Vocalizing Large Time Series Efficiently

Immanuel Trummer, Mark Bryan, Ramya Narasimha
{it224,mab539,rn343}@cornell.edu
Cornell University

ABSTRACT

We vocalize query results for time series data. We describe a holistic approach that integrates query evaluation and vocalization. In particular, we generate only those parts of the query result that are relevant for voice output.

We exploit the fact that voice output has to be concise and simple to be understandable for listeners. Hence, the problem of generating voice output reduces to choosing between several coarse-grained alternatives. To make that choice, it is sufficient to evaluate the time series at a few carefully chosen locations. We use techniques from the area of optimal experimental design to choose optimal sampling points. Our algorithm is iterative and generates in each iteration a set of promising voice description candidates. We consider multiple metrics when generating voice descriptions, including the accuracy of description as well as its complexity and length. Then, we choose a near-optimal batch of sampling points to refine our choice between promising candidates.

We compare this algorithm experimentally against several baselines, demonstrating superior performance in terms of execution time and output quality. We also conducted a user study, showing that it enables users to execute simple exploratory data analysis via voice descriptions alone. We also compare against visual interfaces and sonification (i.e., non-speech sound) interfaces in terms of user performance.

PVLDB Reference Format:
Immanuel Trummer, Mark Bryan, Ramya Narasimha. Vocalizing Large Time Series Efficiently. *PVLDB*, 11 (11): 1563 - 1575, 2018.
DOI: https://doi.org/10.14778/3236187.3236206

1. INTRODUCTION

Imagine you are blind and want to become a data scientist. The problem of presenting data (e.g., query results) to users has received significant attention in the database community. However, prior work focuses nearly exclusively on visual interfaces. In this paper, we focus on the complementary problem of presenting data via voice output. We use the term *Vocalization* in the following.

Supporting visually impaired users is only one out of multiple reasons to consider voice output. The communication between user and computer is more and more shifting towards voice-based interfaces. This is evidenced by devices and services such as Google Home, Amazon Echo, or Apple's Siri. Voice interfaces, specifically for generic data analysis, are currently under development [5, 19]. In all those cases, recent research has rather focused on improving the capability of the system to understand voice queries by the user. We focus on the complementary problem of transmitting query results via voice output. While such mechanisms are required for any voice query interface, their applicability is not restricted to this scenario. Even if queries are submitted via traditional interfaces, voice output can still be required for certain users (or useful to complement visual output).

We have recently proposed an approach for vocalizing small relational tables, containing several tens of rows [31]. Here, we vocalize query results that correspond to (potentially very large) time series. We specifically selected time series data for two reasons. First, this type of data is extremely popular and arises in many scenarios. Second, even large time series seem quite amenable for vocalization: it is typically possible to decompose a time series into coarse-grained patterns that capture the high-level developments. Based on a coarse-grained description of the entire time series, users can always "zoom in" and obtain more detailed descriptions for parts of a time series in an iterative process.

Voice output is subject to extreme constraints in terms of length and complexity. Reading text on screen allows users to quickly re-read passages and to adapt their reading speed for difficult parts. Also, users can quickly skim written text to identify relevant parts. None of that is easily possible via voice output. We conducted an initial user study to assess the limits within voice output is convenient for users. Our results corroborate prior claims [4, 20] and show that voice output has indeed to be short, simple, and coarse-grained. Otherwise, it exceeds the capabilities of typical listeners in terms of short-term memory and cognitive load.

This means that generating voice output is in certain ways more difficult than generating visualizations. The amount of information that can be transmitted to users is very limited. Hence, we need to be extremely thoughtful in selecting which aspects of a data set to transmit and which ones to neglect. We model vocalization as an optimization problem in which we optimize the accuracy of a description under constraints on length and complexity. While the limitations of typical listeners create challenges, they create at the same time opportunities to reduce processing overheads. If detailed query results cannot be transmitted due to the inherent limitations of voice output, it is wasteful to generate them at all.

Hence, we evaluate queries only partially. We "sample" the query result, a time series, by producing result subsets for small time intervals. Our goal is to collect just enough information to determine the best, coarse-grained voice description that complies with all constraints. Note that we assume that we can efficiently generate parts of the query result at specific time points. We discuss the conditions under which this assumption holds in more detail in Section 2.

We could select the "sampling points" at which we generate the query result randomly. However, some points may be more informative than others (e.g., points where alternative voice descriptions predict very different behavior are particularly helpful to determine the best alternative). At the same time, sampling creates computational overheads. Hence, we want to minimize the required number of samples. Collecting required information with minimal overhead is the general goal of methods proposed in the research area of optimal experimental design [27]. Our scenario maps to the scenario addressed by the latter research: sampling the query result is an "experiment" by which we gain information, the time points at which the query result is sampled are the "experimental parameters", finding the most appropriate voice description is the "experimental goal". Active learning and reinforcement learning are most closely related to optimal experimental design [29] since they also propose principled methods to choose what information to collect. They focus however on learning classifiers or policies which applies less to our scenario. Methods from the area of optimal experimental design have been shown to reduce information gathering overheads in various scenarios [26, 27]. Altogether, this justifies using such methods as base for selecting sampling points. Nevertheless, previously proposed methods need to be adapted to the particularities of our scenario (as discussed in more detail in Section 5).

We could collect samples first and generate the voice description that seems most appropriate, based on all samples, in a second step. However, this two-step approach misses opportunities: considering a subset of samples might allow us to select more informative points for the remaining samples. This motivates an iterative algorithm. The algorithm proposed in this paper iterates the following steps. First, it generates samples from a time series at time points chosen in the last iteration (or chosen randomly in the first iteration). Second, it uses bootstrap sampling to generate multiple resamples. Doing so allows us to evaluate uncertainties associated with the current sample. Third, it generates for each bootstrap sample an optimal vocalization. A vocalization is optimal if it describes the corresponding bootstrap sample as accurately as possible under constraints on length and complexity. Finally, we select a batch of near-optimal sampling points at which to evaluate the time series in the next iteration. To choose sampling points, we compare vocalizations generated for different bootstrap samples. We select points that are likely to narrow down our choice between those candidates.

We consider voice descriptions of a simple structure. We describe time series as a sequence of patterns, describing the evolution over parts of the time interval. During optimization, we consider a library of pattern templates, characterized by functions and associated text templates. We instantiate patterns by scaling them to a specific time interval and to a specific value range. We exploit the particularities of voice output and restrict scaling to time and value ranges that are easy to pronounce. For instance, our preliminary user study (see Section 7.1) indicates that recall increases significantly when limiting output values to only one significant digit. Also, the fact that we have to read out time intervals narrows down our choices significantly. For instance, the interval from 11:57:06 PST to 14:11:28 PST may be easy to write or to visualize but should be avoided for voice output (instead, we would rather approximate and say "from noon to 2 o'clock").

EXAMPLE 1. *Consider the following description of a financial time series: "The price is $400 from January to February. The price spikes to $600 in March. Finally, the price falls from $400 to $300 in April." Here, we have composed three simple patterns (one per sentence). The description rounds values to the most significant digit and rounds time interval bounds to entire months.*

Hence, when generating voice output for a time series covering a certain time interval, we only consider a relatively small set of time interval decompositions. We focus on decompositions into sub-intervals whose boundaries are quick to pronounce. Within each interval, we only need to consider an equally restricted set of pattern instantiations (i.e,. we instantiate a pattern by fixing a time and value range), referring to values that are easy to pronounce and to remember. We use dynamic programming to find an optimal pattern sequence for voice output. We associate time ranges with Pareto-optimal vocalizations, i.e. vocalizations that realize optimal tradeoffs between the metrics precision, length, and complexity. We start by generating the set of Pareto-optimal vocalizations for smaller time intervals. Later, we compose Pareto-optimal patterns for smaller intervals into potentially optimal pattern sequences for larger intervals. We show that this approach results in optimal vocalizations within the search space we consider. Searching for an optimal pattern sequence might be expensive if output could be long and complex. However, for the specific case of vocalization, output length and complexity is inherently limited by the abilities of the listener. This guarantees that the aforementioned approach is computationally efficient.

We compare the performance of our main algorithm against multiple baselines in diverse scenarios. We consider time series from different domains, generated via different types of SQL queries or retrieved via remote service calls. We also conducted a user study in which we compare different types of interfaces for an open-ended data analysis task. We compare vocalization against visual interfaces as well as against sonification methods (i.e., non-speech sound output to describe data). As expected, visual interfaces perform best in terms of user time investment and answer quality. Still, audio interface come relatively close according to those metrics, making them a reasonable choice in scenarios where visual output is not available. Among the two audio interfaces, most users had a clear preference for vocalization and used this method more frequently when being able to choose. Our performance results show that our main algorithm realizes the most practical tradeoff between output quality and computational overheads when generating vocalizations.

We summarize the original scientific contributions:

- We present a sampling-based algorithm that efficiently generates voice descriptions for large time series.

- We formally analyze the complexity and output quality of this algorithm.
- We evaluate our algorithm experimentally against several baselines. We compare vocalization against sonification and visual interfaces in a user study.

The reminder of this paper is organized as follows. We introduce our problem model in Section 2. In Section 3, we give an overview of our algorithm. Sections 4 and 5 describe in detail how the algorithm generates optimal voice output based on samples, and how it selects optimal time points for sampling. Section 6 analyzes our algorithm formally and Section 7 presents experimental results.

2. FORMAL MODEL

A vocalization is a voice description of a time series in the following. We consider vocalizations that are composed from atomic patterns, defined next.

Definition 1. A **Pattern** p is characterized by a piecewise linear function $f_p : [0, 1] \to [0, 1]$. Each pattern is associated with a text template t_p used for voice output. The text template contains placeholders for specific times and values (defined by a point on the pattern function) that become available once the pattern is instantiated. The text template may also contain placeholders for value units or value types.

We instantiate patterns to use them for vocalizations.

Definition 2. A **Pattern Instance** i is derived from a pattern p by fixing a time interval $[x_1, x_2]$, and a value interval $[y_1, y_2]$. We scale the pattern function f_p to cover those intervals. We say that the pattern instance predicts values in the time interval $[t_1, t_2]$, predictions are the values of the scaled function. A pattern instance is also associated with a text snippet that is derived from the pattern template t_p by substituting each placeholder.

We consider vocalizations of the following type.

Definition 3. A **Vocalization** is a sequence of pattern instances. Those pattern instances describe consecutive time intervals without overlap and without gaps, pattern instances are ordered by time. A vocalization is associated with a speech text. This text is formed by concatenating text associated with each pattern instance. Let t be a time point in the time interval covered by a vocalization. The vocalization contains a single pattern instance whose interval contains t. We say that the vocalization predicts the value for t that is predicted by the latter pattern instance.

Throughout the paper, we also use the terms **Voice Description** or **Speech** as synonyms for vocalization. Furthermore, we refer to the pattern instances that a vocalization is composed of also as **Speech Fragments**. We compare vocalizations according to the following metrics.

Definition 4. **Vocalization Length** is the number of characters in the text associated with a vocalization.

Definition 5. **Vocalization Complexity** is the number of pattern instances included in a vocalization.

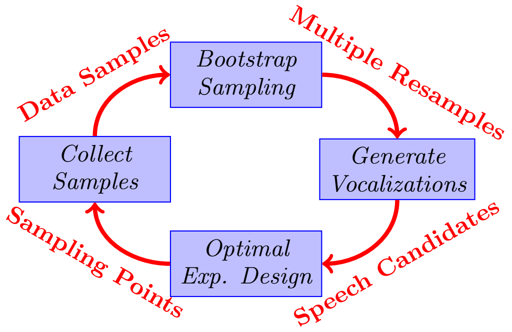


Figure 1: Overview of iterative algorithm.

Definition 6. **Vocalization Error**, with regards to a specific time series, is the mean squared error between actual and predicted values. More precisely, denote by $T = \{\langle x_i, y_i \rangle\}$ the points of a time series and by $v(x)$ the value predicated by vocalization v for time x. Then, the vocalization error with regards to T is defined as $\sum_i (y_i - v(x_i))^2 / |T|$.

We associate a vocalization with a **Cost Vector**, containing values for all three metrics. A cost vector $\vec{c}_1$ dominates vector $\vec{c}_2$, denoted as $\vec{c}_1 \preceq \vec{c}_2$, if $\vec{c}_1$ has lower or equivalent cost than $\vec{c}_2$ according to all three metrics.

Definition 7. An instance of **Optimal Time Series Vocalization** is characterized by a database D, a query q (the result of executing q on D must be a time series), and user preferences $\mathcal{P}.\vec{c}$, capturing upper bounds on speech length and complexity in a cost vector. An optimal solution is a vocalization with minimal vocalization error among all vocalizations satisfying length and complexity bounds (i.e., among all vocalizations with cost $\vec{c}$ such that $\vec{c} \preceq \mathcal{P}.\vec{c}$).

We generally consider SQL queries whose result consists of a time and a value column. The approach presented next assumes that all result rows for a given time interval can be generated efficiently (i.e., generating results for small time intervals is significantly cheaper than generating the entire query result). This is for instance possible for select-project queries on single tables for which an index on the time column is available. It is also possible for queries calculating aggregates and using the time column in the group-by clause (e.g., the query calculating daily crime counts in Chicago, used in our experimental evaluation, is of that form). The approach can be used for join queries as well, as long as the query properties allow to retrieve results for restricted time ranges efficiently (e.g., select-project-join queries with equi-joins on star schemata with time-indexed fact table, where dimension tables are indexed by the join column).

3. ALGORITHM OVERVIEW

We give an overview of an algorithm that efficiently generates voice output to describe a large time series. The algorithm is iterative and continuously improves the accuracy of voice output until a timeout is reached. Then, the final version is read out to the user.

Algorithm 1 is the main function of our algorithm. The input is a database, D, together with a query q on that database. Parameter $\mathcal{C}$ represents several tuning parameters that we discuss in more detail later. Parameter $\mathcal{P}$

```
1: // Generate best possible vocalization for result of
2: // query $q$ on database $D$ within time limit.
3: // Use configuration $\mathcal{C}$ and user preferences $\mathcal{P}$.
4: function VOCALIZE($D, q, \mathcal{C}, \mathcal{P}$)
5:     // Initialize query result samples
6:     $S \leftarrow$ random samples from $q(D)$
7:     // Iterate until timeout
8:     while Time limit $\mathcal{C}.\tau$ not reached do
9:         // Generate vocalization candidates
10:        for $i \leftarrow 1, \ldots, \mathcal{C}.\nu$ do
11:            // Re-sample sample with replacement
12:            $B \leftarrow$BOOTSTRAPSAMPLE($S$)
13:            // Best vocalization for bootstrap sample
14:            $v_i \leftarrow$BESTVOCALIZATION($q, \mathcal{C}, \mathcal{P}, B$)
15:        end for
16:        // Sample optimally to distinguish candidates
17:        $S \leftarrow S \cup$ OEDSAMPLES($D, q, \mathcal{C}, \{v_i\}$)
18:    end while
19:    // Return best vocalization based on full sample
20:    return BESTVOCALIZATION($q, \mathcal{C}, \mathcal{P}, S$)
21: end function
```

Algorithm 1: Generate a vocalization for a query result by interleaving optimization and sampling.

encapsulates several parameters capturing user preferences with regards to voice output. The algorithm returns an approximate voice description of the result when evaluating the query on the database.

We assume that the query result is a time series, i.e. scalar values that are associated with time points. The query may simply retrieve parts of an existing time series or the query may generate a time series from other data. The algorithm is applicable in both cases, as long as the following assumption holds: we assume that we can efficiently generate parts of the time series around specific time points (we formalize the term "efficiency" in this context later).

Generating the entire query result may be prohibitively expensive. Furthermore, voice output needs to be concise in general. This allows only to output a high-level description of the time series, focusing on the most important tendencies. It would be wasteful to generate the entire result as no details can be transmitted to the user. Because of that, Algorithm 1 aims at generating only those parts of the time series that are absolutely necessary in order to decide what voice output describes the data most accurately.

The algorithm is iterative. Iterations continue until a timeout, specified as one of the configuration parameters, is reached. Each iteration entails the following three steps. First, generate multiple bootstrap samples from the set of samples that we have collected so far from the query result (variable S). Second, generate for each of the bootstrap samples the most accurate voice description that is possible, under constraints imposed on voice output (e.g., the maximal length of the voice output, encapsulated as user preference in $\mathcal{P}$). Finally, we compare voice output candidates, generated based on different bootstrap samples. Based on that comparison, we select the time points at which to collect samples before the next iteration. After the final iteration, a last voice description is generated based on all samples collected so far. This description is read out to the user.

Figure 1 illustrates the primary steps of the algorithm. The algorithm is inspired by prior approaches from the domain of optimal experimental design. The focus in prior work was to select, among multiple candidates, a hypothesis that optimally describes a certain domain. Experiments can be executed to help narrowing down the choice between alternatives. By judiciously choosing the parameters of those experiments to make them as informative as possible ("optimal experimental design"), a decision can be reached much faster than via random experiments.

In our scenario, competing hypothesis correspond to competing high-level voice descriptions. Experiments correspond to partial query evaluations. The parameter for those experiments are the time points around which the query result is examined. We want to evaluate the time series at points that give us as much information as possible to help choosing between alternative voice descriptions. Unfortunately, the space of voice descriptions is typically prohibitively large. Hence, we do not consider all possible voice descriptions in our choice of sampling points. Instead, we consider only a few voice description candidates that seem promising according to the current data sample.

To generate multiple vocalization candidates, we could of course take separate samples from the time series. Each sample requires however typically to access the database and to read data from hard disc (or may potentially involve expensive data processing). It is typically much cheaper to re-sample, with replacement, the samples that we already have acquired before (and which are stored in main memory). Such bootstrap samples yield an inexpensive method to gain insights about the sensitivity of the voice descriptions towards different samples.

The next two sections describe two sub-functions of Algorithm 1 in more detail. Section 4 describes how we generate optimal voice descriptions for a given data sample (function BESTVOCALIZATION). Section 5 describes how we select the next time points to sample (function OEDSAMPLES).

4. FINDING OPTIMAL VOCALIZATIONS

Algorithm 2 generates an optimal vocalization, given a set of samples from a time series. An optimal vocalization minimizes vocalization error among all alternatives. Vocalization error is measured by comparing the approximate time series, described by the vocalization speech, against the actual data sample. We use the sum of squares error for comparison (which is equivalent to the mean squared error for a fixed number of samples).

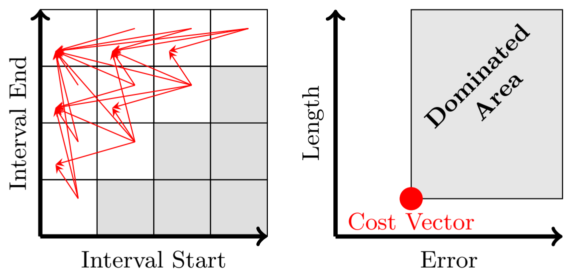
Function BESTVOCALIZATION is invoked by Algorithm 1 (presented in the last section). The input is a query q, algorithm configuration settings, encapsulated in parameter $\mathcal{C}$, and user preferences (parameter $\mathcal{P}$). Finally, we provide a set of samples S from a time series as input. The function returns a vocalization that describes the samples in S as precisely as possible among all vocalizations that satisfy constraints. For instance, the length of voice output and the number of sentences is typically restricted.

Algorithm 1 is based on dynamic programming. We decompose the problem of generating an optimal voice description for a certain interval into sub-problems, associated with sub-intervals. To make this approach viable, we require the vocalization error function to satisfy the principle of optimality. If that is the case, replacing a speech fragment describing a sub-interval by a more precise description, can only improve the entire speech. The mean squared error and many other metrics satisfy that property.

```
 1: // Insert vocalization $v$ into Pareto-optimal
 2: // vocalizations $V^*$ if admissible according to
 3: // user preference $\mathcal{P}$, prune dominated entries.
 4: procedure PRUNE($\mathcal{P}, V^*, v$)
 5:     // Check if new vocalization is admissible
 6:     if $v.\vec{c} \preceq \mathcal{P}.\vec{c}$ then
 7:         // Check if new vocalization dominated
 8:         if $\nexists v^* \in V^* : v^*.\vec{c} \preceq v.\vec{c}$ then
 9:             // Prune vocalizations dominated by new one
10:             $V^* \leftarrow V^* \setminus \{v^* \in V^* : v.\vec{c} \preceq v^*.\vec{c}\}$
11:             // Insert new vocalization
12:             $V^* \leftarrow V^* \cup \{v\}$
13:         end if
14:     end if
15: end procedure
16: // Vocalize result of query $q$ with minimal error w.r.t.
17: // sample $S$, using configuration $\mathcal{C}$ and considering
18: // user preferences $\mathcal{P}$.
19: function BESTVOCALIZATION($q, \mathcal{C}, \mathcal{P}, S$)
20:     // Divide into easily pronounceable intervals
21:     $\langle t_o, \ldots, t_g \rangle \leftarrow$TIMEGRID($q.interval$)
22:     for $0 \leq i < j \leq g$ do
23:         $V^*_{i-j} \leftarrow \emptyset$
24:     end for
25:     // Consider intervals of increasing size
26:     for $w \leftarrow 1, \ldots, g$ do
27:         // Iterate over time interval start
28:         for $s \leftarrow 0, \ldots, g - w$ do
29:             $e \leftarrow s + w$
30:             // Instantiate single patterns
31:             for $p \in \mathcal{C}.patterns$ do
32:                 $i \leftarrow$INSTANTIATE($p, [s, e], S$)
33:                 PRUNE($\mathcal{P}, V^*_{s-e}, i$)
34:             end for
35:             // Consider composite patterns
36:             for $d \leftarrow s + 1, \ldots, e - 1$ do
37:                 for $v_1 \in V^*_{s-d}$ do
38:                     for $v_2 \in V^*_{d-e}$ do
39:                         $i \leftarrow$COMPOSE($v_1, v_2$)
40:                         PRUNE($\mathcal{P}, V^*_{s-e}, i$)
41:                     end for
42:                 end for
43:             end for
44:         end for
45:     end for
46:     // Return complete vocalization with minimal error
47:     return $\arg\min[v \in V^*_{0-g}](v.error)$
48: end function
```

Algorithm 2: Find most precise voice description for given data sample under constraints on length and complexity.

Focusing on vocalization error alone is however insufficient. Voice output needs to be concise and easy to remember. We therefore restrict the length of speech output (e.g., measured as the number of characters) by an upper bound. Also, we restrict the number of atomic elements (typically associated with single sentences) that the voice description contains. Replacing a speech fragment by a more precise description can only decrease the vocalization error of the entire speech. However, if the replacement is longer or more complex than the original fragment, the resulting speech may violate constraints. This means that we need to take into account all relevant metrics, error, length, and complexity (measured by the number of atomic elements), at the same time, when comparing speech fragments that describe the same time interval.



(a) Dependencies between partial results in dynamic programming matrix. (b) Cost vector realized by vocalization with dominated area (in 2D).

Figure 2: Illustrating dynamic programming (left) and multi-objective pruning (right) of Algorithm 2.

Table 1: Example patterns with text templates.

Patterns	╱	╲	—	_/_
Text Template	rises from y_1 to y_2	falls from y_1 to y_2	remains at y	spikes to y_{max}

Algorithm 2 takes those considerations into account. Initially, the time range to which the query refers is divided into a grid. We choose grid points that can be easily pronounced. The granularity of the grid is chosen as a function of the query time range. For instance, we consider time intervals at the granularity of months if the query time range is between two months and one year. We switch to a granularity of quarters after that. We only consider a single time unit (e.g., months, quarters, or years) to keep output simple.

Next, Algorithm 2 iterates over time ranges of increasing length (loop in line 26). We consider time intervals that start and end at some of the previously chosen grid points. We start with single sub-intervals and consider finally the entire time range described by the query. Then, for each interval length (measured by the number of grid points between start and end), we iterate over all intervals of that length (loop in line 28). In each iteration, we collect speech fragments that optimally describe the current interval.

As justified before, we must consider multiple metrics to decide whether a given speech fragment is optimal. Hence, we are dealing with an instance of multi-objective optimization. In the pseudo-code, we denote by $v.\vec{c}$ the cost vector associated with a speech fragment (vocalization) v. We consider the three cost metrics approximation vocalization error, length, and complexity. We write $v_1.\vec{c} \preceq v_2.\vec{c}$ if vocalization v_1 dominates vocalization v_2 in terms of cost. This is the case, if v_1 has lower or equivalent cost according to all three metrics. If we compare two alternative vocalizations v_1 and v_2, referring to the same time interval, and $v_1.\vec{c} \preceq v_2.\vec{c}$, then we can safely discard v_2 during optimization.

We use variables V^*_{a-b} to store all Pareto-optimal vocalizations for the time interval ranging from grid point g_x to g_y. Being initially empty, those variables are filled in ascending order of interval size. We consider two ways of generating a vocalization for a given interval. First, we try instantiating all patterns from a library of pattern templates ($\mathcal{C}.patterns$). This yields vocalizations that consist of a single element. Second, we try dividing the current intervals into two parts and to cover each part by a speech fragment. This yields a composite vocalization that consists of at least two elements.

We discuss the first possibility, pattern instantiation, in more detail. A pattern is generally defined by several connected line fragments within a rectangle, together with an associated text template. We assume that patterns can be stretched along the time and value axis (y axis). Instantiating a pattern means to decide on scaling factors and to "position" the pattern by fixing coordinates for all corners of the associated rectangle. The extend on the time axis is determined by the time interval that we are considering. The extend on the value axis is determined by the sample data in the time range. We optimally match the pattern to the sample data under the constraint that all coordinates must be values that are quick to pronounce and easy to remember. More precisely, we only consider values with a single significant digit (we conducted a preliminary user study showing that users remember such numbers more easily). Function INSTANTIATE encapsulates the instantiation of a pattern, it generates a corresponding text description and calculates all associated cost metrics. Table 1 shows the patterns that we use in our current implementation.

When combining patterns, we consider all possibilities to divide the current interval into two parts. We consider intervals in ascending length order. Hence, dividing the current interval yields two intervals for which optimal vocalizations are already available. We combine those vocalizations via the COMPOSE function. It concatenates the text descriptions of the two component vocalizations, and calculates all three cost metrics for the new vocalization by adding up the cost vectors of the components. Figure 2(a) illustrates dependencies in dynamic programming: an arrow from a first time interval to a second indicates that vocalizations for the first are composed to form vocalizations for the second.

EXAMPLE 2. *Assume we describe data from February to September at month granularity. We first generate Pareto-optimal vocalizations for each of the eight months. Next, for each pair of consecutive months, we combine optimal single-month vocalizations and compare against single patterns newly instantiated for the two-month interval. Next, we treat three-month intervals, then four-month intervals and so on. Algorithm 2 performs eight iterations in total.*

Each newly generated vocalization is compared against all previously generated vocalizations for the same time interval. Based on that comparison, sub-optimal vocalizations are removed. Function PRUNE compares alternative vocalizations. First, it prunes out voice descriptions that violate length or complexity constraints (line 6, $\mathcal{P}.\vec{c}$ denotes upper bounds for all cost metrics determined via user preferences). Then, it checks whether any of the previously inserted vocalizations dominate the new one. If so, the new vocalization can be safely discarded. Otherwise, the new vocalization is inserted into the Pareto set, after pruning old vocalizations dominated by the new one. After iterations finish, variable V^*_{0-g} contains all Pareto-optimal vocalizations for the entire query time range. Figure 2(b) illustrates the area dominated by one cost vector in the cost space (the figure uses only two metrics while we generally consider three metrics).

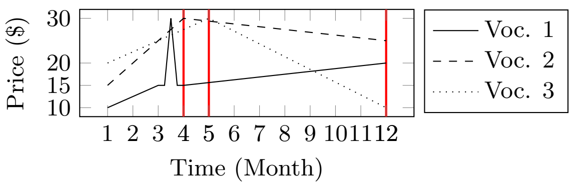


Figure 3: Promising sampling points (red) for comparing three vocalizations of the same time series.

Finally, we return the vocalization with minimal error among all Pareto optima (line 47).

5. SELECTING SAMPLING POINTS

The algorithm described in the previous section generates an optimal vocalization based on a small sample of the query result. We avoid generating the entire query result and collect result samples instead. As discussed in Section 2, we assume that we can generate result samples efficiently for given time points. In this section, we describe how to select the most informative time points to sample.

We choose a set of sampling points in each iteration of our main function (Algorithm 1). We pick sampling points based on a comparison of a set of candidate vocalizations, generated in the previous iteration. Our decision criterion is inspired by prior work in the area of optimal experimental design [26]: we pick sampling points where the developments described by different vocalizations diverge the most. To amortize database access and reasoning overheads, we select multiple sampling points in each iteration. Hence, intuitively, we choose a set of sampling points that offers a chance to prune out the maximal number of vocalization candidates. The following example gives a first intuition before we describe our method in detail.

EXAMPLE 3. *Figure 3 shows the developments described by three candidate vocalizations of the same time series (e.g., vocalization one is associated with the text "The price rises from \$10 to \$15 from January to March. Then, the price spikes to \$30 in March. Finally, the price rises from \$15 to \$20 until December."). Vocalizations one and two diverge maximally in April (May for vocalizations one and three, and December for vocalizations two and three). Hence, those months (marked in red) constitute interesting sampling points. If we have to choose two sampling points, picking April and May is sub-optimal as we cannot distinguish well between vocalizations two and three. Selecting April and December is better as it has the potential to prune any of the three vocalizations based on additional samples.*

Algorithm 3 selects an optimal set of sampling time points and returns a set of corresponding samples. It follows the principles that were discussed before. It takes as input a database D, a query q on that database, configuration parameters $\mathcal{C}$, and a set of promising vocalizations V.

The algorithm executes two steps. First, we determine a set of candidate time points for sampling (Function TIME-CANDIDATES). Second, we select a near-optimal subset of those time points to collect samples (Function GREEDYPICK). Samples for those points are returned. Note that the

```
1: // Collect candidate time points for sampling
2: // to compare vocalizations in $V$.
3: function TimeCandidates($V$)
4:     $T \leftarrow \emptyset$
5:     for $v \in V$ do // Over vocalizations
6:         for $p \in v.patterns$ do // Over patterns
7:             for $s \in p.lsegs$ do // Over line segments
8:                 $T \leftarrow T \cup \{s.x_1, s.x_2\}$
9:             end for
10:        end for
11:    end for
12:    return $T$
13: end function
14: // Greedily select sampling points from $T$ to
15: // compare vocalizations $V$ using configuration $\mathcal{C}$.
16: function GreedyPick($T, V, \mathcal{C}$)
17:     $S \leftarrow \emptyset$
18:     // Select sampling points one by one
19:     for $i \leftarrow 1, \ldots, \mathcal{C}.\beta$ do
20:         // Pick point with maximal utility gain
21:         $s^* \leftarrow \arg\max[s \in T](U(V, S \cup \{s\}) - U(V, S))$
22:         // Add that point into set
23:         $S \leftarrow S \cup \{s^*\}$
24:     end for
25:     return $S$
26: end function
27: // Using configuration $\mathcal{C}$, sample result of query $q$
28: // on database $D$ to optimally compare vocalizations $V$.
29: function OEDsamples($D, q, \mathcal{C}, V$)
30:     // Select candidate time points for sampling
31:     $T \leftarrow$TimeCandidates($V$)
32:     // Select optimal bounded cardinality subset
33:     $S \leftarrow$GreedyPick($T, V, \mathcal{C}$)
34:     // Return samples from selected locations
35:     return $\{q(D, s) | s \in S\}$
36: end function
```

Algorithm 3: Retrieve samples for time points selected via optimal experimental design.

resolution at which time points are considered depends not only on data but also on query properties. For instance, if a query calculates aggregates and groups by date, extracted from a time column with higher temporal resolution, then time points correspond to specific days.

We discuss the generation of candidate sampling times first. As discussed before, a sampling point becomes interesting if different vocalizations predict very different values for that time. Some sampling points might only be interesting to distinguish a subset of vocalizations (i.e., there is a subset of vocalizations predicting very different values for the given point while other promising vocalizations agree on a common value). We ultimately select sample batches and might combine sampling points that yield information on complementary subsets. Hence, we collect sampling point candidates that include for each pair of promising vocalizations the points of maximal difference.

Function TimeCandidates generates a set of potential sampling points, based on a set V of promising vocalizations. We compose voice descriptions from a fixed set of text templates. Each template describes a pattern from a pattern library. Each pattern is a piecewise linear function (i.e, connected line segments). Hence, each vocalization essentially describes a piecewise linear function. We collect the segment boundaries for each input vocalization. The points of maximal difference for each vocalization pair must be contained within. We prove this statement in Section 6.

We reduced the choice of sampling points from a continuous domain to a choice between discrete values. From those values, we select a near-optimal subset with bounded cardinality (defined by tuning parameter $\mathcal{C}.\beta$). We often want to pick more than one sampling point for performance reasons: data access is often associated with constant overheads and can be amortized over multiple queries when evaluating them in a batch. On the other side, we do not want to take samples for all candidate points. Doing so may be expensive and exceed the timeout (specified as input parameter to Algorithm 1) significantly. Also, taking samples yields additional information that can be used to select more informative sampling points for the next batch. Hence, parameter $\mathcal{C}.\beta$ must be chosen to realize a good compromise between the two extremes.

Function GreedyPick selects an optimal subset of sampling points for a given cardinality. We aim to maximize the following utility function U:

$$U(V, S) = \sum_{v_1, v_2 \in V} \max_{s \in S} |v_1(s) - v_2(s)| \tag{1}$$

. Here, V is a set of vocalizations and we denote by $v(s)$ the value implied by vocalization v for time point s. The utility function sums over all pairs of vocalizations the maximal distance over all sampling points. This function is an adaption of the total separation metric [26] which compares single sampling points. The original metric considers for a single point the absolute distances in predicted values over all hypothesis. We use the maximum operator to generalize that metric to sets of sampling points. The intuition behind choosing the maximum operator (as opposed to a sum for instance) is that we want to obtain a set of sampling points that is diverse in the following sense. We want to choose sampling points that help to distinguish as many pairs of vocalizations as possible. Hence, the utility value should not increase when selecting multiple sampling points that help to compare the same two vocalizations.

We show in Section 6 that this utility function has the submodularity property. This means, intuitively, that adding more sampling points tends to increase utility less and less. Furthermore, the function is nonnegative and monotone (i.e., adding more points cannot decrease utility). For functions with that property, the simple greedy algorithm (implemented in Function GreedyPick) is guaranteed to produce near-optimal solutions [22]. We analyze those quality guarantees as well as the complexity of sampling in Section 6.

6. FORMAL ANALYSIS

We first analyze the output quality of Algorithm 2. This algorithm shares some similarities with prior algorithms for constructing optimal histograms [17]. It differs however, in particular as it considers one additional cost metric (length of the verbal description) as opposed to approximation error and approximation size (i.e., speech complexity) alone.

Theorem 1. *Algorithm 2 produces an admissible vocalization with minimal error with regards to the input sample.*

Proof. Without pruning, the algorithm would produce all vocalizations that can be formed with the given pattern library. The final statement (line 47) selects the vocalization with minimal error among the ones satisfying bounds on length and complexity. Hence, we only need to show that pruning does not discard any speech fragments that would be required to form the optimal vocalization in the entire search space. The optimal vocalization is the one minimizing error while satisfying all constraints. During pruning, a speech fragment is discarded if it violates bounds on length and complexity. As both metrics are monotone, the cost of a speech fragment lower-bounds the cost of any vocalization it is a part of. Hence, speech fragments pruned for bound violations cannot be part of an optimal speech. Also, a speech fragment v is pruned if there is another speech fragment v^* with lower or equivalent cost according to all three metrics (error, length, complexity). All three cost metrics are additive (i.e., the cost of a composite speech is the sum of the cost of its components). Hence, for any speech that contains v as a fragment, we can form a speech with equivalent or better cost by replacing v by v^*. Hence, discarding dominated speech fragments does not prevent the algorithm from producing an optimal vocalization. □

The next two theorems refer to Algorithm 3 which chooses sampling points and collects corresponding samples.

THEOREM 2. *For each vocalization pair, a time point maximizing their distance is among the sampling candidates.*

PROOF. The sampling candidates consist of the end points of all line segments that are part of pattern sequences described by vocalizations. We conduct a proof by contradiction. Assume that a time point t^* of maximal distance is not contained in the sampling candidates for two specific vocalizations v_1 and v_2. Denote by l_1 the line segment described by v_1 at t^* and by l_2 the corresponding line segment associated with v_2. Further, denote by Δ_1 the slope of l_1 and by Δ_2 the slope of l_2. If $\Delta_1 = \Delta_2$ then both line segments are parallel and their distance is constant (hence, the candidate set contains a point of maximal distance). If $\Delta_1 < \Delta_2$ and l_1 lies below l_2 then distance between the two line segments increases with increasing time t. Hence, there is a $t > t^*$ maximizing distance and thereby leading to a contradiction. Similar contradictions are obtained for the remaining cases. □

THEOREM 3. *The set of sampling points picked by the greedy algorithm realizes a utility value within factor $1-1/e$ of the optimum.*

PROOF. Consider the previously introduced utility function $U(V,S) = \sum_{v_1,v_2 \in V} \max_{s \in S} |v_1(s) - v_2(s)|$. This function is nonnegative as it adds up nonnegative terms. It is monotone in S as the maximum is a monotone operator. Also, it has the property that $U(V, S \cup \{s\}) - U(V,S) \geq U(V, \widetilde{S} \cup \{s\}) - U(V, \widetilde{S})$ for sets S and $\widetilde{S}$ with $S \subseteq \widetilde{S}$ and element s. In other words: the utility function is sub-modular in S. This can be seen as follows. First, the function $F(S) = \max_{s \in S} |v_1(s) - v_2(s)|$ is sub-modular in S for fixed vocalizations since having more elements in S can only decrease the marginal gain when adding one new element. Second, the utility function is therefore a sum over sub-modular functions with nonnegative coefficients (which implies sub-modularity in general). The postulated bound follows immediately from the bounds of the greedy algorithm by Nemhauser and Wolsey [22] for maximization of functions that are nonnegative, monotone, and sub-modular. □

Next, we analyze time complexity. We denote by n the size of the input data set, by m the number of patterns in the library, and by g the number of grid points into which we divide the query time range when generating voice descriptions (line 21 in Algorithm 2). Furthermore, denote by l the maximal speech length measured in characters and by c the maximal complexity as measured by the number of patterns (parameter $\mathcal{P}.c$ in Algorithm 2). We first bound the number of entries per interval maintained by Algorithm 2.

LEMMA 1. *Algorithm 2 stores at most $O(c \cdot l)$ vocalizations per time interval.*

PROOF. A vocalization is composed of patterns. Each pattern increases complexity by one and length by at least one (as the associated text template must be non-empty). Hence, vocalizations can only realize admissible length values between one and l, and complexity cost values between one and c. The pruning function discards all vocalizations exceeding the bounds before storing them. For a fixed interval, it also discards vocalizations that are dominated by previously generated ones. Hence, we store for each combination of length and complexity values at most one vocalization with minimal error. □

Next, we calculate time complexity of Algorithm 2. In addition to the notations introduced before, we denote by d the number of samples provided as input for Algorithm 2.

THEOREM 4. *Generating an optimal vocalization has time complexity in $O(g^2 \cdot (m \cdot (d + c \cdot l) + g \cdot (c \cdot l)^3))$.*

PROOF. The algorithm iterates over $O(g^2)$ intervals. For each interval, we consider vocalizations that can be formed using a single pattern and composite vocalizations. There are m vocalizations that can be formed from single patterns and $O(g \cdot (c \cdot l)^2)$ composite vocalizations per interval (since we consider $O(g)$ interval splits, and $O(c \cdot l)$ Pareto-optimal vocalizations for each of the two sub-intervals). The most expensive operation when instantiating a pattern is to determine the value range within a given time interval. This is achieved in $O(d)$. Composing two patterns only requires to concatenate their text and to calculate the cost vector for the resulting speech (which requires constant time as the number of cost metrics is constant). Algorithm 2 invokes the pruning function for each generated vocalization. The pruning function compares the newly generated vocalization against all Pareto-optimal vocalizations for the same interval. Pruning complexity is in $O(c \cdot l)$ (using Lemma 1). □

We analyze the time complexity of Algorithm 3. The complexity of taking a single sample may vary depending on the properties of data, query, and storage engine. We assume in the following that data is indexed by time and that the time complexity for taking one sample is in $O(\log(n))$. Let ν be the number of vocalizations that Algorithm 3 receives as input and λ the maximal number of line segments per vocalization. Parameter β is the number of sampling points.

THEOREM 5. *Collecting samples has time complexity in $O(\beta \cdot (\log(n) + \nu\lambda))$.*

PROOF. Selecting candidates sampling points is in time $O(\nu \cdot \lambda)$. Greedily selecting a subset of candidates is in $O(\beta \cdot \nu \cdot \lambda)$. Sampling time is in $O(\beta \cdot \log(n))$. □

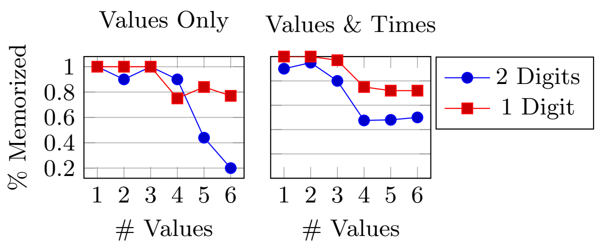
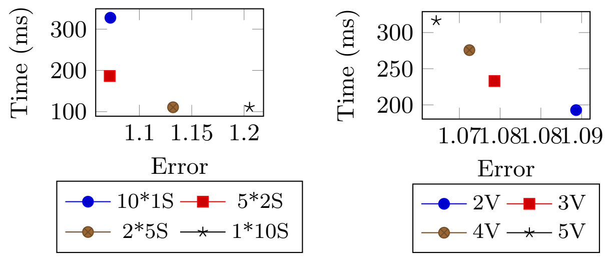


Figure 4: Percentage of values and times remembered by users after listening to voice output.



(a) Partitioning samples over more iterations increases run time and decreases error.

(b) More vocalization candidates increase run time and decrease error.

Figure 5: Impact of tuning parameters.

The latter two theorems imply the complexity of our algorithm for vocalization of large time series (Algorithm 1) as a function of the number of iterations i.

THEOREM 6. *Algorithm 1 has time complexity in* $O(i \cdot (\nu \cdot g^2 \cdot (m \cdot (i \cdot \beta + c \cdot l) + g \cdot (c \cdot l)^3) + \beta \cdot (\log(n) + \nu\lambda)))$.

PROOF. This is an immediate implication of the latter two theorems as sampling and optimization are the sub-functions with dominant time complexity. We also exploit that the number of samples in iteration i is bounded by the number of samples $i \cdot \beta$ collected in prior iterations. □

The complexity of the algorithm is polynomial in the input parameters. Furthermore, its complexity grows only logarithmically in the data set size.

7. EXPERIMENTAL EVALUATION

In Section 7.1, we summarize the results of a preliminary user study that motivates the type of vocalization we consider. In Section 7.2, we compare our vocalization algorithm against multiple baselines in terms of output quality and computational overheads. Section 7.3 describes the results of a crowd user study, comparing visualization, vocalization, and sonification in a simple data analysis scenario.

7.1 Preliminary User Study

We performed an initial user study to compare different types of voice descriptions. We recruited five participants among the computer science students undergraduate students at our institution. We played computer-generated voice descriptions (based on manually generated text) of time series to participants, testing out alternative descriptions for similar test series. We varied the length of the description (e.g., the number of time points and values), the type of information given (e.g., times and values or values alone for equally spaced time intervals), and the text templates used to describe trends and tendencies. We compare alternative descriptions in terms of user preferences and in terms of retention by the listener. We measured the latter by asking users to write down times and values they remembered after listening to voice output. Each study took around one hour and we payed $10 to each participant.

Figure 4 shows an extract from our results. The left chart describes the results of a user study where we read out only values for equally spaced time intervals. The chart on the right side describes results for voice descriptions that contain both time points and values. We performed two separate test series in both cases: in the first one, we read out values with two significant digits. In the second, we restricted precision to one significant digit. On the x axis, we vary the number of data points read out to the user. On the y axis, we see the percentage of time points and values remembered by users (averaging over all participants).

Long voice descriptions are hard to remember. We observe a sharp drop in recall already starting from more than three elements. This corroborates prior research on short-term memory capabilities [4, 20]. Also, the number of value digits has a significant impact on retention. We obtained near-perfect recall for up to three elements when using one significant digit. Removing time points and focusing on values alone (for equally spaced time intervals) does not seem to significantly increase the acceptable length. Hence, we opted for an output space with varying time interval lengths while restricting the output precision to one significant digit.

7.2 Performance Comparisons

We compare different algorithms and configurations according to two metrics. First, we consider the vocalization error of the generated vocalization. We measure the mean squared error between the actual query result (a time series) and the development described in the vocalization (see Section 2 for a formal definition). Second, we consider the run time required to generate the vocalization.

The algorithm described in Section 3 has two parameters: the number of vocalization candidates generated per iteration (parameter $C.\nu$ in Algorithm 1) and the number of samples taken per iteration (parameter $C.\beta$ in Algorithm 3). Figure 5 illustrates the impact of those parameters using a representative query (a range query retrieving the BitCoin price at the Bitstamp exchange over a twelve month period). In Figure 5(a), we retrieve 10 samples but vary the number of iterations over which those samples are retrieved (from 10 samples in a single iteration to 10 samples over 10 iterations). Retrieving samples sequentially decreases result error since we can integrate information gained from prior iterations to choose sampling points for the following iterations. Performing less iterations decreases however execution time as it reduces non-sampling overheads. Figure 5(b) varies the number of vocalization candidates generated per iteration. Generating more candidates allows to choose more informative sampling points while it increases run time.

We compare proposed algorithm against two baselines. The first baseline ("No Sampling" in the following plots) generates the entire query result first. Then, it uses the algorithm from Section 4 to generate the best possible vocalization. The second baseline ("Random Sampling") gen-

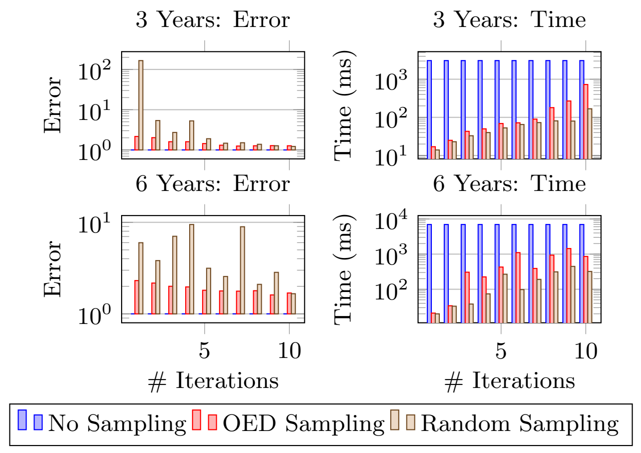


Figure 6: Performance when vocalizing results of range queries on the Bitcoin price data.

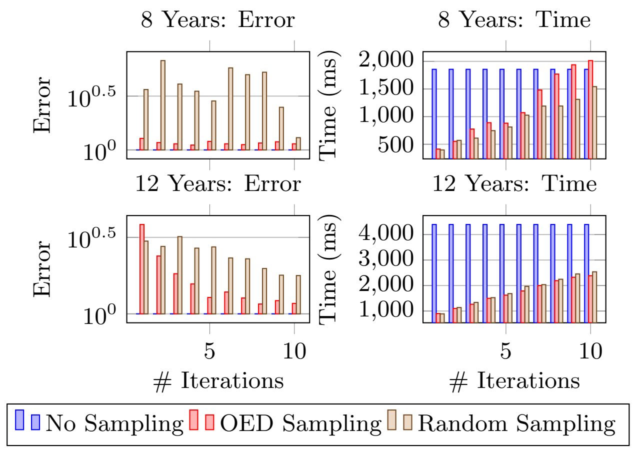


Figure 7: Performance when vocalizing results of group-by/range queries on Chicago crime data.

erates a vocalization based on a query result sample. Sampling points are selected with uniform random distribution from the query time interval. The algorithm that forms the main contribution of this paper ("OED Sampling" in the following plots) compares candidate vocalizations and uses optimal experimental design to select sampling points.

We implemented all algorithms in Java 1.8, the two baselines share the same code with our main algorithm whenever possible. We tune the algorithm from Section 3 as follows: we generate five candidate vocalizations per iteration and collect samples for three time points. We use the patterns described in Table 1 and set the following constraints on output properties. Output length, measured by the number of characters (where we write out numbers as words instead of counting digits), is upper-bounded by 300 characters. This complies for instance with the guidelines of the Google Assistant service for voice output [11]. We use at most three patterns in our voice description and round numbers to the most significant digit (as motivated by the results described in Section 7.1). Wherever the same tuning parameters apply to multiple baselines, we use the same settings.

Next, we report performance results obtained when vocalizing different queries on different data sets. Each data point represents arithmetic averages from 10 runs. We varied the length of the query time range. The start time is chosen with uniform random distribution across the entire time period covered by the corresponding data sets. The experiments were executed on a MacBook Pro with 16 GB of main memory, featuring a 2.8 GHz Intel Core i7 CPU. We used the Postgres 10.1 database management system [23] to store and access data. Each data set had a time-related column on which we created a clustered tree index.

Figure 6 reports performance results when vocalizing the evolution of the Bitcoin course on the Bitstamp exchange. This data set has a size of 2123 MB and covers the time period from 2011 to January 2018. We vocalize the result of range queries of the form SELECT TIME, PRICE FROM [DATA] WHERE [LB] <= TIME <= [UB]. On the x axis, we vary the number of iterations for the two baselines that use sampling ("OED Sampling" and "Random Sampling"). Each of those baselines reads the same number of samples per iteration, i.e. we compare them for the same amount of data read from the database. Note that data points associated with different iteration counts were generated in independent runs (i.e., the output quality may occasionally decrease even though the number of iterations increases). We also report values for the third baseline which is not iterative (hence, the corresponding plot lines are constant). The left side of Figure 6 reports scaled mean squared error of the generated vocalizations. We scale errors to the one obtained by the "No Sampling" baseline. The right side of Figure 6 reports time required for generating vocalizations. Each row corresponds to a different time interval covered by the query (time difference between [UB] and [LB]).

Clearly, generating the entire time series yields optimal voice descriptions. However, the time required to generate the entire query result is prohibitive for exploratory data analysis. It also increases in the query result size (i.e., in the length of the query time interval). On the other side, random sampling and OED sampling are able to perform multiple iterations with sub-second latency. Optimal experimental design adds some overheads in terms of run time (due to the more sophisticated candidate selection method). Those overheads are however moderate and both algorithms seem suitable for iterative analysis in terms of their latency. Using optimal experimental design to select samples improves output quality significantly. The voice descriptions generated by uniform random sampling show a high variance and are occasionally very far from the optimum. Output quality is more stable via optimal experimental design sampling and converges quickly towards the optimum.

We performed additional experiments to corroborate those tendencies. Figure 7 reports performance results for queries on a data set containing crime reports in Chicago from 2001 to 2017 [16]. This data set has a size of 2 GB. We generate crime statistics from data by running queries of the form SELECT TIME, COUNT(*) FROM [DATA] WHERE [LB] <= TIME <= [UB] GROUP BY DAY(TIME). Those queries generate time series containing the number of crimes per day over the time range between [LB] and [UB]. Figure 7 uses the same layout as Figure 6 which was described in detail before. The ranking between the three algorithms, in terms of run time and output quality, is the same as before. However, run time increases for the two sampling-

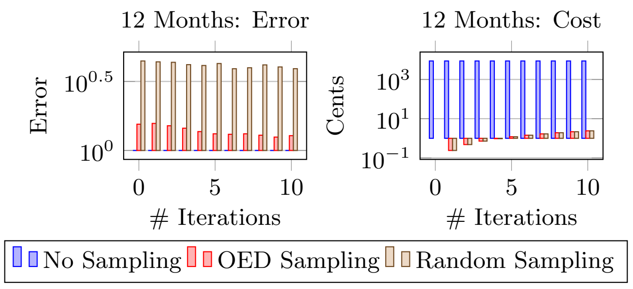


Figure 8: Performance when vocalizing temperature readings retrieved from a remote weather service with invocation cost per call.

Table 2: Comparison against PLA.

Data	**Scaled MSE**		**Time (ms)**	
	This	**PLA**	**This**	**PLA**
S-Uniform	5.04	6.60	789	447
S-OED	2.09	6.17	816	517
Full	1	5.93	5873	123430

based algorithms. This is explained by the properties of the query: retrieving a sample is more expensive as before as it requires counting multiple elements (instead of accessing values directly). The algorithm using OED always generates near-optimal voice descriptions within less than two seconds.

Next, we consider a scenario in which data is retrieved at specific time points via a remote service invocation. We use the historical weather service at `https://darksky.net/dev`. An invocation costs 0.01 cents. Figure 8 compares the baselines in terms of approximation precision and monetary fees. Again, sampling-based vocalization realizes a good tradeoff between monetary fees and approximation precision.

So far, all compared algorithms use the method from Section 4 to generate an optimal vocalization for given input data. The problem of generating an optimal voice description from piecewise linear patterns resembles the problem of finding the best piecewise linear approximation (PLA) for a time series. PLA methods do however not take into account the particularities of our scenario (e.g., constraints on speech length) when choosing an approximation. We combine all three sample generation methods used so far with the pwlf library [14] to generate piecewise linear approximations. We translate those approximations into speech output using the same speech templates as for our approach. The PLA library allows to restrict the number of line segments (which corresponds to speech complexity) but not to restrict speech length. If the PLA approximation with three segments translates into a speech that violates speech length bounds, we reduce the number of segments by one and regenerate the PLA until speech length is acceptable. Table 2 compares run times and scaled mean squared error (MSE), averaging over range queries on the BitCoin data set described before. We average over 50 runs for four queries covering 12 month periods from 2012 to 2016. PLA is very efficient for small input data sets (for large data, its performance suffers since it does not restrict line segment boundaries to easily pronounceable ones). The output quality of PLA is relatively low. Even if PLA has access to the full query result ("Full Data" in Table 2) while our approach has only access to a small result sample ("S-OED"), the PLA error is significantly higher. The PLA problem model does not consider speech length when selecting line segments. Hence, the PLA using three line segments violates speech length constraints in 80% of cases. In those cases, we need to reduce the number of line segments which increases error.

7.3 Crowd User Study

We performed a usability study with crowd workers. Our goal was to find out whether vocalization enables user to perform simple, exploratory analysis of time series data. We compare against two baselines. First, we used a visual interface showing the evolution of a time series over a time range chosen by the user. Second, we tested a sonification [12] interface that translates time series data into non-speech sounds. The latter approach is similar to the one proposed by Ramloll et al. [24] and to approaches used by state-of-the-art sonification tools [28]. We translate data points into notes: the higher the value, the higher the frequency. We generate equally spaced samples for sonification and visualization. For vocalization, we select samples via OED.

We use the price development of the Bitcoin course from 2011 to 2018 as a test case, we used the same data set as in the previous section. We gave crowd workers the following task: propose within a given time range an optimal time to buy and to sell a Bitcoin. More precisely, we asked workers to maximize their monetary gain by the transaction. The Bitcoin price is publicly available via many interfaces. To avoid interference by the use of other interfaces or prior knowledge, we did **not** reveal the nature of the analyzed time series (we used the generic term "price development of a financial asset" consistently in our task description).

We selected this task as a representative of simple exploratory analysis tasks according to the following criteria. First, it is based on a popular data set and loosely connects to real use cases (i.e., analyzing financial time series to maximize gains). Second, it is open-ended and leaves users the choice about which and how many queries to ask. Third, it allows to measure the quality of the worker replies, and thereby, implicitly, the quality of different user interfaces.

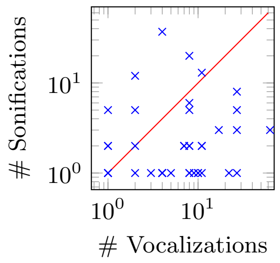
We recruited crowd workers on the Amazon Mechanical Turk platform [1]. We did not restrict access to our tasks in any way (e.g., we did not impose any restrictions on worker acceptance rate) to obtain representative results for average workers. We paid a base reward of 25 cents per task and promised workers an additional reward of 25 cents if their reply was within 50% of the optimal gain achieved by any crowd worker for the same task. We created 120 tasks (half of them for visual interfaces and half of them for audio interfaces), receiving 99 replies over a 24 hours time range.

We made all baselines accessible online, separating the visual interface from the two audio interfaces. For the audio interfaces, users can select themselves between the two output modes: vocalization and sonification. All interfaces are identical except for the output method. Users choose a time range and click on a corresponding button to generate either a visualization, voice description, or sonification describing price evolution in the selected time interval.

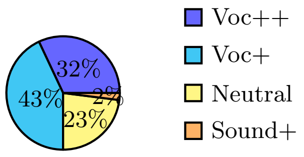
Table 3 compares the two audio interfaces against the visual interface. We report average gain, the average task duration, and the average number of queries asked per user. As expected, the visual interface performs slightly better

Table 3: Comparison of visual versus audio interfaces for analysis of financial time series.

Criterion / Interface	Visual	Audio	Delta
Gain	\$6,002	\$5,051	-16%
Time (s)	278	335	+21%
# Queries	8	12	+50%



(a) Ratio of vocalization to sonification queries run per crowd worker (blue).



(b) Percentage of crowd workers with strong (++) or some preference (+) for vocalization/sonification.

Figure 9: Comparison of alternative audio interfaces for data analysis: vocalization (speech output) vs. sonification (non-speech sounds).

according to all metrics. Our goal is merely to show that the audio interface performs reasonable for this simple task, compared to a visual interface. This seems justified based on the relatively similar results. The highest relative deviation of 50% is obtained for the average number of queries asked by users. This is partially due to the fact that vocalizations are "fleeting" (i.e., users need to query again to refresh their memory) as opposed to a visualization. We found that 19% of queries are duplicates of prior queries by the same user, 12% are immediate repetitions (i.e., no other query between two duplicates). We derived an upper bound of 7% of queries that are immediate repetitions within short-term memory duration (assuming a high duration of 30 seconds). Hence, we are typically successful in transmitting data into the user's short-term memory.

Our goal is not to replace visual interfaces as the primary means of data analysis. Instead, we want to make the best out of situations where visual interfaces are not available. Figure 9 compares the two audio interfaces. Figure 9(a) show the number of vocalization and sonification queries executed by each crowd worker (represented by a blue cross). Clearly, most workers tend to generate more voice descriptions than sonifications. Out of 30 workers who had tried both, sonification and vocalization, at least once, 20 generated finally more voice descriptions than sonifications, three generated the same number, and only seven generated more sonifications. While the number of executed queries is only a proxy for preferences, Figure 9(b) reports user preferences directly. Three quarters of workers report preferences for vocalization over sonification. One third of them express strong preferences, describing vocalization as "much clearer" or "much easier to understand". 23% did not answer or indicated no preferences. Only 2% of workers expressed some preference for sonification.

8. RELATED WORK

We optimally vocalize small relations in prior work [31]. Our prior work differs from the current one as follows. First, our algorithm interleaves query evaluation and vocalization (i.e., the input is a query and a database). Our prior work presents a pure vocalization strategy (i.e., the input is a query result). Hence, our prior work requires generating the entire query result which is impractical for large data sets. Second, we minimize vocalization error for bounded speech length and complexity. Our prior work minimizes speech length for bounded error. Third, our current work focuses on time series where high-level patterns can summarize many data points. Our prior work focuses on general relational data. It outputs at least one key value for each output tuple and is therefore impractical to large result sets.

The problem of translating a time series into a textual descriptions has been studied previously [3, 9, 13, 10, 30, 33, 32]. Our work is particular since we focus on exploratory analysis of large time series. We focus on scenarios where the computational cost of merely accessing (or generating) the entire time series is already prohibitive for a responsive interface. This motivates our iterative sampling strategy. Also, we focus specifically on voice output, imposing tight constraints on output length and complexity. This motivates an approach based on optimization to transmit the maximal amount of information under all applicable constraints.

Our work is similar in intent to prior work in the database community on visualizing time series data [2, 21, 25, 34]. Specifically, we propose an approach for interleaved data processing and output generation which exploits particularities of the output format to minimize processing overheads. This connects our work to prior work aimed at minimizing processing overheads when generating visualizations [15, 18]. However, our work is complementary as we present query results via voice output as opposed to visualizations. Our work is also complementary to prior work from the area of sonification and auditory display [12, 24] which uses primarily non-speech sounds to transmit data.

Our work relates to prior work on generating approximate representations of data such as histograms [8]. Size and approximation error have been used as metrics for histogram construction as well. However, we consider additionally the length of the verbal description which influences our pruning function and the interval bounds we consider. Sampling has been used for histogram construction [6]. The criterion by which we select samples is however very specific (i.e., to narrow down the choice between voice descriptions). In a similar way, our work differs from prior work on approximating time series via linear segments [7].

9. CONCLUSION

We presented an algorithm that summarizes query results via concise voice descriptions. It avoids full query evaluation and generates only result fragments that are helpful to determine optimal voice output. We have shown that users can perform simple data analysis tasks via voice interfaces, producing analysis results of slightly lower but still comparable quality to users accessing visual interfaces. Further, we have shown that users tend to prefer voice data descriptions over non-speech sounds. We focused on time series while we plan to extend our approach to different data types in the future. We also plan to consider new query types.

10. REFERENCES

[1] Amazon. Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome.

[2] N. Begum and E. Keogh. Rare time series motif discovery from unbounded streams. *PVLDB*, 8(2):149–160, 2014.

[3] S. Boyd. TREND: a system for generating intelligent descriptions of time-series data. In *IEEE International Conference on Intelligent Processing Systems*, 1998.

[4] D. Caplan, V. Burnham, and G. S. Waters. Verbal working memory and sentence comprehension. *Behavioral and Brain Sciences*, 22(1):77–94, 1999.

[5] D. Chandarana, V. Shah, A. Kumar, and L. Saul. SpeakQL: towards speech-driven multi-modal querying. In *HILDA*, pages 1–6, 2017.

[6] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction. *SIGMOD Record*, 27(2):436–447, 1998.

[7] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. X. Yu. Indexable PLA for efficient similarity search. In *PVLDB*, pages 435–446, 2007.

[8] G. Cormode. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2011.

[9] D. Gkatzia, H. Hastie, and O. Lemon. Finding middle ground? Multi-objective natural language generation from time-series data. In *EACL*, pages 210–214, 2014.

[10] E. Goldberg, N. Driedger, and R. Kittredge. Using natural-language processing to produce weather forecasts. *IEEE Expert-Intelligent Systems and their Applications*, 9(2):45–53, 1994.

[11] Google. Google Assistant SDK. https://developers.google.com/assistant/sdk/overview.

[12] T. Hermann, A. Hunt, and J. G. Neuhoff. *The Sonification Handbook*. 2011.

[13] J. Hunter, Y. Freer, A. Gatt, E. Reiter, S. Sripada, and C. Sykes. Automatic generation of natural language nursing shift summaries in neonatal intensive care: BT-Nurse. *Artificial Intelligence in Medicine*, 56(3):157–172, 2012.

[14] C. Jekel. PWLF Library. https://jekel.me/2017/Fit-a-piecewise-linear-function-to-data/.

[15] U. Jugel, Z. Jerzak, and G. Hackenbroich. M4 : A visualization-oriented time series data aggregation. *PVLDB*, 7(10):797–808, 2014.

[16] Kaggle. Chicago Crime Data. https://www.kaggle.com/currie32/crimes-in-chicago.

[17] H. V. Kgadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *PVLDB*, pages 275 – 286, 1998.

[18] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5):521–532, 2015.

[19] G. Lyons, V. Tran, C. Binnig, U. Cetintemel, and T. Kraska. Making the case for Query-by-Voice with EchoQuery. In *SIGMOD*, pages 2129–2132, 2016.

[20] G. A. Miller. The magical number 7, plus or minus 2 - some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[21] R. Neamtu, R. Ahsan, E. Rundensteiner, and G. Sarkozy. Interactive time series exploration powered by the marriage of similarity distances. *PVLDB*, 10(3):169–180, 2016.

[22] G. Nemhauser and L. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[23] PostgreSQL Global Development Group. PostgreSQL. https://www.postgresql.org/, 2017.

[24] R. Ramloll, W. Yu, and B. Riedel. Using non-speech sounds to improve access to 2D tabular numerical information for visually impaired users. In *Conference of the British HCI Group*, pages 515–529, 2001.

[25] K. Rong and P. Bailis. ASAP: prioritizing attention via time series smoothing. *PVLDB*, 10(11):1358–1369, 2017.

[26] P. Roth. *Design of experiments for discrimination among rival models*. PhD thesis, 1967.

[27] E. G. Ryan, C. C. Drovandi, J. M. Mcgree, and A. N. Pettitt. A review of modern computational algorithms for Bayesian optimal design. *International Statistical Review*, 84(1):128–154, 2016.

[28] SAS. http://support.sas.com/software/products/graphics-accelerator/.

[29] B. Settles. Active learning literature survey. Technical report, 2010.

[30] S. G. Sripada, E. Reiter, J. Hunter, J. Yu, and I. P. Davy. Modelling the task of summarising time series data using KA techniques. In *Applications and Innovations in Intelligent Systems*, pages 183–196, 2002.

[31] I. Trummer, J. Zhu, and M. Bryan. Data vocalization: optimizing voice output of relational data. *PVLDB*, 10(11):1574–1585, 2017.

[32] J. Yu, E. Reiter, J. Hunter, and C. Mellish. Choosing the content of textual summaries of large time-series data sets. *Natural Language Engineering*, 13(01):25–49, 2006.

[33] J. Yu, E. Reiter, J. Hunter, and S. Sripada. SumTime-Turbine: A knowledge-based system to communicate gas turbine time-series data. In *Developments in Applied Artificial Intelligence*, pages 199–210, 2003.

[34] K. Zoumpatianos, S. Idreos, and T. Palpanas. RINSE: Interactive data series exploration with ADS +. *PVLDB*, 8(12):1912–1915, 2015.