation remains difficult [14] and subject to ambiguity. Voice-to-SQL interfaces [6, 19, 31, 35] exacerbate the problem due to noisy speech recognition. Prior work often requests additional input from users to resolve ambiguities [5, 7, 17, 19, 30, 31]. We compare against a corresponding baseline in our experiments. Other work aims at reducing ambiguity by considering more information (e.g., query logs [1]). Such approaches are complementary and can be combined with our approach (used to resolve remaining ambiguity after pruning). More generally, our work connects to prior work on visualization [15, 20, 22, 26, 33, 44], in particular work on visualization planning. However, our work differs by the goal of visualization planning (covering alternative query interpretations in voice query interfaces). To select multiplots, we propose a user cost model, based on user studies. Hence, to a moderate degree, our work connects to prior work studying human perception of visualizations [9, 11, 12, 23]. Our primary research contribution is however in the design and analysis of MUVE.

11 CONCLUSION AND FUTURE WORK

We presented an approach to deal with ambiguities in voice querying via multiplot visualizations. Our current implementation supports simple aggregation queries that yield a single number as result. Queries with multiple result rows and up to two numerical result columns (e.g., time series) could be plotted as lines or scatter plots. For more than two result columns per query, our visualization method would have to change fundamentally. We do not consider queries with entirely non-numerical results as they are typically not supported by data visualization tools. Our methods for visualization planning (Sections 5 and 6) and process optimization (Section 8) abstract away from the concrete structure of query templates. However, they use information on which queries can overlap work (see Section 8) or can be presented in the same plot (e.g., Section 5). Determining this becomes harder for complex query templates (a common problem with other areas [27]).

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback, notably for proposing several extensions to our original approach.

REFERENCES

[1] Christopher Baik, Hosagrahar V Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 374–385.
[2] Dharmil Chandarana, Vraj Shah, Arun Kumar, and Lawrence Saul. 2017. SpeakQL: towards speech-driven multi-modal querying. In *HILDA*. 1–6.
[3] Cornell University. 2021. https://researchservices.cornell.edu/sites/default/files/2019-05/IRB%20Decision%20Tree.pdf.
[4] Ahmet Cosar, E.P. Lim, and J. Srivastava. 1993. Multiple query optimization with depth-first branch-and-bound and dynamic query ordering. In *Information and*

Knowledge Management. 433–438. http://portal.acm.org/citation.cfm?id=170088.170181

[5] Kenneth Cox, Rebecca E Grinter, Stacie L Hibino, Lalita Jategaonkar Jagadeesan, and David Mantilla. 2001. A multi-modal natural language interface to an information visualization environment. *International Journal of Speech Technology* 4, 3-4 (2001), 297–314.

[6] Matteo Francia, Enrico Gallinucci, and Matteo Golfarelli. 2020. Towards conversational OLAP. *CEUR Workshop Proceedings* 2572 (2020), 6–15.

[7] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. 2015. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 489–500.

[8] Wael H. Gomma and Aly A. Fahmy. 2013. A Survey of Text Similarity Approaches. *International Journal of Computer Applications* 68, 13 (2013), 13–18.

[9] Connor C Gramazio, Karen B Schloss, and David H Laidlaw. 2014. The relation between visualization size, grouping, and user performance. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 1953–1962.

[10] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. (2019), 4524–4535. https://doi.org/10.18653/v1/p19-1444 arXiv:1905.08205

[11] Weidong Huang, Peter Eades, and Seok-Hee Hong. 2009. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Information Visualization* 8, 3 (2009), 139–152.

[12] Jessica Hullman, Eytan Adar, and Priti Shah. 2011. Benefitting infovis with visual difficulties. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2213–2222.

[13] Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069* (2019).

[14] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[15] Georgia Koutrika and Yannis Ioannidis. 2005. Constrained optimalities in query personalization. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 73–84.

[16] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 6943–6954.

[17] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 709–712.

[18] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. (2020), 4870–4888. https://doi.org/10.18653/v1/2020.findings-emnlp.438 arXiv:2012.12627

[19] Gabriel Lyons, Vinh Tran, Carsten Binnig, Ugur Cetintemel, and Tim Kraska. 2016. Making the case for Query-by-Voice with EchoQuery. In *SIGMOD*. 2129–2132.

[20] Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 438–448.

[21] GL Nemhauser and LA Wolsey. 1978. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research* 3, 3 (1978), 177–188. http://mor.journal.informs.org/content/3/3/177.short

[22] Deokgun Park, Steven M Drucker, Roland Fernandez, and Niklas Elmqvist. 2017. Atom: A grammar for unit visualizations. *IEEE transactions on visualization and computer graphics* 24, 12 (2017), 3032–3043.

[23] Robert E Patterson, Leslie M Blaha, Georges G Grinstein, Kristen K Liggett, David E Kaveney, Kathleen C Sheldon, Paul R Havig, and Jason A Moore. 2014. A human cognition framework for information visualization. *Computers & Graphics* 42 (2014), 42–58.

[24] Philips and Lawrence. 1994. The double metaphone search algorithm. *C/C++ Users Journal* 18, 6 (1994), 38–43. http://dl.acm.org/citation.cfm?id=349132

[25] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Ozcan. 2016. ATHENA: An ontology-driven system for natural language querying over relational data stores. *VLDB* 9, 12 (2016), 1209–1220.

[26] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350. https://doi.org/10.1109/TVCG.2016.2599030

[27] Timos K Sellis. 1988. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)* 13, 1 (1988), 23–52.

[28] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: natural language querying for complex nested SQL queries. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2747–2759. https://doi.org/10.14778/3407790.3407858

[29] Jaydeep Sen, Greg Stager, Chuan Lei, Fatma Ozcan, Ashish Mittal, Diptikalyan Saha, Abdul Quamar, Manasa Jammi, and Karthik Sankaranarayanan. 2019. Natural language querying of complex business intelligence queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2019), 1997–2000. https://doi.org/10.1145/3299869.3320248

[30] Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 365–377.

[31] Vraj Shah, Side Li, Arun Kumar, and Lawrence Saul. 2019. *SpeakQL: towards speech-driven multimodal querying of structured data.* Technical Report. 1–16 pages.

[32] Vraj Shah, Side Li, Kevin Yang, Arun Kumar, and Lawrence Saul. 2019. Demonstration of SpeakQL: speech-driven multimodal querying of structured data. In *SIGMOD Demo Track*. 2001–2004.

[33] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (2002), 52–65.

[34] Immanuel Trummer. 2019. Data Vocalization with CiceroDB. In *CIDR*.

[35] Immanuel Trummer. 2020. Demonstrating the voice-based exploration of large data sets with CiceroDB-zero. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2869–2872. https://doi.org/10.14778/3415478.3415496

[36] Immanuel Trummer, Yicheng Wang, and Saketh Mahankali. 2019. A holistic approach for query evaluation and result vocalization in voice-based OLAP. In *SIGMOD*. 936–953.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 2017-Decem, Nips (2017), 5999–6009. arXiv:1706.03762

[38] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942* (2019).

[39] Guoping Wang and Chee-Yong Chan. 2013. Multi-Query Optimization in MapReduce Framework. *VLDB* (2013), 145–156. https://doi.org/10.14778/2732232.2732234

[40] Ziyun Wei, Immanuel Trummer, and Anderson Connor. 2021. Demonstrating Robust Voice Querying with MUVE: Optimally Visualizing Results of Phonetically Similar Queries. In *SIGMOD*.

[41] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: generating structured queries from natural language without reinforcement Learning. 1–13. arXiv:1711.04436 http://arxiv.org/abs/1711.04436

[42] Qilian Yu, Easton Li Xu, and Shuguang Cui. 2017. Submodular maximization with multi-knapsack constraints and its applications in scientific literature recommendations. *2016 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2016 - Proceedings* (2017), 1295–1299. https://doi.org/10.1109/GlobalSIP.2016.7906050

[43] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2020. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018* (2020), 3911–3921. https://doi.org/10.18653/v1/d18-1425 arXiv:1809.08887

[44] Qianrui Zhang, Haoci Zhang, Thibault Sellam, and Eugene Wu. 2019. Mining precision interfaces from query logs. In *Proceedings of the 2019 International Conference on Management of Data*. 988–1005.

[45] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. (2017), 1–12. arXiv:1709.00103 http://arxiv.org/abs/1709.00103