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

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

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

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

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.

ACM Reference Format:
Ziyun Wei, Immanuel Trummer, and Connor Anderson. 2021. Demonstrating Robust Voice Querying with MUVE: Optimally Visualizing Results of Phonetically Similar Queries. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3452753

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 benefit population groups, such as visually impaired users, who cannot use traditional interfaces. This has recently motivated systems that enable relational databases for voice access, including for instance EchoQuery [8], CiceroDB [18], SpeakQL [15, 16], and approaches for voice-based OLAP [2, 19], among others.

Voice query interfaces (VQIs) typically built on prior work on natural language querying [4–7, 11, 13, 14, 21]. 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 [1], this makes query interpretation for VQIs very hard.

Prior work on natural language and VQIs typically requests user feedback to resolve ambiguities. For instance, users may provide feedback on candidate queries [6] or select query fragments [1, 3]. Alternatively, users may get asked specific clarification questions [8] (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.

In this demo, we explore a complementary approach to resolve ambiguity in voice querying. We consider scenarios in which users query by voice but obtain visual result output. This is suitable for devices that accept voice input but feature displays (e.g., cell phones, desktop computers, or the newest generation of smart speakers such as "Echo Show" and Google's "Smart Displays"). Instead of resolving ambiguities with the help of the user, we try to display results for all of the most likely query interpretations. This approach is implemented in the MUVE system (Multiplots for Voice quEries), the focus of our demonstration.

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 input is a set of **Candidate Queries** with associated probabilities. Each candidate corresponds to one possible interpretation of voice input. The output is a **Multiplot**, containing multiple rows of **Plots**. Each plot presents results for a set of queries instantiating the same query template. 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. Given the initial query probabilities, this probability corresponds to the sum of probabilities over all displayed query results.

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

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452753

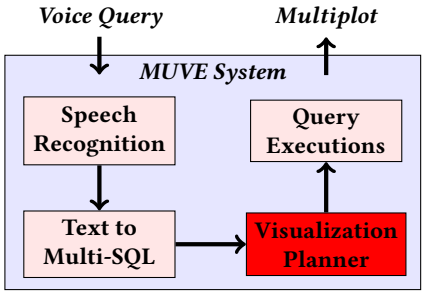

Figure 1: Overview of the MUVE system: voice queries are mapped to a probability distribution over queries, the visualization planner (our research focus) determines an optimal multiplot covering results for the most likely queries.

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.

The multiplot selection problem is NP-hard. This can be seen by a reduction from the knapsack problem. We can map any knapsack instance to a multiplot selection instance using one row of plots, considering only plots that each contain one single query result. In this transformation, knapsack weights map to (horizontal) plot dimensions, utility maps to probability, and weight constraints map to the resolution constraint. MUVE features two solvers (which can be selected on a per-query basis in the demo software). 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.

Our work connects to prior work on natural language query interfaces [4–7, 11, 13, 14, 21] and VQIs [2, 8, 15, 16, 18, 19]. However, unlike most prior work in this domain, MUVE does not request further user input to resolve input ambiguities. Instead, it tries to cover an optimal set of input interpretations via a single visualization. To do so, it uses cost-based planning. More broadly, our work connects to prior work on data visualization [9, 10, 12, 17, 20]. However, our work differs by its focus on visualizing alternative interpretations of voice queries.

In the remainder of this paper, we first give a more detailed overview of MUVE in Section 2. Next, we present an extract from our experimental results, including results of a user study, in Section 3. Finally, we describe our demonstration plan in Section 4.

2 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. 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).

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 (which is discussed in more detail in Section 4).

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].

Despite recent advances, automated speech recognition remains a challenging task. The challenges of speech recognition can be exacerbated by factors such as background noise, less common dialects, or low-quality microphones. Even under ideal circumstances, it may not be possible to distinguish certain spellings via speech input alone (e.g., "John" versus "Jon"). For those reasons, speech to text transcription is a first source of uncertainty with regards to query interpretation.

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 [5]. 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 10). Candidate queries are derived from the original query (raw output of text to query translation) by replacing elements with

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

those alternatives. Finally, we assign probabilities to the different query candidates (probabilities of all candidates sum to one). The probability of a single replacement is based on a distance function that measures phonetic similarity between text fragments. The probability of multiple replacements corresponds to the product of probabilities for single replacements. Note that we do not need to explicitly represent the set of query candidates (which can be large) for any of those steps.

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 is to generate a visualization maximizing $\sum_{q \in D} r(q)$ where $D \subseteq Q$ is the set of queries whose result is on display. This visualization must respect constraints imposed by plot dimensions and the resolution. We call this optimization problem "multiplot selection".

MUVE implements two approaches to solve multiplot selection: an approach based on integer programming and a greedy variant. Next, we discuss them in more detail.

Integer Programming Visualization Planner. Multiplot selection is NP-hard (we briefly sketch a corresponding reduction in Footnote 1). A common method to solve such optimization problems is to transform them into integer linear programming. This allows applying sophisticated solver tools targeted at that formalism.

Multiplot selection is based on the output of the text to multi-SQL component, as well as the screen dimensions. Our first approach to multiplot selection entails the following steps. First, we associate all queries with candidate plots in which their results could appear as data point. Also, we calculate minimal plot dimensions. Second, we transform the multiplot selection problem into an integer linear program. Third, we use the GLPK integer programming solver[3] to find an optimal solution (the solution may not be optimal only if optimization time reaches a user-specified timeout). Finally, we transform the solution to the integer program back into the corresponding multiplot.

Greedy Visualization Planner. The first planner guarantees optimal solutions (unless a timeout is reached). For complex problem instances, planning overheads can become prohibitive. This is why MUVE offers an additional greedy planner with polynomial complexity in the input problem dimensions. As shown in more detail in our experiments, this planner tends to be very fast (order of milliseconds) while producing near-optimal solutions. MUVE allows users to switch between those planners on a per-query basis to explore the differences.

[3]https://www.gnu.org/software/glpk/

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.

3 EXPERIMENTAL EVALUATION

We present an extract from our experiments, illustrating differences between the two visualization planners (Section 3.1), and comparing MUVE to a baseline in a user study (Section 3.2).

We use three datasets, one covering contacts for advertising, provided by a partner in industry, data about the department of buildings in NYC (DOB)[4], and data on NYC's 311 service requests[5].

3.1 Comparison of Visualization Planners

We compare the two planners in terms of optimization time and quality of output. This experiment was conducted on a MacBook Pro (15-inch, 2018) with a 2.9 GHz 6-Core Intel Core i9 processor and 32 GB of DDR4 memory. MUVE is implemented in Java, we use GLPK 4.65 to solve integer programs with a timeout of 10 seconds. We randomly generated count queries on DOB by selecting columns and constants for unary equality predicates with uniform distribution. We used the process described before to obtain a set of phonetically similar query candidates. We measure planning time and result utility (i.e., sum of probabilities of all query interpretations whose results are shown), scaled to the maximum value. Each data point corresponds to the arithmetic average of five runs.

Figure 3 shows time and utility as a function of the number of candidate queries (with two rows of plots). Clearly, integer programming generates slightly better solutions but suffers from high overheads for some range of candidate counts. Figure 4 varies the number of rows in the multiplot for 100 candidate queries. Integer programming works well up to two rows, the greedy algorithm is preferable after that. Note that the greedy algorithm even finds better solutions than the ILP solver starting from four rows. This is due to the high ratio of timeouts, preventing the ILP solver from finding optimal solutions.

3.2 User Study

We conducted a user study over Zoom with 10 participants, nine of them college students with four CS students, comparing MUVE against a baseline. 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 [3]. 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 discard the first 10 queries per participant as warmup and report arithmetic averages in Figure 5. Clearly, visually identifying the desired result in a multiplot is faster than resolving ambiguities by clicking buttons.

[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

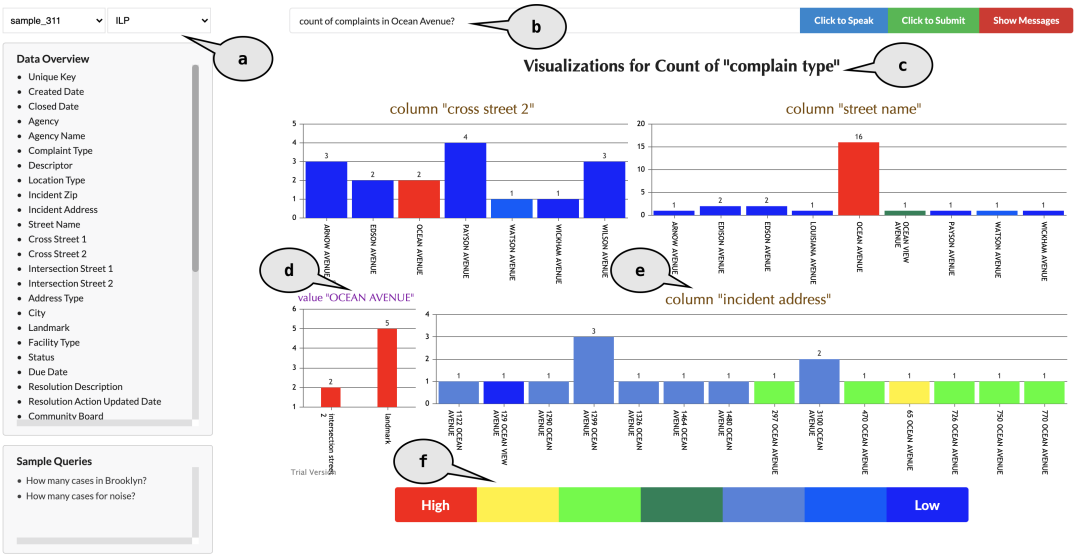

Figure 2: MUVE allows users to select dataset and visualization planner (a) and to speak (or type) natural language queries (b). The resulting multiplot contains results for similar queries whose common elements are outlined in the headline (c), while covering specific templates in specific plots (d, e). The bar color is the probability of the associated query interpretation (f).

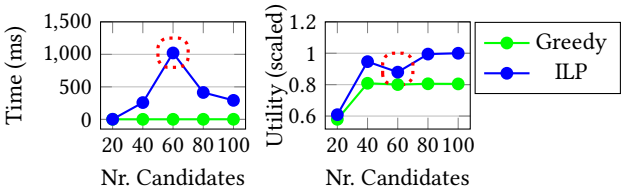

Figure 3: Time and utility for different visualization planners, dependent on the number of query candidates. Dashed red lines mark timeouts.

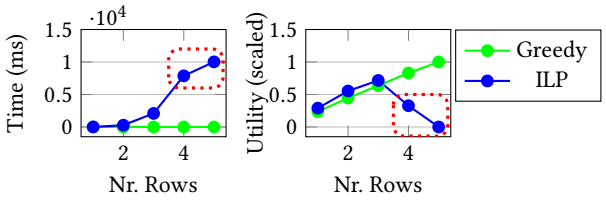

Figure 4: Time and utility for different visualization planners, dependent on the number of multiplot rows. Dashed red lines mark timeouts.

4 DEMONSTRATION

We will make our demo available on a publicly accessible Web site (the interface is Web browser-based). Figure 2 shows the demo interface and explains its primary components.

Participants can select among the three datasets used in the experiments. Dataset columns and example queries are shown on the left side of the interface. Users activate voice recognition by clicking a corresponding button, then formulate their query (alternatively, users can enter queries via keyboard). The voice query is transcribed and a multiplot is shown, containing results for different query interpretations. Also, participants can select between the two visualization planners and click on the red button to obtain additional information (e.g., planning time).

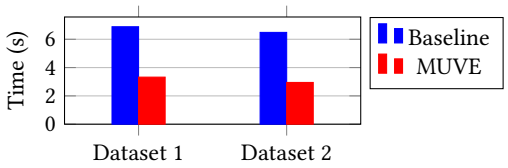

Figure 5: Average ambiguity resolution time for users with MUVE, compared to baseline.

REFERENCES

[1] Dharmil Chandarana, Vraj Shah, Arun Kumar, and Lawrence Saul. 2017. SpeakQL: towards speech-driven multi-modal querying. In *HILDA*. 1–6.
[2] Matteo Francia, Enrico Gallinucci, and Matteo Golfarelli. 2020. Towards conversational OLAP. *CEUR Workshop Proceedings* 2572 (2020), 6–15.
[3] 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.
[4] 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
[5] 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).
[6] 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.

[7] 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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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

[13] 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

[14] 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

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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