A Holistic Approach for Query Evaluation and Result Vocalization in Voice-Based OLAP

Immanuel Trummer, Yicheng Wang, Saketh Mahankali
Cornell University, Ithaca (NY)
{itrummer,yw876,sm2684}@cornell.edu

ABSTRACT

We focus on the problem of answering OLAP queries via voice output. We present a holistic approach that combines query processing and result vocalization.

We use the following key ideas to minimize processing overheads and maximize answer quality. First, our approach samples from the database to evaluate alternative speech fragments. OLAP queries are not fully evaluated. Instead, sampling focuses on result aspects that are relevant for voice output. To guide sampling, we rely on methods from the area of Monte-Carlo Tree Search. Second, we use pipelining to interleave query processing and voice output. The system starts providing the user with high-level insights while generating more fine-grained results in the background. Third, we optimize speech output to maximize the user's information gain under speaking time constraints. We use a maximum-entropy model to predict the user's belief about OLAP results, after listening to voice output. Based on that model, we select the most informative speech fragments (i.e., the ones minimizing the distance between user belief and actual data).

We analyze formal properties of the proposed speech structure and analyze complexity of our algorithm. Also, we compare alternative vocalization approaches in an extensive user study.

ACM Reference Format:
Immanuel Trummer, Yicheng Wang, Saketh Mahankali. 2019. A Holistic Approach for Query Evaluation and Result Vocalization in Voice-Based OLAP. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3300089

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3300089

1 INTRODUCTION

How to analyze data if you are blind? Almost all prior work on OLAP focuses on visual interfaces. Our goal is to answer OLAP-style queries via voice output instead.

Example 1.1. The system presented in this paper allows for instance the following interaction. Assume we obtain the voice input *"How does the flight cancellation probability in New York depend on flight date and start airport?"* from the user. As our focus is on voice output generation, our system uses a simple, keyword-based method to translate voice input into an OLAP query such as SELECT avg(cp) FROM table WHERE airportState='New York' GROUP BY flightSeason, airportCity. Next, it uses data sampling, user modeling, and optimal voice output planning, to select a concise description of the query result. It might generate the output *"Considering flights from airports in New York and on any date. Results are broken down by flight season and airport city. One percent is the average cancellation probability. Values increase by 20% for flights in Winter. Values increase by 5% for flights starting from New York City."*.

Our problem setting is not only motivated by the needs of specific user groups. In general, the communication between user and computer is more and more shifting towards speech interfaces. Devices such as Google Home or Amazon Echo are designed primarily for voice interaction. Voice interfaces are more convenient if users are distracted or unable to access screen and keyboard (e.g., imagine the example of a surgeon who needs to access patient data while performing a procedure [17]). Those factors have recently led to several publications on voice-based query interfaces [2, 17] and efficient result output ("vocalization" [24, 25]).

To the best of our knowledge, we present the first system for voice-based OLAP. Our research focus lies on the generation of concise voice answers (our system also features a simple voice input interface). Voice output needs to be extremely concise and simple [24, 25]. In contrast to written text, users cannot easily re-read difficult passages or skim text to quickly discard irrelevant parts. Hence, we treat vocalization as an optimization problem. Our goal is to transmit the maximal amount of information under constraints on speaking time and speech complexity. In contrast

to prior work, we target results of OLAP queries in particular [24, 25]. We will demonstrate in our experiments that prior vocalization methods are unsuitable in this scenario.

In Section 3, we develop a specialized grammar for describing OLAP results. We show that our grammar has several properties that are desirable from the user perspective (i.e., informative and simple speeches) and system perspective (i.e., we can efficiently generate corresponding speeches). Our ultimate goal is to teach listeners about OLAP results. To judge the quality of a candidate speech, we must reason about how it influences the user's belief about data. This is challenging as voice output must be inherently imprecise due to conciseness constraint. We perform a pilot user study to learn how users fill in missing information. Based on study results, we develop a probabilistic model capturing user expectations after listening to a given speech. Our goal is to bring the user's expectations as close as possible to the true query result.

In Section 4, we show how to generate near-optimal voice answers under real time constraints. Our approach combines query evaluation and result vocalization. We demonstrate in our experiments that such a holistic approach has tremendous performance benefits. Evaluating OLAP queries on large data sets can be expensive. Also, our search space for voice output is quite large and evaluating the quality of a given speech (by applying our user model) is expensive. We exploit the following main ideas to generate voice answers with almost imperceptible delays (i.e., well below the threshold of 500 milliseconds that makes interactive data analysis convenient [16, 19, 23]).

Sampling. Due to conciseness constraints, we consider a space of relatively coarse-grained voice descriptions. Transmitting precise OLAP results is typically impossible under those constraints. Hence, instead of evaluating queries entirely, we evaluate them on small data samples. We use those samples in order to estimate the quality of speech fragments. Also, we do not fully instantiate our user model for each evaluated speech. Instead, we only instantiate our model for randomly selected aspects of an OLAP result.

Prioritization. Our algorithm is iterative and chooses in each iteration a speech whose quality estimate are refined via sampling. Our search space is large. We cannot obtain precise quality estimates for all speech candidate. We use an approach based on Monte-Carlo Tree Search (MCTS) [14] to prioritize samples. This approach balances near-optimally between exploration (i.e., refining estimates for speeches about which little is known) and exploitation (i.e., refining estimates for promising speeches).

Pipelining. Voice output is transmitted sequentially (as opposed to visual output). We exploit that property and interleave query processing and voice output planning with readout. While the current sentence is spoken, we determine the best follow-up in the background. This gives us the freedom to start voice output with high-level insights that can be quickly generated. While voice output is playing, we can generate more specific follow-ups in the background.

We evaluate our algorithm in comparison to baselines. We measure raw processing time as well as user preferences in multiple user studies. In those studies, we asked crowd workers to analyze several data sets via an online interface for voice-based OLAP. We summarize our original scientific contributions:

- We introduce the problem of OLAP vocalization. We present a speech grammar and an associated user behavior model.
- We describe an approach for interleaved query evaluation and result vocalization. This approach uses sampling and pipelining and is tailored to the particularities of our scenario.
- We evaluate this approach experimentally in several user studies. We demonstrate favorable performance compared to baselines in terms of processing times and user preferences.

The remainder of this paper is organized as follows. We formalize our problem model in Section 2. We introduce our speech model in Section 3 and our query evaluation and vocalization approach in Section 4. Next, we present our experimental results in Section 5 and compare against related work in Section 6. The appendix contains additional experimental and theoretical results.

2 PROBLEM SCENARIO

We evaluate OLAP-style aggregation queries via sampling and describe their result via voice output. A *query* q is characterized by an *aggregation function*, $q.fct$, an *aggregation column*, $q.col$, and a set of *aggregates*, $q.aggs$. Motivated by our query evaluation approach based on sampling, we support three common aggregation functions: average, sum, and count. Functions such as minima and maxima are notoriously difficult to approximate via sampling [20]. We focus on queries with a single aggregation function, applied to a single column. Our approach could be easily extended to support multiple functions and columns. However, allowing more complex queries seems irrelevant as voice output needs to be kept simple.

An aggregate in the query result is characterized by a filter condition on the input data. Different aggregates for the same query are associated with mutually exclusive conditions (i.e., each row in the data set is relevant for at most one aggregate per query). We also say a tuple is *within scope* for a specific aggregate if it satisfies the associated condition. The *query scope* consists of rows that are within the scope of any of its aggregates. Via query evaluation, an aggregate is associated

with an aggregate value (which results from applying the aggregation function to all rows in its scope). An exact OLAP *query result* maps each aggregate to its accurate value.

Our model is slightly unusual in that we model query aggregates explicitly. Typically, aggregates are modeled as Cartesian product between admissible values in different dimensions. We choose this representation as it simplifies our pseudo-code in certain cases. Our actual implementation represents query aggregates however more efficiently.

Our definition leaves open the source of rows that are considered for aggregation. We only assume that rows can be produced without significant startup overheads and at a sufficiently high frequency. This is for instance the case if scanning a single source table or when joining fact table entries with indexed dimension tables.

We assume that aggregates are associated with conditions of the following structure. A condition corresponds to a conjunction of several atomic conditions. Each atomic conditions restricts values in one column to a subset of its value domain. We call the columns on which restrictions can be placed *dimensions*. We assume that their value domain is structured into a *(dimension) hierarchy*. Each hierarchy has one or multiple *levels* that allow to place restrictions at different granularities.

Example 2.1. Assume a relational table storing employees with their mid-career (column M) and start salary (S) as well as their college (C) containing information on employees. We consider start salary and college as dimension columns. We structure their value domain hierarchically, e.g. by grouping colleges first by city, then by state, and finally by US region. For the start salary, we can group values by rough salary category (e.g., 60 K or 90 K) or even by larger salary ranges (e.g., less than 75 K versus more than 75 K). Query q with $q.fct = AVG$, $q.col = M$, $q.aggs = \{C = Northeast \wedge S = 60K, C = Northeast \wedge S = 80K, C = Midwest \wedge S = 60K, \ldots\}$ retrieves mid-career salary averages for the Northeast and Midwest regions for salaries from 60 to 80 K, grouped by region and 10 K salary granularity.

We address the (optimization) problem of *vocalizing OLAP query results*. A problem instance is characterized by a query on a specific database. An optimal problem solution is a speech, describing the query result as good as possible under all applicable constraints. Following prior work [24], we consider (threshold) constraints on speech length (measured as the number of characters) and number of speech fragments (i.e., sentences). We will define the search space for OLAP vocalization in Section 3 (as part of our original scientific contribution). Based on user studies, we also associate each speech with a probability distribution modeling users' beliefs about the query result (after hearing the speech). Intuitively, our optimization goal is to find a speech that brings the user's belief as close as possible to the actual query result. More precisely, we judge speech quality as defined next.

Definition 2.2. Denote by $\mathcal{B}(a, t)$ the user's belief about an aggregate a in the result to query q after listening to speech text t, i.e. $\mathcal{B}(a, t)$ is a probability distribution that assigns a probability to values for a. The **Speech Quality** q of t is defined as $\sum_{a \in q.agg} \Pr(a(D)|\mathcal{B}(a, t))/|q.agg|$ where $a(D)$ is the actual query result (or, for continuous distributions, a value range including the actual value).

Intuitively, speech quality is the average probability that users assign to actual query result values, after listing to the speech. We will show later that this metric correlates with the performance of users in estimating query result values after listening to speech output. Finally, we formally introduce the problem that we are targeting.

Definition 2.3. The problem of **Optimal OLAP Vocalization** is characterized by a query (whose result is unknown) and a search space T of speech candidates (subject to constraints on speech length). Denote by $quality(t)$ the quality metric introduced before. The goal in optimal OLAP vocalization is to find a speech $t \in T$ such that $\forall t \in T : quality(t) \geq quality(\widetilde{t})$.

Note that our problem definition does not assume that a query result is initially available. This motivates the interleaved approach for query evaluation and vocalization presented in Section 4.

3 SPEECH DESIGN

Our goal is to define a grammar for voice descriptions of OLAP query results and an associated semantics. In Section 3.1, we discuss several desirable properties for such a grammar. Those properties integrate the user perspective (i.e., what makes a speech understandable to listeners) as well as the system perspective (i.e., what properties make speech generation easier). In Section 3.2, we present a grammar that has those properties. In order to compare vocalization candidates, we need to model how speech fragments influence the listener's belief about data. This is challenging as concise voice output leaves many details open, requiring us to reason about how users fill in information gaps. We present a user study on this issue in Section 3.3. Based on the results, we associate our speech grammar with semantics in terms of user beliefs (Section 3.4).

3.1 Design Considerations

In the following, we define, loosely, several properties of speech grammars. We also justify why those properties are desirable for vocalization.

Conciseness. Voice output needs to be very concise due to short-term memory constraints of the listeners [24, 25].

Table 1: Symbols used in speech grammar.

Symbol	Semantics
<Pr>	Preamble: summarizes input query
<P>	Predicate: characterizes data subset
<L>	Name of query dimension level
<B>	Baseline: introduces default result value
<V>	Value for aggregation function
<A>	Name of query aggregate
<R>	Refinement: describes a data subset
<C>	Change: how values differ from default
<Q>	Quantifier of change
<Dc>	Dimension context: embeds member name
<M>	Member in dimension hierarchy

OLAP query results may contain too many aggregates to describe each one via voice output (we show that this case is common in Section 5). Hence, a speech grammar for OLAP must include means of abstraction that summarize many aggregates.

Precision. When vocalizing small results (or being configured with particularly relaxed speaking time constraints), a system for voice-based OLAP should be able to benefit. In such cases, the grammar should allow us to generate precise descriptions. While means of abstraction are important for conciseness, we need speech fragments that can transmit fine-grained result details as well.

Extensibility. Voice output, as opposed to visual output, is generated sequentially. This creates, as we show later, opportunities to overlap processing time and speech output. To leverage those opportunities, we need a speech structure that allows refinements. More precisely, the speech structure should enable us to refine prior coarse-grained claims without creating contradictions.

Simplicity. The speech grammar spans the search space for result vocalization. The overhead of planning methods often grows in the search space size. Vocalization is subject to tight time constraints, motivated by the requirements of interactive data analysis. Hence, it is desirable to keep the search space (and thereby search overheads) as small as possible under the aforementioned constraints.

3.2 Speech Syntax

We propose a speech grammar and justify why it satisfies, to a large extent, the properties introduced before. Figure 1 shows the syntax of the proposed grammar in EBNF form, Table 1 describes the grammar symbols. We start each speech

```
<Speech>::=<Pr><B><R>*
<Pr>::=Considering (<P>|(<P>)+ and <P>).
 [Results are broken down by (<L>|<L>+ and <L>).]
<B>::=<V> is the <A>.
<R>::=Values <C> for (<P>|(<P>)+ and <P>).
<C>::=(increase|decrease) by <Q>
<P>::=<Dc> <M>
```

Figure 1: Speech syntax represented in EBNF.

with a preamble, providing context for the following result. We observed that this feature helps users to keep track of their position in the dimension hierarchies. Next, we set a baseline that all the following statements relate to. More precisely, we provide an aggregate value that is typical for the result to vocalize. The baseline is followed by arbitrarily many refinement statements. While the baseline is maximally coarse-grained (as it applies to the query result as a whole), refinements only refer to subsets. Each refinement is characterized by predicates and a change descriptor. Predicates define the scope of the refinement by fixing attributes to specific values. The change descriptor describes a relative change in expected aggregate value. This change is relative either to the baseline value or to the last refinement whose scope subsumes the current one. Next, we show an example speech, derived using the grammar in Figure 1.

Example 3.1. The following speech answers a query for average mid-career salary in the entire data set, broken down by graduation region and rough start salary (e..g, SELECT avg(midCs) FROM T GROUP BY colReg, roughStS). We mark up speech elements via brackets with the corresponding grammar symbol as subscript: *"{Considering {{graduates from}$_{Dc}$ {any college}$_M$}$_P$ and {{a start salary of}$_{Dc}$ {any amount}$_M$}$_P$. Results are broken down by {region}$_L$ and {rough start salary}$_L$.}$_{Pr}$ {{90 K}$_V$ is the {average mid-career salary}$_A$.}$_B$ {Values {increase by {5 percent}$_Q$}$_C$ for {{graduates from}$_{Dc}$ {the North East}$_M$}$_P$.}$_R$ {Values {increase by {20 percent}$_Q$}$_C$ for {{a start salary of}$_{Dc}$ {at least 50 K}$_M$}$_P$.}$_R$"* This speech could have been generated by the system we evaluate in Section 5 (assuming that the system has been configured with appropriate dimension names, level names, and member names). The speech refers for instance to members (e.g., "the North East" or the dimension tree root "any college") and dimension tree level names ("region") in the college location dimension, using a dimension-specific context template ("graduates from") to embed member values. Note that our example query assumes a denormalized database while our system can handle queries on star schemata as well.

This speech grammar allows to choose the level of abstraction flexibly (by choosing the number of predicates in

refinement statements). It therefore supports concise high-level descriptions as well as fine-grained descriptions down to the level of single aggregates. Also, we can extend a speech as more fine-grained information becomes available: we can make coarse-grained statements about larger data subsets without knowing which fine-grained statements follow. Note that this property also derives from the fact that we use *relative* instead of *absolute* refinements. The following example illustrates the difference.

Example 3.2. We use the same query as in Example 3.1. Assume the sentence *"90 K is the average mid-career salary"* is followed by an absolute refinement of the form *"95 K is the average for graduates from the North East"*. This prevents us from making further statements about data subsets including graduates from the North East, without creating contradictions. The relative refinements used in the previous example allow us to add claims about start salary as new information becomes available (even though entries for specific start salary ranges may include entries for the North East region).

Apart from the previously discussed properties, the grammar has few elements. This is desirable in order to make planning more efficient.

3.3 User Model

To choose the best speech, we need to predict how users react to hearing a given speech candidate. More precisely, we need to reason about the beliefs that a speech induces about the query result. This becomes challenging due to the constraints of voice output. Voice output needs to be very concise. Hence, voice output is typically coarse-grained and leaves many details open. We must predict how users will intuitively fill in the gaps.

Example 3.3. Consider the speech *"Around 1% is the average flight cancellation probability in summer. Values increase by 1% in June."* What is the cancellation probability in July and August? As shown in the following experiments, most users intuitively assume a probability of slightly below one percent for both months (thereby assuming the most uniform distribution consistent with the description).

Our goal is to map a speech to a probability distribution over query results. This distribution represents the beliefs of an average user about the query result after listening to the speech. To map speeches to beliefs, we must make assumptions about the behavior of average users. We performed a user study to verify several hypotheses on user behavior. Those hypotheses derive from assumptions made in prior work on OLAP (e.g., the maximum entropy principle (MEP) [6, 18, 22]) or from common sense. More precisely, we verify the following hypotheses on user behavior:

Table 2: Summarized results of pilot user study on implicit assumptions.

Model Aspect	#Consistent Answers	#Inconsistent Answers
Symmetry	15	5
Concentration	28	12
Composition	21	19
Uniformity	15	5
Normal($\sigma \leq \mu$)	74	6

Symmetry. Given an average, users assume that deviations into both directions (i.e, larger and smaller values with a given distance to the average) are equally likely.

Concentration (unimodality). Given an average value, users assume that smaller deviations are more likely than larger deviations.

Composition. Given two refinements with overlapping scopes, users will be able to integrate their accumulated effect into their mental model.

Uniformity (MEP). Users tend to assume the most uniform result (where uniformity can be measured via the entropy) that is consistent with speech output.

In addition, we show that the value distribution that users intuitively assume can be approximated via a normal distribution. We try normal distributions with different standard deviation values for a given mean.

We recruited 20 crowd workers on the AMT platform for a pilot user study. We formulated a series of questions that test each of the aforementioned hypotheses. Table 2 summarizes the results. We report for each hypothesis the number of crowd answers supporting the hypothesis as well as the number of deviating answers. The precise questions and more details can be found in the appendix in Table 10. Overall, the majority of answers is in support of our hypotheses in each case (note further that only one out of three answer options is consistent for all aspects except normal distribution variance). We therefore base our speech semantics, modeling user beliefs, on those hypotheses.

3.4 Speech Semantics

We formally define the semantics of a speech from the user perspective. We model that semantics as a probability distribution over possible query results. It represents the user's expectations about the query result.

Voice output has to be concise. This makes it often necessary to use generalizing claims that apply to groups of aggregates. Our results from the last subsection demonstrate that

users tend to translate such claims into probability distributions. More precisely, in the absence of further information, users tend to imagine a distribution that is symmetric and unimodal. Further, they tend to imagine a distribution that is quite close to a normal distribution with a standard deviation proportional to the mean. We do not claim that this distribution is the only one consistent with our observations. As the normal distribution is well studied and efficiently computable, we use it as base for our user model. More formally, we define the semantics $\mathcal{B}$ of a speech text t as a function that maps query aggregates a to normal distributions with scenario-specific variance σ[1]:

$$\mathcal{B}(a,t) = \mathcal{N}(\mathcal{M}(a,t),\sigma)$$

We designate by $\mathcal{M}$ the mean of the normal distribution that is assigned by speech to an aggregate. We define $\mathcal{M}(a,t)$ recursively as follows. Assume first that t is of type `<Baseline>` and let v be its aggregate value. Then we set $\mathcal{M}(a,t) = v$ for all aggregates a. Here, we exploit our findings from before that users tend to assume more uniform value distributions. This is a common assumption in the area of OLAP in general [22].

Now, assume that t is the concatenation of a prefix x and a single refinement r (i.e., $t = x \circ r$). Further assume that the refinement r describes an additive change of Δ. Refinements refer to a subset of aggregates that are defined via predicates. Clearly, we set $\mathcal{M}(a,t) = \mathcal{M}(a,x) + \Delta$ if a falls within the scope of r. Here, we exploit our findings on how users compose effects of multiple refinements.

Refinements should also change user's belief with regards to aggregates that do *not* fall into their scope. This is due to the baseline in particular, the only absolute statement in a speech. The baseline claim sets the average over all aggregates. If a refinement increases expected values for some of the aggregates, it must at the same time lower expected values for the others to remain consistent with the baseline claim. Hence, if refinement r applies to m out of n result aggregates, we must set $\mathcal{M}(a,t) = \mathcal{M}(a,t) - m \cdot \Delta/(n-m)$ for each aggregate a outside of r's scope.

Example 3.4. Consider the following speech with text t: *"The average salary is 80 K. Values increase by 50% for graduates from the Northeast."* Assume that query results are broken down by region (Northeast, Midwest, West, and South). Hence, we have four aggregates in the result. It is $\mathcal{B}(Northeast,t) = \mathcal{N}(120,000,\sigma)$ (since adding 50% to 80,000 yields 120,000). We choose $\sigma = 40,000$ (half the average of the entire data set). For all other aggregates, e.g. the Midwest, we set $\mathcal{B}(Midwest,t) = \mathcal{N}(66,667,\sigma)$. This assures that averaging over all four aggregates yields a result consistent with the first speech sentence.

Note that we can generate user beliefs easily for specific aggregates (as opposed to having to calculate belief distributions for each aggregate). This property is important for the design of our algorithm, presented in the next section.

We analyze the formal properties of our speech semantics in more detail in Appendix A.

4 QUERY EVALUATION AND VOCALIZATION

We describe how to evaluate queries and to vocalize query results in a unified approach. Our goal is to support interactive analysis, implying tight time constraints (less than 500 ms before voice output starts [16, 19, 23]). This is challenging due to the following factors. First, processing OLAP queries on large data sets may easily exceed those time limits. Second, despite a relatively simple speech structure, we consider a search space of elevated size. Also, evaluating the quality of a single speech can be expensive, too. To fully evaluate speech quality, we must compare the belief it implies on each aggregate to the real value. The number of aggregates in the query result may however be large (it grows exponentially in the number of dimensions the query refers to).

Our algorithm is based on the three main ideas summarized in Section 1. In Section 4.1, we describe a high-level algorithm implementing those ideas. The next subsections describe sub-functions used by that algorithm in more detail and some extensions. We also analyze the complexity of this algorithm in Appendix A.

4.1 Main Function

Algorithm 1 is the main procedure of our algorithm for combined query evaluation and result vocalization. It obtains as input a query and a description of user preferences. User preferences restrict speech length (measured in characters) as well as the number of speech fragments (i.e., the number of refinements). The procedure generates voice output, sentence by sentence, while processing data in the background.

The algorithm uses several auxiliary functions. We group those functions into three categories: speech generation, voice output, and search tree operations. By convention, we use the first two letters of each function name to hint at the associated group. Table 3 summarizes the auxiliary functions discussed in this and the following sub-sections.

[1]We model sigma as a constant that is approximately proportional to 50% of the mean when aggregating over the entire data set. While one could adapt σ dynamically to the mean in the current query, we believe that our approach represents a good tradeoff between model complexity and accuracy.

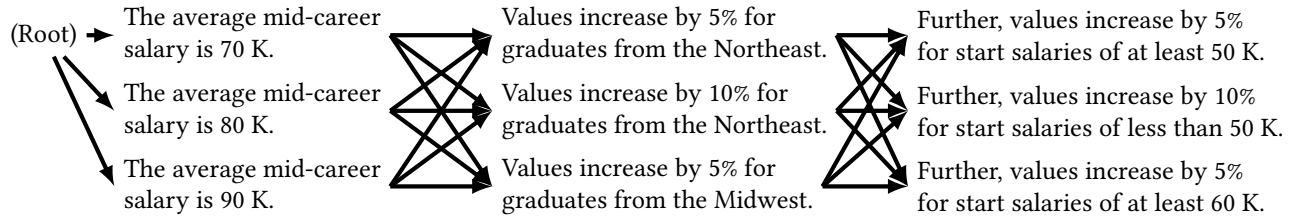

Figure 2: Example for (part of) speech search tree.

```
1:  // Partially evaluate query q via sampling to generate
2:  // near-optimal speech describing query result while
3:  // respecting user preference p.
4:  procedure EvalVocal(q, p)
5:      // Start voice output of preamble
6:      t ←SG.preamble(q)
7:      VO.start(t)
8:      // Initialize search tree for speech output
9:      root ←ST.newNode(−, t)
10:     ST.expand(q, p)
11:     // Iterate until result speech finished
12:     finished ← false
13:     while ¬finished do
14:         // Iterate until current voice output finishes
15:         while VO.isPlaying do
16:             ST.sample(q, root)
17:         end while
18:         // Is the speech finished?
19:         if root.isLeaf then
20:             finished ← true
21:         else
22:             // Choose next sentence
23:             root ← arg max_{c∈root.children}
                    c.reward/c.visits
24:             // Start playing next sentence
25:             VO.start(root.lastSentence)
26:         end if
27:     end while
28: end procedure
```

Algorithm 1: Main algorithm for combined query evaluation and result vocalization.

Table 3: Overview of auxiliary functions for voice output, speech generation, and speech tree operations.

Function	Description
VO.start(s)	Starts speaking text s (returns immediately)
VO.isPlaying	Returns true iff voice output is still playing
SG.preamble(q)	Generates speech preamble for query q
SG.refinements(q, t)	Generates next sentence candidates refining description t for result to query q
SG.isValid(t, p)	Returns true iff speech text t is valid according to user preferences p
ST.newNode(t, l)	Initializes new search tree node representing speech with prefix t and last sentence l
ST.expand(q, p, n)	Expands search node n for speech candidates for query q respecting user preferences p
ST.sample(q, n)	Refines speech quality estimates by sampling speech tree from node n
ST.maxUctChild(n)	Returns most promising child of node n according to UCT formula
ST.addChild(n, c)	Adds c as new child for node n in speech tree

Algorithm 1 initially generates the preamble, summarizing the input query. This is helpful for users to orient themselves when exploring a new data set with multi-level hierarchies. We start voice output for the preamble using function VO.start. This function is asynchronous and returns immediately while voice output starts playing. While reading out the preamble, we create a search tree representing possible continuations. For that, we use function ST.newNode (to generate the search tree root) and function ST.expand (to expand the tree from the root). The implementation of both functions is discussed in more detail in Section 4.2.

Next, Algorithm 1 starts iterating. We iterate until the current speech cannot be refined further (without violating user preference constraints). In that case, the flag $finished$ is set to **true**. In the first part of each iteration, we refine our speech quality estimates while voice output is playing. We use function VO.isPlaying to determine whether the previous sentence is still being spoken out. We overlap voice

output with invocations to the function ST.SAMPLE. The implementation of this function is discussed in the next subsection. On a high level of abstraction, the function refines quality estimates about speech candidates via sampling. In each invocation, it selects an interesting candidate speech and a data sample, and verifies how well the speech describes the data sample.

Our stopping condition is based on the duration of voice output for the current sentence. After voice output for the previous sentence stops playing, we decide whether further output is possible. If so, we select the most promising child node of the current node in the search tree. Nodes in the search tree represent alternative speeches. They are associated with accumulated reward values (capturing how well the speech seemed to describe data samples) and a number of visits, i.e. the number of times the corresponding speech was evaluated. The most promising child maximizes the obtained average reward. We start voice output for the newly added sentence and continue iterations.

Note that planning does not start from scratch for each new sentence. Instead, we simply make the root of a prior sub-tree the new tree root. This means that all statistics collected previously within that sub-tree remain available. Thereby we avoid redundant planning work.

Example 4.1. Figure 2 shows an extract of an example search tree. Each node is associated with speech fragments. Each path corresponds to a speech (that concatenates the sentences associated with all traversed nodes). Each tree level in the figure is associated with a different aspect of the data set. The height of the tree (here: the horizontal extent) is limited by user preferences, limiting speech length and the number of fragments by a threshold. Here: the first tree level after the root sets a baseline while the following levels correspond to refinements.

Our algorithm associates each node in the figure with quality estimates (not shown in the figure). It refines those estimates continuously by comparing speech claims against samples drawn from the database. Sampling starts at the left-most node in Figure 2 until the preamble (not shown) is read out completely. Assume that node *"The average mid-career salary is 80 K."* has highest quality estimates at that point. We use that node as new root for sampling while the latter sentence is read.

4.2 Search Tree Operations

We describe the operations on the search tree, modeling speech candidates, in more detail. Table 4 summarizes the fields that are associated with each search tree node. Those fields can be decomposed into fields describing the associated speech, fields capturing the tree structure, and statistics used by the planner. Fields of the first two categories are rather self-explaining. We discuss planning and the associated statistics next.

Table 4: Overview of search node fields.

Field	Semantics
text	Text of speech represented by this node
lastSentence	Last sentence of speech represented by this node
children	Set of child nodes of this node
isLeaf	Flag indicating a leaf node (i.e., $children = \emptyset$)
visits	Number of times the node was visited during sampling
reward	Sum of rewards accumulated during sampling over all paths traversing this node

The goal of planning is to identify speech fragments that best describe the query result. Of course, we do not know the full query result but only result samples. This means that we cannot obtain accurate values for the quality of a speech candidate. Instead, we can only obtains samples from a quality distribution whose mean is centered on true speech quality (we discuss quality samples in more detail in Section 4.3). To cope with this situation, we need a search algorithm that can deal with uncertain quality values.

MCTS [1] is among the most popular algorithmic frameworks that satisfy this requirement. Methods from this area evaluate solutions, derived from a search tree, via sampling. In our case, the search tree represents alternative vocalizations. Sampling refers to testing speech accuracy on randomly selected query result samples.

More precisely, we use the UCT algorithm [14], a specific representative from the MCTS family. The UCT algorithm is characterized by its strategy for prioritizing nodes during sampling. Each search tree node is associated with two counters: the accumulated reward and the number of times the node was visited. Those counters can be used to determine confidence bounds on the average reward realized by specific nodes. The UCT algorithm calculates the upper bound of the confidence intervals and prioritizes accordingly. This strategy is optimistic in that the algorithm assumes for each node the highest possible reward that is consistent with prior observations. This strategy also balances the two aforementioned criteria, exploration (i.e., gathering information on rarely visited nodes) and exploitation (i.e., refining estimates for nodes that currently seem promising) as follows. Exploration is motivated by the fact that confidence bounds shrink once more samples are known. This favors less frequently

visited nodes over frequently visited nodes even if the observed mean reward is the same. Exploitation is motivated by the fact that higher reward averages imply higher confidence bounds. We choose the UCT algorithm as it offers formal guarantees on striking an optimal balance between those two criteria [14].

Algorithm 2 shows how the UCT algorithm is applied in our scenario. The term UCT describes a family of algorithms rather than a single algorithm [1]. Among the more unusual design decisions we make for our variant is the pre-processing step, in which we generate the search tree in its entirety. This is not practical in typical application scenarios for the UCT algorithm [4]. In our scenario, the height of the search tree, and therefore the search space size, is however restricted by user preference constraints. For that reason, we find it more practical to resolve all overheads related to tree generation in a pre-processing step. Note finally that the formula by which we select child nodes in our main algorithm (Algorithm 1) differs from the one used in Algorithm 2. In Algorithm 2, collecting more information is useful as it allows us to make better choices in future iterations. This is why the formula contains a term capturing exploration reward (the second term). In contrast to that, Algorithm 1 cannot afford further exploration when selecting the best child node (otherwise, voice output would be interrupted). Hence, selection is based on exploration bonus alone.

Example 4.2. We consider the search tree from Figure 2 again. The sampling procedure starts at the root node (left), and picks at each level the most promising child node until a leaf is reached. Assume that this results in the speech *"The ... is 70 K. ... by 5% ... Northeast. ... by 5% ... at least 50 K."* We evaluate it by comparing against a data sample (described in more detail in the following section). Assume that a reward value of 0.7 is obtained. We update the statistics of all three nodes it refers to: we increment the counter representing number of visits by one and add 0.7 to the accumulated reward. The counters of other nodes do not change.

4.3 Speech Evaluation via Sampling

We evaluate a speech based on how well it matches a query result sample. There has been prior work on using sampling to speed up OLAP query processing [3, 8, 10, 15]. Our sampling method exploits however several particularities of voice-based (as opposed to visual) OLAP. First, while a visualization needs to present (at least approximate) values for each aggregate in the query result, a vocalization can only address a subset (due to conciseness constraints). Second, when vocalizing, we can freely choose the level of abstraction of our claims (by choosing the number of predicates in each refinement). Third, while a visualization appears at once, voice output is transmitted sequentially to the user.

```
// Adds child nodes to node n representing valid
// speeches for query q according to preferences p.
procedure ST.EXPAND(q, p, n)
    for r ∈SG.REFINEMENTS(q, n.text) do
        if ST.ISVALID(n.text ∘ r, p) then
            c ←SG.NEWNODE(n.text, r)
            n.children ← n.children ∪ {c}
            ST.EXPAND(q, p, c)
        end if
    end for
end procedure

// Returns child of n optimizing tradeoff
// between exploration and exploitation.
function ST.MAXUCTCHILD(n)
    // Prioritize unvisited children
    S ← {c ∈ n.children|c.visits = 0}
    if S = ∅ then
        // Maximize UCT formula
        S ← arg max_{c∈n.children}
            c.reward/c.visits + sqrt(2·log(n.visits) / c.visits)
    end if
    return Random pick from S
end function

// Sample speech tree for query q from node n,
// evaluate speech quality via sampling from the
// database, update tree statistics accordingly.
procedure SAMPLE(q, n)
    path ← {n}
    // Traverse tree until reaching leaf
    while ¬n.isLeaf do
        n ←ST.MAXUCTCHILD(n)
        path ← path ∪ {n}
    end while
    // Evaluate leaf speech via database
    r ←SPEECHDBEVAL(q, n.text)
    // Update statistics on tree path
    for n ∈ path do
        n.visits ← n.visits + 1
        n.reward ← n.reward + r
    end for
end procedure
```

Algorithm 2: Operations on speech search tree.

Altogether, we have more flexibility when vocalizing query results instead of visualizing them. We can exploit that flexibility to facilitate sampling. In the context of visual OLAP, the structure of the visualization is decided upon first and implies constraints on sampling. This makes sampling hard, e.g.

```
// Randomly pick an aggregate from the result
// of query q to sample speech quality.
function PickAggregate(q)
    // Determine eligible aggregates
    if q.fct ∈ {COUNT, SUM} then
        // All query aggregates
        E ← q.aggs
    else
        // Query aggregates with cache content
        E ← {a ∈ q.aggs|Ca.size(a)> 0}
    end if
    return Pick random element from E
end function

// Generate value estimate for aggregate q
// from cache content.
function CacheEstimate(a)
    // Get fixed-size sample from cache
    V ←Ca.resample(a)
    // Calculate count estimate
    e_C ← nrRows·Ca.size(a)/Ca.nrRead
    // Calculate sum estimate
    e_S ← e_C · Σ_{v∈V} v/|V|
    // Calculate average estimate
    e_A ← e_S/e_C
    return e_C, e_S, or e_A depending on q.fct
end function

// Evaluates a speech t based on how well it
// predicts database samples approximating the
// result for query q, returns reward value.
function SpeechDBeval(q, t)
    // Pick random aggregate with cached entries
    a ←PickAggregate(q)
    // Estimate aggregate value from cache
    e ←CacheEstimate(a)
    // Calculate user belief based on speech
    b ← B(a, t)
    // Compare cache estimate and belief
    return Pr(e|b)
end function
```

Algorithm 3: Evaluating a candidate speech via database samples.

if a visualization requires samples from rare sub-populations. In voice-based OLAP, we can collect samples first and choose output topics based on what samples are obtained later. Hence, in an inversion of the aforementioned dependencies, the collected samples motivate topics for voice output.

Algorithm 3 implements those ideas. Function SpeechDBeval (used in Algorithm 2) takes as input a query and a speech text. It evaluates how well the speech text describes the query result. As the query result is not fully available, it uses samples to get an approximation. The function returns a reward value between zero and one. The higher the reward, the higher the likelihood that the speech accurately describes the query result.

Each invocation of SpeechDBeval performs the following steps. First, we select a random aggregate from the query result. Next, we estimate a value for this aggregate from database samples. We retrieve those samples from a cache whose specifics are discussed later. Then, we generate a probabilistic model of user beliefs about data after listening to the speech. For that, we use the behavior model that was presented in Section 3.4. We do not generate a full model of the user's beliefs. Instead, it is sufficient to calculate user beliefs only for the aggregate selected previously. The user belief corresponds to a probability distribution. Hence, we return as reward the probability it assigns to the cache-based estimate. Hence, we reward speech text leading users to beliefs that match the data well.

We continuously collect samples from the database that are stored in a cache. Compared to approaches for OLAP visualization, we have much more time for sampling without violating run time constraints. As we overlap sampling with voice output, most output sentences can benefit from many seconds of sampling time without degrading user experience. The speech grammar presented in Section 3.2 places the preamble and the baseline speech fragments first. The preamble describes the query scope and does not require information from the database. Also, the baseline refers to the query result as a whole and does not require samples with regards to smaller sub-populations. We can choose to make refining statements about small sub-populations. Those refinement statements are then already based on samples collected over several seconds (often up to ten seconds). This sampling approach worked well for data sets we experimented on (see Section 5). It could be extended using prior work on sampling in the context of OLAP (e.g., specialized indexing structures [10]) to retrieve estimates for particularly small data subsets.

We consider the cache content when picking an aggregate for speech evaluation in Function PickAggregate. We can only evaluate a speech based on an aggregate if at least some information on its value is available in the cache. Function Ca.size(a) returns the number of entries stored in cache whose attributes match the constraints of aggregate a. The cache keeps track of counts during insertions in order to return counts in constant time. For an average aggregation function, we need at least one entry to derive estimates. For sum and count, the fact that no corresponding entry was retrieved yields information as well (by setting it in relation to the total number of samples retrieved from the database).

Among all eligible aggregates, we select one randomly with uniform random distribution. We use a uniform distribution as each result aggregate is equally important according to our quality metric (see Section 2).

Function CacheEstimate estimates a value for the given aggregate. We construct an estimate based on a fixed size subsample from the cached rows associated with one specific aggregate. This subsample is retrieved by a call to Function Ca.resample(a). The cache indexes entries with regards to the current query aggregates at cache insertion time to implement this function efficiently. In our current implementation, we use a fixed size of 10 samples. We do not use the full set of cached samples for the current aggregate. This would slow down sampling more and more as iterations progress (and as the cache is being filled). Alternatively, old cache entries can be discarded periodically to avoid this problem. We calculate unbiased estimators for the standard aggregation function as shown in the pseudo-code. Variable *nrRows* denotes the number of rows in the entire database. Function Ca.NrRead returns the total number of rows considered for caching. This is in general lower than the number of cached rows (since the cache inserts rows only if they fall within the current query scope). Finally, the function returns the appropriate estimate, given the input query aggregation function.

Example 4.3. We sketch out how quality estimates are obtained for the speech introduced (in abbreviated form) in the previous example. First, we select an aggregate to evaluate the speech on. The speech uses the average as aggregation function. Hence, we only consider aggregates for which at least one cache entry is available. Assume we select the aggregate "Graduates from the Midwest with an average salary of at least 50 K", respecting this condition. We use a subsample from the cache to calculate a rounded salary estimate of 90 K for that group. Next, we analyze the speech text and apply our user behavior model to obtain a probability distribution. Assume we obtain the normal distribution $\mathcal{N}(82K, 40K)$ as a representation of user belief after listening to our speech. The reward value is 0.1 since 10% is the probability to obtain this rounded value (i.e., a value between 85 K and below 95 K) according to user belief.

4.4 Extensions

Our algorithm can be extended to provide users with information on uncertainty. In order to transmit confidence bounds, we implemented two modes. Either, we generate the default speech and attach a general warning for the user if confidence in spoken values is lower than a threshold. Or, we speak out the precise confidence bounds at the point where voice rendering for the corresponding sentence starts. We can calculate confidence bounds based on the random samples we retrieve in the cache. The way in which confidence bounds are calculated is not specific to vocalization.

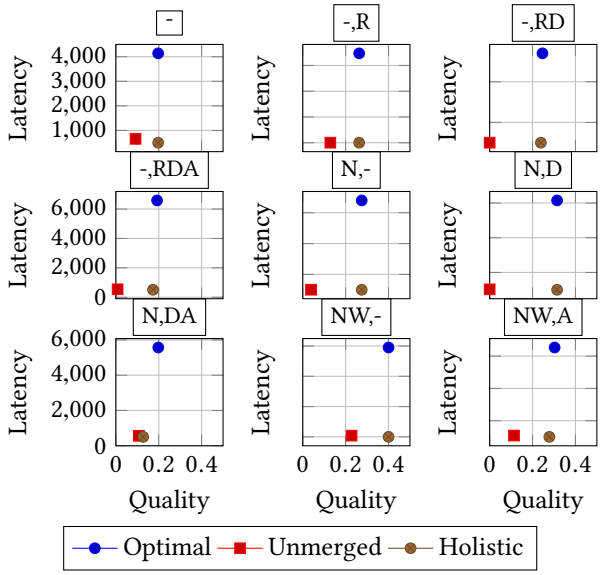

Figure 3: Performance of vocalization variants.

5 EXPERIMENTAL EVALUATION

We analyze the performance of our approach for query evaluation and result vocalization. We compare against the approach by Trummer et al. [25] for vocalizing relational tables (note that another recently proposed vocalization method [24] is only applicable to time series data). We use two data sets, a large data set containing information on flight cancellation probabilities with three dimensions, and a small data set containing information on mid-career salaries with two dimensions. All details of our experimental setup can be found in the appendix (Section B.2). Section 5.1 compares vocalization approaches in terms of processing time and speech quality. In Section 5.2, we report on the results of a user study in which crowd workers analyze data via different vocalization methods.

5.1 Processing Time and Speech Quality

We compare different variants of our approach in terms of latency (i.e., time after query submission until the speech starts) and in terms of speech quality. For the latter, we measure exact speech quality according to our model and based on the entire data set (i.e., average over all query result fields of the probability predicted for the actual value by the user belief distribution, created by that speech). We compare an "optimal" approach generating optimal speeches considering all data and calculating precise quality for each speech

before starting output, the "holistic" approach proposed in the previous section, and an "unmerged" approach which does not merge vocalization, sampling, and planning. Instead, it samples according to the same strategy as the "holistic" approach for 500 ms (threshold for interactivity) and then selects the speech with highest quality estimates.

Figure 3 reports results for different queries on a 600 MB data set about flight cancellations, calculating average flight cancellation probability. Queries are specified as pairs X,Y where X are members in predicates and Y dimensions for the breakdown. R, D, A are region, date, and airline respectively, N designates the North-East region and W is winter (e.g., N,DA is a query calculating cancellation probability for the North-East, broken down by date (at season granularity) and airline.

We make two observations. First, the latency for optimal vocalization is much higher than for other approaches and far above the "interactivity threshold" of 500ms (note that we process data from memory, disk access would likely worsen results further). This is explained by the fact that the optimal approach samples neither from the data nor in the plan space. Second, the quality of the unmerged approach is typically far below the other baselines. The unmerged approach cannot overlap sampling and planning time with vocalization. It therefore has less time to read data and explore the search space, compared to its competitors. Third, the quality of the holistic approach is almost identical to the optimal one. For the majority of queries, both approaches generate exactly the same speech.

So far, we compare approaches based on quality estimates, based on our user model. This model is based on a pilot user study but of course simplifying. Hence, we need to verify that speeches with higher quality estimates lead indeed to a better understanding of the data by users. We performed an additional user study on AMT, asking users to estimate query result fields based on different voice descriptions. We selected a query in which all three approaches produced different speeches (which is rare for the optimal and the holistic approach). Table 5 shows alternative descriptions generated by the three methods (without the preamble). A second example (showing similar tendencies) can be found in the appendix. The full query result, containing 20 fields, can be found in the appendix (Table 12). We posted 10 tasks, paying $1 per task and promising a bonus of $2 if worker's replies are close to the actual result. Each crowd worker was led to a separate Google sheets spreadsheet containing value combinations for the two dimensions (region and season, in the same order as in Table 12), and links to the three voice descriptions. We instructed users to "listen carefully" to voice descriptions and to "estimate values in each row". We posted tasks at 8:39 PM PDT, received nine replies, and eliminated one duplicate submission by the same crowd worker.

Table 5: Speeches generated by different approaches summarizing the query result from Table 12.

Approach	Speech	Quality
Optimal	Around two percent is the Average cancellation probability. Values increase by 50 percent for flights starting from the North East. Values increase by 100 percent for flights scheduled in Winter.	0.25
Unmerged	Over ten percent is the Average cancellation probability. Values increase by 100 percent for flights starting from the West. Values increase by 50 percent for flights scheduled in Winter.	0.0
Holistic	Around one point five percent is the Average cancellation probability. Values increase by 100 percent for flights starting from the North East. Values increase by 100 percent for flights scheduled in Winter.	0.24

Table 6: Absolute error (%) made by users estimating all result fields based on different speech descriptions.

User	Optimal	Holistic	Unmerged
1	**27.2**	39.4	34.4
2	1.16	**0.81**	12
3	1.16	**0.84**	11.7
4	1.16	**0.8**	11.9
5	5.9	**0.72**	11.7
6	1.2	**0.8**	12.2
7	1.6	**0.8**	11.9
8	**50**	56	52
Median	1.4	**0.81**	12

Table 6 summarizes results. We compare the user's estimate for each result field against the actual value, and average the absolute distance over all result fields. The general tendencies correlate well with our initial quality estimates. Optimal and holistic approach lead to very similar results, the unmerged speech fares far worse. We observed two outliers (user 1 and 8), who we believe misunderstood speech fragments such as "values increase by 100%" to mean "values increase **to** 100%". If using our system regularly, such misunderstandings would certainly be resolved. For six out of

eight users, the holistic speech leads to an average error of rounded 0.8% at most. Given the extreme constraints of the scenario, we believe that those results are promising. Further, detecting relative tendencies in data is in certain scenarios (e..g, if users are interested in extreme values) more important than absolute values. We show in the appendix that users interpret relative tendencies correctly in the majority of cases for speeches generated by the optimal or holistic approach.

5.2 Exploratory Analysis Study

We compare our vocalization method against a baseline [24] in a user study. We recruited participants on the AMT platform. Participants had access to a Web interface for voice-based OLAP. The Web interface allows users to switch freely (i.e., for each single query) between the two compared vocalization methods. Users can also choose between two input modes: either users issue voice commands or commands are entered via keyboard. Our research focus in this paper is on vocalization, i.e. voice output. Our component for interpreting user input is rather simple and based on keywords. Users can drill down, roll up, and add or remove dimensions in the OLAP result by mentioning related keywords. Users can request help (via keyboard or voice input) to obtain all available keywords (via voice output). We log user queries on the server and asked users to provide their crowd worker ID. This allowed us to correlate user preferences to the queries they issued.

We issued 40 human intelligence tasks (HITs) in total, 20 hits for analyzing the aforementioned data set about flight cancellations, and 20 hits for analyzing a small data set about average salary (we provide more details on those data sets in the appendix). Our tasks were open to any crowd worker and we paid a reward of 50 cents per task. We posted the study at 9:33 PM PDT and associated it with the keywords "user study, data analysis, voice output". We elicited input from crowd workers via the instructions "Use the interface for at least five minutes to answer the following questions", followed by questions about data ("State five non-trivial facts you learnt from your analysis"), input method preferences ("Which input method did you prefer", "Why do you prefer this input method"), and output method preferences ("What output method do you prefer", "Why do you prefer this output method").

Users issued in average 26 and up to 125 queries per session (we only count queries that were parsed correctly and trigger vocalization, excluding help requests). Table 7 shows a few, representative example facts extracted by different crowd workers from the flight data set. We also report the dimensions that the fact refers to. As those examples demonstrate, workers were able to extract facts related to all dimensions of the data set. Workers were able to extract high-level insights relating to large subsets of the data (e.g., flights in the northeast) as well as claims about small sub-populations (e.g., flights operated by one specific airline in Orlando). We saw examples of workers combining results from multiple queries into one claim (e.g., the third claim combines data that can only be retrieved via separate queries due to the structure of our dimension hierarchies). We also saw examples of workers combining insights learned from the data with data from other sources (e.g., the first claim relates higher cancellation probabilities in Winter to associated weather conditions).

Asked about input preferences, about one quarter of users (nine out of 40) preferred keyboard input over voice input. Some of the cited reasons do not intrinsically relate to the quality of that input method (e.g., reasons such as "I have no microphone" or "Only because I was in a noisy environment"). Others cite problems with speech recognition and faster keyboard input. The reasons behind a voice input preference include "Easier than typing" or "There is something rewarding about getting info 'talking' with the computer".

Table 7: Example facts extracted by crowd workers via voice-based data analysis.

Dimensions	Fact
Flight date	the main cancellation probability are on winter and so thanks to the weather, around 2% is the average cancellation probability
Airline, Start airport	An Alaska Airlines flight is 200% more likely than normal to have a cancellation from Orlando.
Start airport	The greatest cancellations are in the northeast, and it seems there may be a trend where the middle of the country otherwise sees some of the higher cancellation rates, as places like Iowa/Illinois/Arkansas had fairly high values, while places like Florida/Georgia/Minnesota were less.

Table 8: Preferences of crowd workers with regards to vocalization, comparing a prior vocalization method against this approach. Plus indicates slight preference, repeated plus indicates strong preference.

Data	Prior++	Prior+	Neutral	This+	This++
Salary	2	1	7	4	6
Flights	0	3	4	5	8

Table 9: Length of speeches (in characters) generated during data analysis by crowd workers.

Scenario	Aggregate	This	Prior
Salary	Average	146	240
	Maximum	400	2500
Flights	Average	190	1219
	Maximum	432	54838

Voice input is not the focus of this work. Still, it is interesting that a non-negligible part of users prefer voice input even with a relatively simple implementation.

Table 8 reports user preferences with regards to the vocalization methods. Users chose between slight or strong preference for our approach or the baseline, or indicated equal preference for both methods. Clearly, most users express a preference for our approach. This effect is stronger for the flights data set. The reasons for a baseline preference included a higher degree of detail as well as a preference for the output format resembling bullet points. The majority of users prefer our approach, citing long waiting and output times with regards to the baseline and ease of understanding as reasons. Many users based their preferences on speech length. We analyzed the logs to see whether those claims are based on actual tendencies.

Table 9 analyzes the length (in characters) of speeches generated over the user study. Results are broken down by method and scenario. Clearly, speeches generated by our method are more concise. The difference is more pronounced for the flights data set. This is expected as it features more dimensions (and the number of aggregates reported by the baseline in the worst case increases exponentially in the number of dimensions). For the flights data set, speeches generated by the baseline are over six times longer in average. The maximal baseline speech length exceeds the maximal speech length of our approach by two orders of magnitude. The reasons cited by users for preferring our approach relate therefore to measurable effects. Overall, generating concise summaries with length restrictions seems more appropriate than the baseline approach for OLAP-style data analysis.

6 RELATED WORK

There are two recently proposed algorithms for vocalizing data sets [24, 25]. We compare against the first vocalization method [25] in our experimental evaluation. The method differs in multiple aspects from our approach. First, it does not interleave query processing and result vocalization. Second, it does not allow to place limits on speech output length. Third, it uses different methods (i.e., linear programming and greedy algorithms) to generate speeches. Altogether, the prior baseline is not optimized for vocalizing OLAP results (which can be large) in an interactive scenario.

The second vocalization approach [24] is specific to time series data. It is not applicable to tabular OLAP results. The focus on time series data motivates a specific speech structure (a sequence of patterns) and optimization algorithm (based on dynamic programming). Also, the processing algorithm requires data to be indexed according to query range conditions. In case of OLAP, it is typically not possible to index data according to all possible combinations of dimension values. The time series approach uses a method based on optimal experimental design to determine interesting time points for data retrieval. This would not make sense in our scenario: we cannot efficiently retrieve previously chosen parts of the query result without indexes. Altogether, the prior approach makes fundamentally different design decisions compared to our work.

Our work relates to prior work proposing tailored processing strategies to generate visualizations efficiently [11–13]. Our work differs by its focus on voice output instead of visual output. This focus creates specific challenges (i.e., conciseness constraints) as well as specific opportunities (e.g., leveraging the sequential nature of voice output).

Furthermore, this work relates to prior work on OLAP processing and user interaction. For instance, our user model is based on prior work in the context of user-adaptive exploration [9, 18, 22]. Our work is similar in that we present a user-centric metric quantifying the value of information transmitted, followed by a method that efficiently selects which information to transmit. Of course, our work is specific as it focuses on a very specific output mode. Also, this work connects to prior work on using sampling to answer OLAP queries [3, 8, 10, 15]. However, we do not need to approximate a fixed set of aggregates with a certain precision. Instead, we only need to determine the speech approximating the query result aggregates best.

Finally, our work relates to prior work in sonification and auditory display [7, 21]. The main difference between our and prior work is the focus on voice output (as opposed to non-speech sounds) and large data sets. This motivates our approach for interleaved processing and voice output.

7 CONCLUSION

We introduce the problem of generating voice answers to OLAP queries. We introduce a speech model and a speech generation approach. Our approach is tailored to the particularities of our scenario. We demonstrate significant speedups over baselines and better user preference results in extensive experiments. In future work, we plan to improve the voice recognition component of our current system.

A FORMAL ANALYSIS

A.1 Speech Properties

Each speech starts with an absolute claim on result averages, the baseline statement. We prove formally that our recursive formula from Section 3.4 creates belief models that are consistent with the baseline statement.

Theorem A.1. *We assign each speech to a user belief model that is consistent with its baseline statement.*

Proof. We use structural induction. The theorem holds trivially for speeches that consist only of a single baseline statement (induction start). In that case, we assign each result aggregate to the same baseline value and are therefore consistent. Assume now (induction step) that $\mathcal{M}$ yields a consistent assignment for speech t. We show that the same must hold for speech $t \circ r$ where r represents an arbitrary refinement. Denote by m the number of aggregates within the scope of r and by n the total number of result aggregates. Further, denote by Δ the absolute change assigned to aggregates within the scope. Hence, $-m \cdot \Delta/(n-m)$ is the change assigned implicitly to aggregates outside of r's scope. Summing over the change in all aggregates yields $m \cdot \Delta - m \cdot (n-m) \cdot \Delta/(n-m) = 0$. As the assignment for t was consistent with its baseline (i.e., the average of expected values over all aggregates yields the value postulated in the baseline), the new assignment for $t \circ r$ is, too. □

A.2 Complexity Analysis

We analyze time complexity of our algorithm. Our algorithm is an anytime algorithm that continues sampling as long as voice output is not finished. Hence, we analyze its complexity per iteration and its pre-processing overheads.

We denote by k the maximal number of speech fragments (k is upper-bounded according to user preferences). By m, we denote the maximal number of children of any node in the search tree. This is a property of the speech grammar and not of the input query.

The following lemma refers to Function SpeechDBeval from Algorithm 3.

Lemma A.2. *Evaluating a speech with regards to one aggregate and sample takes $O(k)$ time.*

Proof. We can pick a suitable aggregate in constant time (we assume that the cache stores aggregates for which information is available in a structure allowing constant-time random access). Our cache-based estimate is derived from a fixed-size cache sample. User belief for a specific aggregate can be calculated independently from other aggregates. To calculate the distribution for one specific aggregate, we require $O(1)$ operations per speech fragment. □

The following lemma refers to Function Sample from Algorithm 2.

Theorem A.3. *Sampling the search tree takes $O(k \cdot m)$ time.*

Proof. We traverse at most k nodes on the path from the root to a leaf. For each visited node, we must compare all its children to identify the most interesting one. Hence, the time for identifying the most interesting speech to evaluate is in $O(k \cdot m)$. According to the previous lemma (and as updating the statistics takes only time in $O(k)$), the time complexity for this operation dominates. □

The latter theorem describes the complexity of an iteration of the inner while-loop of Algorithm 1. Having a low complexity for the inner loop is important: otherwise, we might have a perceptible pause between two sentences in voice output. The low complexity of $O(k \cdot m)$ makes this extremely unlikely, matching our experiences in practice. We analyze time complexity of the pre-processing step of Algorithm 1.

Theorem A.4. *Pre-processing time is in $O(m^k)$.*

Proof. Each node in the search tree has at most m children. Hence, the maximal number of nodes in the search tree is in $O(m^k)$. Initializing all fields for a node takes constant time. □

This complexity is acceptable for two reasons. First, pre-processing is overlapped by a first sentence in voice output. Hence, pre-processing is typically not on the critical path. Second, k is due to limits of human perception (maximal number of speech fragments that can be kept in short-term memory). Such limits cannot be expected to scale. Hence, prior work considers those limits to be constants during complexity analysis [25]. Under those assumptions, complexity of pre-processing is polynomial.

B ADDITIONAL EXPERIMENTS

We report additional experimental results and provide more details on the setup of our studies.

B.1 Implicit Assumptions Pilot Study

Table 10 provides more details on our pilot study on implicit assumptions. We test whether users tend to make certain assumptions in the absence of more specific information. Due to the need for concise voice output, this scenario is the rule. We use the results of this user study to model how users react to hearing voice output.

Our task was open to crowd workers with an average acceptance rate of at least 50%. We paid five cents per task and promised a bonus of five cents if worker answers match the majority opinion. The study was started at 10 AM PDT.

Table 10: Detailed results for pilot study on implicit assumptions.

Model Aspect	Question	Answer 1	Answer 2	Answer 3	#Replies (1/2/3)
Symmetry	Assume the typical salary is $10. Which of the following options seems most likely to you?	Most people get more than $10 salary	About half the people get less and half the people get more than $10 salary	Most people get less than $10 salary	3/15/2
Concentration	Assume the typical salary is $10. Which of the following options seems most likely to you?	A salary between $10 to $15 is more likely than one between $15 and $20	A salary between $10 to $15 is equally likely as one between $15 and $20	A salary between $15 and $20 is more likely than one between $10 and $15	15/4/1
	Again, assume the typical salary is $10. Which of the following options seems most likely to you?	A salary between $5 to $10 is more likely than a salary between $1 to $5	A salary between $1 to $5 is equally likely as a salary between $5 and $10	A salary between $1 to $5 is more likely than a salary between $5 to $10	13/5/2
Variance	Assuming the typical salary is $10. Which percentage of people are paid more than $15?	Between 0% and 20%	Between 20% and 40%	Between 40% and 60%	11/8/1
	Assuming the typical salary is $10. Which percentage of people are paid less than $5?	Between 0% and 20%	Between 20% and 40%	Between 40% and 60%	17/3/0
	Assume the typical salary is $100. Which percentage of people are paid more than $150?	Between 0% and 20%	Between 20% and 40%	Between 40% and 60%	11/7/2
	Again, assume the typical salary is $100. Which percentage of people are paid less than $50?	Between 0% and 20%	Between 20% and 40%	Between 40% and 60%	10/7/3
Uniformity	Assume the average salary over cities A and B is $10. Without further information, what do you assume about the salary distribution?	The salary in city A is higher	The salary in city A is about the same as in city B	The salary in city B is higher	4/15/1
Composition	Salary doubles for profession A compared to the average. It also doubles when living in city B. What is your salary estimate for a person with profession A living in city B?	The same as average	Two times higher than average	Four times higher than average	4/9/7
	Salary halves for profession A compared to the average. It doubles when living in city B. What is your salary estimate for a person with profession A living in city B?	The same as average	Two times higher than average	Four times higher than average	14/3/3

We used the keywords "user study, data, filling in gaps" to advertise the study on AMT. Table 10 reports the questions we used to test implicit assumptions. Each question focuses on verifying one of the hypotheses outlined in Section 3. Each question corresponds to one AMT human intelligence task. Questions were asked (AMT batch order) in the order in which they appear in Table 10. We elicit replies via the following instructions: "We study how users fill in missing information in a description. Answer the following question by gut feeling." We offered three answer possibilities per question (as a radio button, options appear top down from Answer 1 to Answer 3). For most questions, two of those answer possibilities are inconsistent with our hypothesis. As can be seen from the detailed results, the hypotheses are verified in most cases. Typically, around three quarters of crowd workers provide an answer supporting our hypothesis.

B.2 Experimental Setup

We provide more details on the experimental setup in Section 5. We use two data sets for the experiments: a data set correlating mid-career salary with other factors [2] and data on flight cancellations in 2015 [3]. We deliberately selected

[2] https://www.kaggle.com/wsj/college-salaries
[3] https://www.kaggle.com/usdot/flight-delays

Table 11: Statistics on benchmark data.

Data Set	Dimensions	#Rows	Size
Mid-career salary	College location, start salary	320	36 KB
Flight cancellations	Flight date, airline, start airport	5.3 million	600 MB

one very small data set (the salary data set with 36 KB) and a larger data set (data on flights with 600 MB). This enables us to demonstrate how user preference and processing overheads vary as a function of data size for different query evaluation and vocalization methods. For salary data, we consider the dimensions college (with three levels: region, state, and specific institution), and the dimension start salary (with two levels: rough start salary and precise start salary). For flights data, we consider three dimensions: the start airport (with four levels: region, state, city, and airport), the flight date (with levels season and month), and the airline (one level). Table 11 summarizes basic statistics with regards to the two data sets.

Our approach can be tuned by the granularity at which numerical values are represented. Following prior user studies [24], we reduce precision of voice output to one significant digit for numerical values. We restrict the output length of the main speech (without preamble) to 300 characters. This value is recommended for voice-based interactions [5]. We compare our approach against a prior vocalization method [25]. By "Prior", we refer to this baseline in all plots. Among the three prior algorithms, we select the greedy version. The greedy algorithm has been shown to realize good tradeoffs between planning time and output quality. We set configuration parameters $m_S = m_C = 1$ which are the settings used in the baselines user study.

We implemented both algorithms in Java 1.8. We use Postgres 9.6 as underlying database engine. For our user studies, we made both approaches accessible via a Web interface. The server-side component of the Web version was implemented using the JEE framework [4]. The Web client uses JavaScript to send asynchronous requests to the server. Speech recognition and voice output are realized via the ResponsiveVoiceJS API [5] (which is based on the Google speech-to-text service).

We run the performance benchmarks in Section 5.1 on an MacBook Air computer with a 2.2 GHz Intel Core i7 processor and 8 GB of RAM. Our Web interface runs on the Heroku platform [6], using the "basic" service plan for the database backend. All crowd user studies are performed on the Amazon Mechanical Turk (AMT) [7] platform.

[4]http://www.oracle.com/technetwork/java/javaee/

[5]https://responsivevoice.org/

[6]https://www.heroku.com/

Table 12: Full result for a query about flight cancellation probabilities, broken down by region and season.

Region	Season	Cancellation
the North East	Winter	0.0555
the Midwest	Winter	0.03944
the South	Winter	0.02851
the North East	Spring	0.02296
the Midwest	Summer	0.018
the North East	Summer	0.01662
the South	Spring	0.01656
the Midwest	Spring	0.01576
the West	Winter	0.01562
the United States territories	Winter	0.01424
the Midwest	Fall	0.01313
the South	Summer	0.01097
the West	Summer	0.00927
the North East	Fall	0.00794
the United States territories	Summer	0.00741
the West	Spring	0.00725
the United States territories	Spring	0.0065
the West	Fall	0.0056
the South	Fall	0.00537
the United States territories	Fall	0.00183

B.3 Speech Quality Comparisons

We provide additional details on the speeches generated by different approaches. First, Table 12 shows the detailed query result described by the speeches in Table 5. Table 13 shows speeches generated for another query, the tendencies are similar. Table 14 is complementary to Table 6 (reporting absolute errors). It reports the number of relative tendencies in the query result that were correctly identified by users. Given two estimates e_1 and e_2 for two result fields with actual values v_1 and v_2, we count the tendency as correctly identified if $e_1 < e_2$ and $v_1 < v_2$ or $e_1 \geq e_2$ and $v_1 \geq v_2$. Table 14 reports percentage of correctly identified tendencies over all query result field pairs. The tendencies are the same as for Table 6.

ACKNOWLEDGMENTS

This research was supported by the Google Faculty Research Award "Optimizing Voice-Based Output of Relational Data".

REFERENCES

[1] CB Browne and Edward Powley. 2012. A survey of monte carlo tree search methods. *Trans. on Computational Intelligence and AI in Games* 4,

[7]https://www.mturk.com/

Table 13: Speeches generated by different approaches answering the same query. The full result has 378 fields, quality is measured via our user belief model.

Approach	Speech	Quality
Optimal	Five to ten percent is the Average cancellation probability. Values increase by 50 percent for flights operated by American Eagle Airlines Inc..	0.3
Unmerged	Around a quarter percent is the Average cancellation probability. Values increase by 200 percent for flights starting from Massachusetts. Values increase by 100 percent for flights scheduled in December.	0.05
Holistic	Five to ten percent is the Average cancellation probability. Values increase by 50 percent for flights starting from Massachusetts. Values increase by 50 percent for flights scheduled in February.	0.28

Table 14: Correct relative tendencies in user input (%).

User	Optimal	Holistic	Unmerged
1	**71**	70	55
2	**73**	72	56
3	**73**	72	56
4	**73**	72	56
5	**73**	70	56
6	63	**72**	53
7	**73**	72	56
8	61	**63**	46
Total	70	70	54

1 (2012), 1–49. http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6145622

[2] Dharmil Chandarana, Vraj Shah, Arun Kumar, and Lawrence Saul. 2017. SpeakQL: towards speech-driven multi-modal querying. In *HILDA*. 1–6.

[3] Yu Feng and Shan Wang. 2002. Compressed data cube for approximate OLAP query processing. *Journal of Computer Science and Technology* 17, 5 (2002), 625–635. https://doi.org/10.1007/BF02948830

[4] Sylvain Gelly, L Kocsis, and Marc Schoenauer. 2012. The grand challenge of computer go: monte carlo tree search and extensions. *Commun. ACM* 3 (2012), 106–113. http://dl.acm.org/citation.cfm?id=2093574

[5] Google. [n. d.]. Google Assistant SDK. https://developers.google.com/assistant/sdk/overview.

[6] Silviu Guiasu and Abe Shenitzer. 1985. The principle of maximum entropy. *The Mathematical Intelligencer* 7, 1 (1985), 42–48. https://doi.org/10.1007/BF03023004

[7] Thomas Hermann, Andy Hunt, and John G Neuhoff. 2011. *The Sonification Handbook*. 301–324 pages. https://doi.org/10.1017/CBO9781107415324.004 arXiv:arXiv:1011.1669v3

[8] Ruoming Jin, Leo Glimcher, Chris Jermaine, and Gagan Agrawal. 2006. New sampling-based estimators for OLAP queries. In *ICDE*. 18. https://doi.org/10.1109/ICDE.2006.106

[9] Manas Joglekar, Hector Garcia-molina, and Aditya Parameswaran. 2015. Smart drill down. *VLDBJ* 8, 12 (2015), 1928–1931. arXiv:arXiv:1412.0364v1

[10] Shantanu Joshi and Christopher Jermaine. 2008. Materialized sample views for database approximation. *ICDE* 20, 3 (2008), 337–351. https://doi.org/10.1109/TKDE.2007.190664

[11] Uwe Jugel, Zbigniew Jerzak, and Gregor Hackenbroich. 2014. M4 : A Visualization-Oriented Time Series Data Aggregation. *VLDB* 7, 10 (2014), 797–808. https://doi.org/10.14778/2732951.2732953

[12] Niranjan Kamat and Arnab Nandi. 2017. InfiniViz: Interactive Visual Exploration using Progressive Bin Refinement. *arXiv preprint arXiv:1710.01854* (2017). arXiv:1710.01854 http://arxiv.org/abs/1710.01854

[13] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. 2015. Rapid sampling for visualizations with ordering guarantees. *VLDB* 8, 5 (2015), 521–532. https://doi.org/10.14778/2735479.2735485 arXiv:1412.3040

[14] Levente Kocsis and C Szepesvári. 2006. Bandit based monte-carlo planning. In *European Conf. on Machine Learning*. 282–293. http://www.springerlink.com/index/D232253353517276.pdf

[15] Xiaolei Li, Jiawei Han, Zhijun Yin, Jae-Gil Lee, and Yizhou Sun. 2008. Sampling cube: a framework for statistical olap over sampling data. In *SIGMOD*. 779–790. https://doi.org/10.1145/1376616.1376695

[16] Zhicheng Liu and Jeffrey Heer. 2014. The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization & Computer Graphics* 20, 12 (2014), 2122–2131. https://doi.org/10.1109/TVCG.2014.2346452

[17] 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.

[18] Patrick Marcel, Place Jean Jaurès, and Stefano Rizzi. 2012. Towards intensional answers to OLAP queries for analytical sessions. In *DOLAP*. 49–56.

[19] Robert B. Miller. 1968. Response time in man-computer conversational transactions. In *AFIPS*. 267–277. https://doi.org/10.1145/1476589.1476628

[20] Navneet Potti and Jignesh M. Patel. 2015. DAQ: A new paradigm for approximate query processing. *VLDB* 8, 9 (2015), 898–909. https://doi.org/10.14778/2777598.2777599

[21] Rameshsharma Ramloll, Wai Yu, and Beate Riedel. 2001. Using non-speech sounds to improve access to 2D tabular numerical information for visually impaired users. In *Conference of the British HCI Group*. 515–529. http://eprints.gla.ac.uk/3223/

[22] S. Sarawagi. 2000. User-adaptive exploration of multidimensional data. In *VLDB*. 307–316. http://citeseer.ist.psu.edu/sarawagi00useradaptive.html

[23] Ben Shneiderman. 1984. Response time and display rate in human performance with computers. *Comput. Surveys* 16, 3 (1984), 265–285. https://doi.org/10.1145/2514.2517

[24] Immanuel Trummer, Mark Bryan, and Ramya Narasimha. 2018. Vocalizing large time series efficiently. In *VLDB*. 1–12.

[25] Immanuel Trummer, Jiancheng Zhu, and Mark Bryan. 2017. Data vocalization: optimizing voice output of relational data. *VLDB* 10, 11 (2017), 1574–1585.