Data Vocalization: Optimizing Voice Output of Relational Data

Immanuel Trummer, Jiancheng Zhu, Mark Bryan
{itrummer,jz448,mab539}@cornell.edu
Cornell University

ABSTRACT

Research on data visualization aims at finding the best way to present data via visual interfaces. We introduce the complementary problem of "data vocalization". Our goal is to present relational data in the most efficient way via voice output. This problem setting is motivated by emerging tools and devices (e.g., Google Home, Amazon Echo, Apple's Siri, or voice-based SQL interfaces) that communicate data primarily via audio output to their users.

We treat voice output generation as an optimization problem. The goal is to minimize speaking time while transmitting an approximation of a relational table to the user. We consider constraints on the precision of the transmitted data as well as on the cognitive load placed on the listener. We formalize voice output optimization and show that it is NP-hard. We present three approaches to solve that problem. First, we show how the problem can be translated into an integer linear program which enables us to apply corresponding solvers. Second, we present a two-phase approach that forms groups of similar rows in a pre-processing step, using a variant of the apriori algorithm. Then, we select an optimal combination of groups to generate a speech. Finally, we present a greedy algorithm that runs in polynomial time. Under simplifying assumptions, we prove that it generates near-optimal output by leveraging the sub-modularity property of our cost function. We compare our algorithms experimentally and analyze their complexity.

1. INTRODUCTION

Prior work studying the optimal representation of relational data to users is mostly targeted at visual interfaces. There are however many emerging scenarios in which data is presented via voice output instead. Cell phones offer nowadays audio interfaces as an alternative to more traditional interaction modes. Devices such as Google Home or Amazon Echo use voice output and input as primary means of communication. Often, such devices need to transmit relational data to their users, be it structured Web search results, information on nearby restaurants, or result relations answering SQL queries (a voice-based SQL interface based on Amazon Echo was recently proposed [13]). This motivates the problem of optimal "data vocalization" (complementary to the problem of optimal data visualization) which is the focus of this paper.

As pointed out in prior work [18], voice output is subject to specific constraints, compared to written text. Voice output has to be of a simple structure in order to restrict cognitive load on the listener. We need to respect limitations of the user's short term memory [15] as users cannot simply re-read preceding passages. For instance, we cannot generate speeches that require users to memorize many facts mentioned initially in order to understand the rest of the speech. Most importantly, we need to make voice output as concise as possible to avoid exceeding the user's attention span. While users can themselves quickly identify relevant parts in a plot or in written text (via skimming), they have to trust the computer to select only the most relevant information for voice output.

We formalize voice output generation as an optimization problem. Our goal is to minimize speaking time under constraints that reflect the particularities of voice output. The search space is formed by speeches of a simple structure that is easy to understand for users, even after listening to output only once. We restrict the search space via additional constraints, limiting the number of facts that users need to keep in mind at any point in order to understand the generated speech. Compared to naive output of relational data (i.e., reading out one row after the other [13]), we reduce speaking time by summarizing rows with equal or similar values in certain columns.

The generated speeches may contain passages, called context in the following, that assign some relation columns to values or value domains. A context creates a scope, i.e. a part of the speech within which those value assignments are valid. For rows that are read out within a scope, we can omit reading out values for attributes that have been fixed by the context.

If we assign an attribute to a value domain instead of a single value then we loose information: listeners will be unable to map the following rows to one specific value out of the value domain. The larger the value domain assigned by the context, the less precise is the information transmitted via voice output. Often, there is a tradeoff between output precision and speaking time: if we are willing to accept less precise output then speaking time can be reduced. We allow to bound the precision of transmitted information by

Proceedings of the VLDB Endowment, Vol. 10, No. 11

constraints that refer to the size of value domains assigned by a context. We show in Section 2 that finding optimal voice output plans is an NP-hard problem. We introduce our running example to illustrate problem and search space.

EXAMPLE 1. *We read out information on nearby restaurants, stored in the following relation, to a mobile user.*

Restaurant	***Rating***	***Food Category***
Upstate	*4.75*	*Traditional American cuisine*
Thai Castle	*4.3*	*Thai cuisine*
John's	*4.7*	*Traditional American cuisine*
Paris	*4.3*	*French cuisine*

A naive output plan reads out one row after the other which results in the following output: "Restaurant Upstate, four point seven five stars average rating, food category traditional American cuisine. Restaurant Thai Castle, four point three stars average rating, food category Thai cuisine. Restaurant John's, four point seven stars average rating, food category traditional American cuisine. Restaurant Paris, four point three stars average rating, food category French cuisine." We reduce redundancy by summarizing rows with equal values in a context, e.g. "Entries for food category traditional American cuisine: Restaurant Upstate, four point seven five stars average rating. Restaurant John's, four point seven stars average rating. Entries for four point three stars average rating: Restaurant Thai Castle, food category Thai cuisine. Restaurant Paris, food category French cuisine." This example uses two contexts. Each context assigns one attribute to a value that applies to all rows within the corresponding scope. If we accept approximate output (e.g., we consider contexts that assign attributes to intervals such that the upper bound is higher than the lower bound by no more than 25%) then we generate even shorter output such as "Entries for food category traditional American cuisine and four to five stars average rating: Restaurant Upstate. Restaurant John's. Entries for four point three stars average rating: Restaurant Thai Castle, food category Thai cuisine. Restaurant Paris, food category French cuisine.".

We present several approaches to solve that problem. First, we show in Section 3 how to translate voice output optimization into an integer program. After the transformation, solvers such as CPLEX or Gurobi can be used to solve the resulting programs. While this approach guarantees to find an optimal solution, it turns out that optimization time is often prohibitive for large problem instances. This motivates our second approach, presented in Section 4, which uses integer programming as well but within a restricted search space. We reduce the search space via a pre-processing step in which we select promising context candidates (i.e., sets of assignments from attributes to value domains). Our goal is to generate context candidates that can be used to output a large number of rows. We generate them using a modification of the apriori algorithm: as context candidates become valuable if they cover frequent attribute-value pairs, the problem of generating optimal candidates is similar to the problem of mining frequent item sets. To deal with even larger problem instances, we finally present a polynomial time greedy algorithm in Section 5. Based on submodularity properties of our cost function, we lower-bound its output quality under simplifying assumptions.

```
<Output>::=<Row>* <Scope>*
<Scope>::=<Context> <Row>+
<Context>::="Entries for " <DomAsgs> ": "
<DomAsgs>::=<DomAsg>|(<DomAsg>", ")+ "and "<DomAsg>
<DomAsg>::=<CatDomAsg>|<NumDomAsg>
<CatDomAsg>::=<Value> " " <Attribute>|
        (<Value>", ")+ "or "<Value>" "<Attribute>
<NumDomAsg>::=<Num> " " <Attribute>|
        "from " <Num> " to " <Num> <Attribute>
<Row>::=<ValAsg>|(<ValAsg>", ")+ "and "<ValAsg>"."
```

Figure 1: Speech output structure in EBNF.

In summary, our original scientific contributions in this paper are the following:

- We introduce and analyze the problem of voice output optimization for relational data.
- We present several exhaustive and non-exhaustive algorithms for that problem.
- We prove bounds on the output quality of those algorithms and analyze their complexity.
- We experimentally compare our algorithms based on several realistic use cases. We also verify, via a small user study, that the output generated by our algorithms is comprehensible.

The remainder of this paper is organized as follows. We formalize the voice output optimization problem in Section 2 and show that it is NP-hard. Then, we show how to translate the problem into an integer program in Section 3. In Section 4, we present a two-phase approach to voice output optimization that selects promising context candidates in a pre-processing step before using integer programming. In Section 5, we present a greedy algorithm and prove lower bounds on its output quality. We analyze the complexity of all presented algorithms in Section 6 and evaluate them experimentally in Section 7. Finally, we compare to related work in Section 8.

2. PROBLEM MODEL

We treat voice output generation as an optimization problem. For a given relation to output[1], we seek a *Voice Output Plan* that minimizes speaking time under certain constraints. Those constraints refer to the precision of the transmitted information and to the cognitive load placed on the listener.

We consider output plans of a simple structure summarized in Figure 1. Each row in the relation to output is either read out separately (by reading all associated attribute - value pairs) or within a *Scope*. At the beginning of a scope, we provide *Context* by assigning a subset of attributes to value domains. The rows within the scope inherit those assignments and we omit reading out values for the context attributes for those rows.

A context assigns categorical attributes to value sets and numerical attributes to intervals. For value sets, we upper-bound the number of values (a measure of imprecision) by a

[1]Note that we output only a single relation (which may however result from a query over multiple relations).

Table 1: Overview of Introduced Symbols.

Symbol	Semantics
m_S	Maximal context size
m_W	Maximal width for intervals
m_C	Maximal domain cardinality
$T(.)$	Speaking time of element
MATCHES	Row matches context?

parameter m_C. For intervals, we upper-bound the relative width (i.e., the factor separating upper and lower bound) by parameter m_W. Interpreting rows in a scope requires to hold all domain assignments in short-term memory. Parameter m_S models the number of slots in short term memory [15] and restricts therefore the context size (i.e., the number of assignments).

EXAMPLE 2. *We illustrate the aforementioned concepts in a sentence from Example 1:* [[*Entries for* [*food category traditional American cuisine*]$_{DomAsg}$ *and* [*four to five stars average rating*]$_{DomAsg}$ *:*]$_{Context}$ [*Restaurant Upstate.*]$_{Row}$ [*Restaurant John's.*]$_{Row}$]$_{Scope}$

We formalize the voice output optimization problem. A relation R to output is a set of rows. Each row $r \in R$ is a set of assignments from attributes to values. A context c is a set of assignments from attributes to value domains. A context is valid if the sizes of all of its value domains are acceptable as defined by parameters m_C and m_W and if its size, $|c|$, is not above the threshold m_S. A row r matches a context c, denoted by the predicate MATCHES(c, r) in the following, if the row assigns each attribute to a value that lies within the value domain assigned to that attribute by c (if any).

Let $T(r)$ be the time for reading out a row without context. We denote by $T(c, r) \leq T(r)$ the time for reading out value assignments only for the attributes that have not been fixed by context c. $T(c)$ is the time for reading out the context itself. For a fixed plan p, we denote by C the set of contexts it uses, by $R_W \subseteq R$ the rows that are read out without context, and by $R_C = R \setminus R_W$ the other rows. Further, we denote for any row $r \in R_C$ by $c_r \in C$ the context that plan p assigns to it. Then the speaking time for the plan, $T(p)$, is given by the formula $(\sum_{c \in C} T(c)) + (\sum_{r \in R_W} T(r)) + (\sum_{r \in R_C} T(c_r, r))$. Given a relation R, the goal in *Voice Output Optimization* is to find an output plan p for R whose duration $T(p)$ is minimal. Table 1 summarizes the most important symbols.

THEOREM 1. *Voice output optimization is NP-hard.*

PROOF. We reduce vertex cover to voice output optimization in polynomial time. We create a relation that contains one row for each edge in the vertex cover instance, and one categorical column for each vertex. For a given column and row, we store a distinguished value α in the cell if the corresponding vertex is incident to the corresponding edge. All other values in the relation are mutually different. We assume that speaking time is directly proportional to the number of values that are read out (i.e., all values have unit length and the context template text is empty). We set m_S and m_C both to one (single assignment contexts and single value domains). Denoting by n the number of vertices and by m the number of edges, we find a voice output plan with length $(n-1) \cdot m + k$ iff we find a cover with k vertices. This can be seen as follows. For a given voice output plan, we select all vertices associated with columns appearing in any context (thereby covering all edges associated with the rows that are output within those contexts). For rows that are output without context, we select an arbitrary vertex to cover the associated edge. The resulting cover has k vertices. On the other hand, for a given vertex cover, we create a voice output plan with one context for each selected vertex. We output each row within an arbitrary, matching context. The resulting voice output plan has length $(n-1) \cdot m + k$. □

3. INTEGER PROGRAMMING

We show how to transform an instance of voice output optimization into an integer linear program. Integer programs consist of a set of integer variables, a set of linear constraints, and a linear function to minimize or maximize. Mature integer programming solvers such as CPLEX or Gurobi can find guaranteed optimal solutions for such programs using exponential time algorithms. We show how to represent valid output plans in Section 3.1. In Section 3.2, we show how to calculate speaking time for a given plan. Tables 2 to 4 summarize the transformation.

3.1 Representing Output Plans

A context is only helpful if it frees us from reading out similar attribute values repeatedly for different rows. Hence, there is an optimal plan that outputs at least two rows within each context. Given a relation with n rows, we therefore integrate $c_{max} = \lfloor n/2 \rfloor$ context slots into the corresponding ILP. Each slot can be used to model one context in an output plan. For each slot, we introduce a binary variable $g(c)$, indicating whether slot c is actually used (we set $g(c)$ to one in that case).

For each context slot c and attribute a, we introduce a binary variable $f(c, a)$ indicating whether the context assigns a domain to the attribute (then $f(c, a) = 1$). Categorical attributes are assigned to value sets by a context. We introduce binary variables of the form $d(c, a, v)$ that are set to one iff value v is in the domain assigned to categorical attribute a by context c. Numerical attributes are assigned to intervals (or single values as a special case). We introduce binary variables of the form $l(c, a, v)$ that are set to one iff context c assigns value v to numerical attribute a as a lower bound. Similarly, we describe upper bounds by binary variables of the form $u(c, a, v)$. For the latter two families of variables, we consider values that appear for the numerical attribute in the input relation, as well as close-by values that are fast to read since they only have one significant digit.

Each context is subject to constraints that relate to the precision of transmitted information and to the cognitive load placed on the listeners. We restrict the size of a context c by constraints of the form $\sum_a f(c, a) \leq m_S$. For each categorical attribute a, we restrict the cardinality of the domain assigned by context c via constraints of the form $\sum_v d(c, a, v) \leq m_C$ (summing over all values in the attribute value domain). For each numerical attribute a, we restrict the width of the interval assigned by context c via constraints of the form $\sum_v v \cdot u(c, a, v) \leq \sum_v v \cdot l(c, a, v) \cdot m_W$. For each numerical attribute a, we ensure that the lower

Table 2: Summary of ILP Variables.

Variable	Semantics
$g(c)$	Is context number c generated?
$f(c,a)$	Context c fixes attribute a?
$l(c,a,v)$	Context c sets v as lower bound for a?
$u(c,a,v)$	Context c sets v as upper bound for a?
$e(c,a,v)$	Lower and upper bound for a in c equal v?
$d(c,a,v)$	Context c includes v in domain of a?
$w(c,r)$	Row r read within context c?
$s(c,r,a)$	Save time for reading a in r due to c?

Table 3: Summary of ILP Constraints.

Constraint	Semantics
$\sum_c w(c,r) \leq 1$	Row in at most one context
$\sum_a f(c,a) \leq m_S$	Limit on context size
$\sum_v d(c,a,v) \leq m_C$	Limit on domain size
$\sum_v v \cdot u(c,a,v) \leq m_W \cdot \sum_v v \cdot l(c,a,v)$	Limit on interval width
$\sum_v l(c,a,v) = f(c,a)$	Context fixes lower bound
$\sum_v v \cdot l(c,a,v) \leq \sum_v v \cdot u(c,a,v)$	Lower bound below upper
$l(c,a,v)+w(c,r)+f(c,a) \leq 2$ $u(c,a,v)+w(c,r)+f(c,a) \leq 2$ $w(c,r)+f(c,a)-d(c,a,v_r) \leq 1$	Row must match context
$s(c,r,a) \leq w(c,r)$	Need context for savings
$s(c,r,a) \leq f(c,a)$	Must fix attributes to save
$g(c) \geq w(c,r)$	Generate used contexts
$e(c,a,v) \leq l(c,a,v)$ $e(c,a,v) \leq u(c,a,v)$	Bounds equal if same value

bound assigned by context c (if any) is not above the upper bound via constraints of the form $\sum_v v \cdot l(c,a,v) \leq \sum_v v \cdot u(c,a,v)$. Finally, we add constraints of the form $\sum_v l(c,a,v) = f(c,a)$ for each numerical attribute a to ensure that we pick a lower bound whenever context c fixes the attribute to a domain (and analogue constraints for upper bounds).

We still need to model the assignment of rows to specific context slots. We introduce binary variables of the form $w(c,r)$ for each row r and context c that are set to one iff the row is read out within the corresponding context. We introduce constraints of the form $g(c) \geq w(c,r)$ for each context slot c and row r to ensure that each context used for assignments is generated. Rows can only be assigned to a context mapping attributes to domains that contain the values found in the row. For each row r with value v_r in numerical attribute a, we introduce constraints of the form $l(c,a,v)+w(c,r)+f(c,a) \leq 2$ for each context c and $v > v_r$ to ensure that one of the following holds: either the row is not read out within context c (i.e., $w(c,r)=0$), or the context does not assign attribute a to a domain (i.e., $f(c,a)=0$), or the lower bound is not above v_r (i.e., $l(c,a,v)=0$ for $v > v_r$). Similarly, we introduce constraints of the form $u(c,a,v)+w(c,r)+f(c,a) \leq 2$ to account for upper bounds. For each row r with value v_r for a categorical attribute a, we introduce constraints of the form $(1-d(c,a,v_r))+w(c,r)+f(c,a) \leq 2$ for each context c. This ensures that rows are only assigned to contexts with matching value assignments for categorical attributes.

Finally, we introduce constraints of the form $\sum_c w(c,r) \leq 1$ for each row r to ensure that the row is assigned to at most one context. This is necessary since the objective function, presented in the next subsection, calculates time savings by summing over all assignments for a given row.

Table 4: Summary of ILP Cost Terms.

Term	Semantics
$-\sum_{c,r,a} s(c,r,a) \cdot (T(a)+T(v_r^a))$	Savings due to context
$\sum_c g(c) \cdot T(\text{“Entries for : ”})$	Context boilerplate time
$\sum_{c,a} f(c,a) \cdot T(a)$	Attribute names in context
$\sum_{c,a,v} d(c,a,v) \cdot T(v)$	Categories in context
$\sum_{c,a,v} T(v) \cdot l(c,a,v)$	Lower bounds in context
$\sum_{c,a,v}(T(v)+T(\text{“ from to ”})) \cdot (u(c,a,v)-e(c,a,v))$	Upper bounds in context

3.2 Evaluating Output Plans

We show how to formulate our objective function, representing speaking time. Instead of optimizing absolute speaking time, we equivalently optimize the time difference to a naive plan (reading out one row after the other without using any context). This time difference is given by the overhead due to reading out context text minus the time savings by omitting attributes that are fixed in a context.

For each context, we read out boilerplate text (e.g., “Entries for : ”) and value domain assignments. Time overhead for boilerplate text can be captured by the term $\sum_c g(c) \cdot T(\text{“Entries for :”})$. The term $\sum_{c,a} f(c,a) \cdot T(a)$ accounts for time required to read out attribute names inside the domain assignments. The time required for reading out the values in those assignments is captured by the term $\sum_{c,a,v} d(c,a,v) \cdot T(v)$ for categorical attributes. For numerical attributes, we capture time required for reading out the lower bound by the term $\sum_{c,a,v} l(c,a,v) \cdot T(v)$. Upper bounds only need to be read out, together with additional boilerplate text, if the upper bound is different from the lower bound. We introduce a family of binary variables of the form $e(c,a,v)$ indicating whether lower and upper bound assigned to numerical attribute a by context c are both equal to value v. Introducing constraints of the form $e(c,a,v) \leq l(c,a,v)$ and $e(c,a,v) \leq u(c,a,v)$ forces those variables to zero if the associated condition is not satisfied (forcing them to one is not required as it minimizes the following cost term). The time required for reading out upper bounds (if different from the corresponding lower bound) is captured by the term $\sum_{c,a,v}(T(v)+T(\text{“ from to ”})) \cdot (u(c,a,v)-e(c,a,v))$. Note that our cost function is slightly simplified since we do not take into account the effect of connectors (e.g., “and” and “or”).

Next, we model time savings by the use of context. We introduce binary variables of the form $s(c,r,a)$ indicating whether we save time by omitting attribute a when reading out row r within context c. Clearly, it is $s(c,r,a) \leq f(c,a)$ and $s(c,r,a) \leq w(c,r)$. Denoting by v_r^a the value of row r

for attribute a, the term $-\sum_{c,r,a}(s(c,r,a)\cdot(T(a)+T(v_r^a)))$ captures time savings due to the use of context. The overall optimization goal is to minimize the sum of all terms described in this subsection.

EXAMPLE 3. *We sketch the transformation for Example 1. We have four tuples (i.e., four restaurants) and need therefore at most two contexts. Variables $g(1)$ and $g(2)$ indicate whether the two context slots are actually used. We introduce eight variables $w(c,r)$ representing assignments between a tuple and a context. We have two non-key attributes (the food category, and the average rating) and introduce four variables of the form $f(c,a)$, indicating for each attribute whether it is assigned to a domain by the corresponding context. As the rating is a numerical attribute, we introduce variables of the form $l(c,$"rating"$,v)$, $u(c,$"rating"$,v)$, and $e(c,$"rating"$,v)$ for each context, describing lower and upper bounds of the domain assigned by the context (if any). We introduce those variables for each rating value v that appears in the corresponding column and for several rounded values in between. The food category is a categorical attribute and we introduce variables of the form $d(c,$"food category"$,v)$ for both contexts and each food category value v that is mentioned in the column. We enforce consistent variable assignments (e.g., tuples are only assigned to matching contexts) and user preferences (e.g., context size is below threshold) by constraints. Our cost function sums overhead for reading context and rows, taking into account that attributes fixed in a context are never repeated in the same scope.*

4. TWO-PHASE ALGORITHM

We present a two-phase approach to voice output optimization. In the first phase, we generate a set of promising context candidates for a given relation. In the second phase, we assign rows to context candidates in an optimal fashion (thereby deciding which candidates are generated). The algorithm can be tuned by a parameter that decides how many context candidates get generated, thereby trading output quality for optimization time. We describe the first phase in Section 4.1 and the second phase in Section 4.2.

4.1 Generating Promising Contexts

Algorithm 1 generates a set of contexts that are potentially useful. Only the resulting contexts are considered in the row assignment phase described in the next subsection. Function ContextCandidates takes as input a relation R to output and a parameter k limiting the number of generated context candidates. It returns a set of context candidates that are potentially useful to reduce output time. Each context is modeled as a set of assignments from attributes to value domains. Function DomainAssignments returns for a given relation R the set of all assignments for all attributes (i.e., all admissible intervals for numerical attributes and all admissible value sets for categorical attributes).

Algorithm 1 is inspired by the apriori algorithm for mining frequent item sets (and association rules) [1]. Contexts, modeled as assignment sets, take the place of item sets.

The apriori algorithm is based on the apriori rule, specifying that item sets with infrequent subsets cannot be frequent. This enables the algorithm to avoid generating many infrequent item sets. We need to find a similar rule for voice output optimization. We base this rule on the following fact: a context can only be useful if the time required for reading

```
1: // Generates contexts that could shorten readout of R.
2: // Keeps at most k contexts per context size.
3: function ContextCandidates(R, k)
4:     // Generate contexts with single assignment
5:     A ←DomainAssignments(R)
6:     // Initialize context candidates
7:     C_0 ← {∅}
8:     // Iterate over number of assignments
9:     for i ← 1, ..., m_S do
10:        // Generate new contexts
11:        C_i ← {c ∪ {a} | c ∈ C_{i−1}, a ∈ A \ c}
12:        // Prune useless contexts
13:        C_i ←PruneUseless(C_i, R)
14:        // Select diverse subset
15:        C_i ←MaxRowCover(C_i, R, k)
16:    end for
17:    // Return potentially useful contexts
18:    return ∪_{1≤i≤m_S} C_i
19: end function
```

Algorithm 1: Generate diverse set of context candidates that are potentially useful for speech output.

out the context is below the potential time savings when reading out rows after the context. We calculate potential time savings of a context as follows: we sum up the time difference between reading out an entire row (with all attributes and values) and reading out the key attribute (with unique value) alone over all rows that match the domain assignments of the context. Clearly, a context is useless if the potential time savings do not match the time required for reading the context, i.e., if the following formula is satisfied:

$$T(c) \geq \sum_{r \in R | \text{Matches}(c,r)} (T(r) - T(r.key)). \qquad (1)$$

Furthermore, we show in the following that each possible specialization (i.e., we specialize a context by adding more value assignments) of a useless context is useless as well.

LEMMA 1. *A specialization of a useless context is useless.*

PROOF. Specializing a context, i.e. adding more value assignments, increases the time required for reading out the context. On the other side, specializing the context can only reduce the number of rows matching its assignments. Specializing the context can of course reduce the time required for reading out a row after that context. However, when calculating potential time savings, we already assume the maximal time savings per row that can be achieved by any specialization of a context. Specializing a context can therefore only reduce the potential time savings. Hence, if the reading time of a context exceeds the potential time savings, then this applies for each specialization as well. □

Algorithm 1 iteratively generates candidate sets of increasing size, up to the maximal context size m_S. In each iteration, we extend the candidates generated in the last iteration by one assignment. Function PruneUseless exploits (1) to identify and discard useless context candidates. Still, the set of potentially useful context candidates can be large, leading to prohibitive context generation time. Therefore, Algorithm 1 offers the possibility to upper-bound the number of context candidates kept after each iteration via parameter k. Setting k to infinity generates all potentially

useful context candidates (which allows us to find optimal output plans). Setting k to a finite value ensures that at most that many context candidates are kept after each iteration (i.e., $|C_i| \leq k$).

Function MaxRowCover returns a set of context candidates of cardinality at most k. A context tends to be more useful, the more rows it can cover (i.e., the more rows match the context). Ranking context candidates by the number of covered rows and selecting the top-k candidates leads however to the following problem: the top-k context candidates might be very similar and cover essentially the same set of rows, thereby leaving many rows uncovered. Our goal is to select a rather diverse set of context candidates that, taken together, cover as many rows as possible. We use a simple greedy algorithm to select a fixed number of context candidates: at each step, we select the context candidate that covers the highest number of rows among the rows not yet covered by the previously selected candidates.

This corresponds to a classical greedy algorithm for submodular maximization [17]. Also, row cover is a submodular function (i.e., adding more and more context candidates has less and less effect on the total number of covered rows) as shown by the following lemma.

Lemma 2. *Row cover is submodular.*

Proof. Let $\mathcal{U}(C)$ be the number of rows in R matching a context in C. We need to show $\mathcal{U}(C \cup \{c\}) - \mathcal{U}(C) \geq \mathcal{U}(C' \cup \{c\}) - \mathcal{U}(C')$ for $C' \supseteq C$. The set of new rows covered by adding a new context c into C is the intersection of all rows covered by c but not by any context in C. The latter set can only shrink when replacing C by C'. Hence, the number of newly covered rows can only shrink as well. □

Hence, we select a near-optimal context set when implementing Function MaxRowCover by the greedy algorithm.

Theorem 2. *Function* MaxRowCover *selects contexts whose row cover is within factor* $1 - 1/e$ *of the optimum.*

Proof. Row cover is non-negative, monotone, and submodular. It satisfies the condition for the near-optimality guarantees given by Nemhauser and Wolsey [17]. □

4.2 Mapping Rows to Contexts

The algorithm from the last subsection generates a set of potentially useful context candidates. We use integer programming to map rows to context candidates, thereby implicitly selecting which of the context candidates are actually used. The corresponding integer program has some similarity with the one presented in Section 3. It is however much simpler as we delegate many decisions to the pre-processing step. In the following, we denote by C the set of potentially useful context candidates that was generated by the algorithm from the last subsection. Additionally, C contains a special context, the empty context, which does not include any assignments. Mapping a row to the empty context means that the row is output without any context (i.e., we read each attribute in the row). Introducing the empty context simplifies the following expressions as we can assume that each row is mapped to exactly one context (while not neglecting the possibility to renounce using any context).

We introduce again a set of binary variables $w(c, r)$ indicating whether row r is output within context c (in that case, we set $w(c, r)$ to one). Each row is mapped to one context which translates into constraints of the form $\sum_{c \in C} w(c, r) = 1$. We introduce another set of binary variables $g(c)$ indicating whether context candidate $c \in C$ is generated. A context needs to be generated before it can be used to output rows. This translates into constraints of the form $g(c) \geq w(c, r)$. We can express speaking time using the constants $T(c)$, indicating speaking time for context c, and $T(c, r)$, expressing the time required to output row r within context c (with $T(c, r) = T(r)$ for the empty context c). Speaking time is now the sum of the time overhead due to generating context, $\sum_c g(c) \cdot T(c)$, and the time overhead of reading out rows within their respective context, $\sum_{r,c}(w(c, r) \cdot T(c, r))$. The optimization goal is to minimize that linear formula.

Example 4. *We illustrate how the two-phase algorithm applies to Example 1. Setting $k = 2$ for instance, the first phase generates at most two potentially useful context candidates for each context size (e.g., the context $\{\langle$"food category","traditional American cuisine"$\rangle\}$ with size one or the context $\{\langle$"food category","traditional American cuisine"$\rangle$, $\langle$"average rating",$[4, 5]\rangle\}$ with size two). In the second phase, we introduce binary variables of the form $w(c, r)$ for each context and each of the four rows to output, indicating whether the row is read out within the corresponding context. We also introduce a variable $g(c)$ for each context candidate c, indicating whether at least one row maps to it. Our cost formula sums over the variables $g(c)$ and $w(c, r)$, weighted by the time it takes to read out the corresponding context or to read out the corresponding row within the associated context.*

5. GREEDY ALGORITHM

Our greedy algorithm consists of two parts. In Section 5.1, we present an algorithm that forms several sets of context candidates. For each context set, it generates the best plan that uses only the context candidates in the set. Finally, it returns the plan with minimal run time among all generated plans. The algorithm in Section 5.1 relies on a sub-function that tries to generate the most promising context candidate in a given situation. We discuss the implementation of that function in Section 5.2. Our greedy algorithm is deliberately kept simple to minimize optimization overhead. Nevertheless, we show that it finds at least near-optimal solutions under several simplifying assumptions.

5.1 Main Function

Algorithm 2 greedily generates voice output for a given relation. Function GreedyVOO takes as input a relation and returns a corresponding output plan. The main idea underlying that function is the following. We decompose plan generation into two steps: First, we choose what set of context candidates the plan may use. Then, we map each relation row to the context which minimizes its output time. When mapping rows to context candidates, we only consider the time required for reading out rows within a context but not the time required for reading out the context itself.

Function GreedyVOO initially generates a naive output plan (reading out one row after the other one) that does not use any context. Next, it initializes a set of context candidates that is extended by one context in each iteration. As discussed in Section 3, an optimal output plan uses at most one context per row pair. Hence, the size of the largest set of context candidates that we consider is half the number of rows in the input relation. Each iteration of the for loop

```
// Use contexts in C to generate fastest output plan
// for relation R.
function MinTimePlan(C, R)
    // Collect unmatched rows
    U ← {r ∈ R|∄c ∈ C :Matches(c, r)}
    // Start speech with those
    S ←Speech(U)
    // Continue with matched rows
    R ← R \ U
    // Iterate over available contexts
    for c ∈ C do
        // Which rows match that context?
        M ← {r ∈ R|Matches(c, r)}
        // Which rows favor that context?
        F ← {r ∈ M|T(c, r) = min_{c̃∈C} T(c̃, r)}
        // Any row favors current context?
        if F ≠ ∅ then
            // Append to speech
            S ← S∘Speech(c, F)
        end if
        // Discard treated rows
        R ← R \ F
    end for
    return S
end function
// Greedily optimize voice output for relation R.
function GreedyVOO(R)
    // Initialize context set
    C ← ∅
    // Generate plan without contexts
    naivePlan ←MinTimePlan(C, R)
    // Initialize candidate plans
    P ← {naivePlan}
    // Up to maximal number of useful contexts
    for i ∈ {1, ..., ⌊|R|/2⌋} do
        // Generate most promising context
        c* ←BestContext(C, R)
        // Add context to set
        C ← C ∪ {c*}
        // Best plan for given context set
        p* ←MinTimePlan(C, R)
        // Add to plan candidates
        P ← P ∪ {p*}
    end for
    // Return best plan among candidates
    return arg min_{p∈P} T(p)
end function
```

Algorithm 2: Greedy algorithm for generating near-optimal voice output for a given relation.

adds one context candidate. That context candidate is generated by an invocation of function BestContext which we discuss in the next subsection. For each set of context candidates, we generate a plan by choosing an optimal mapping from relation rows to context candidates. Finally, we return the plan with minimal run time (function $T(p)$ returns the speaking time for plan p).

Function GreedyVOO uses sub-function MinTimePlan to identify the best plan for a fixed context set (i.e., we assume that each context in the set is generated anyway and thereby simplify the problem compared to the one we solve in Section 4.2). We discuss the implementation of that function next. Function MinTimePlan first identifies all rows that do not match any context in the given set. We start the output speech by reading out those rows one after the other one (we use function Speech to generate naive output for a set of rows). Next, we focus on the remaining rows that match one or several context candidates in the set. We iterate over the set of context candidates and select for each context all rows that have minimal output time under that context (among all context candidates). We extend the speech by outputting each of those rows (if any) within the current context. We use function Speech with two parameters to output a row set within a given context, we concatenate speech fragments by the $\circ$ operator. We continue until all rows have been assigned to a context and are included in the speech.

Next, we show that Algorithm 2 generates near-optimal plans assuming that function BestContext returns always the best context candidate (this assumption is simplifying). Our analysis is based on a diminishing returns property when generating more and more context candidates. Intuitively, the more context candidates we have already, the more likely it is that one of them is similar to a new context. We formalize this intuition and introduce the *Time Savings* of a context set C for relation R:

$$Savings(C, R) = \sum_{r \in R} (T(r) - \min_{c \in C : \text{Matches}(c,r)} T(c, r))$$

We prove several properties of time savings.

Lemma 3. *Time savings is submodular in the context set.*

Proof. We set $\mathcal{U}(C) = Savings(C, R)$ for an arbitrary but fixed relation R in the following. We need to show that $\mathcal{U}(C \cup \{c\}) - \mathcal{U}(C) \geq \mathcal{U}(C' \cup \{c\}) - \mathcal{U}(C')$ for an arbitrary context c and two arbitrary context sets $C \subseteq C'$. The additional time gain $\mathcal{U}(C \cup \{c\}) - \mathcal{U}(C)$ when adding one more context c is determined by the sum, over all rows for which c becomes the ideal context, of the additional gain per row. Replacing C in the last expression by a superset C' means that the new context c is ideal for the same rows as before or a subset. Also, the additional gain per row is at most the same as before. Taken together, this implies diminishing returns when adding more and more context candidates. □

Lemma 4. *Time savings are non-negative.*

Proof. It is $T(c, r) \leq T(r)$ for each context c matching r (since we can omit all attributes fixed by the context when reading out row r within context c). Hence, time savings is a sum over non-negative terms. □

Lemma 5. *Time savings are monotone in the context set.*

Proof. We subtract the minimum over all context candidates when calculating time savings. Hence, increasing the context set can only increase time savings. □

Based on the previous lemmata, we can lower-bound the output quality of Algorithm 2. This bound refers to the *Total Time Savings*, by which we mean the time difference of a naive plan compared to the greedy plan (note the distinction to time savings where we do not take into account the overhead for reading out context). Besides the simplifying assumption that function BestContext returns optimal results, we make another assumption: we assume that

the overhead for outputting each context is a constant. This assumption is simplifying but not unreasonable: the number of values that can be fixed by a context is typically restricted by a small constant (m_S) such that a large part of context output time is due to reading out boilerplate text (which is the same for each possible context).

THEOREM 3. *Algorithm 2 generates a plan with total time savings within factor* $1 - 1/e$ *of the optimum.*

PROOF. We assume a constant time overhead per context. The time overhead for reading out context depends therefore only on the number of used contexts. The total time savings are the sum of the time savings minus the time overhead for reading context. Therefore, if we find for each possible number of used contexts a plan with maximal time savings then the optimal plan is one of them. We find for a given context set a plan with maximal time savings (via function MINTIMEPLAN). However, we may not find the context set leading to optimal time savings among all sets with the same cardinality. Still, we find a near-optimal context set as justified in the following. If we greedily select k elements in order to maximize a monotone, non-negative, and submodular function (i.e., we always select the element leading to the biggest increase) then we find a solution within factor $1 - 1/e$ of the optimum [17]. Time savings, as a function of the context set, has all required properties as shown by the previous three lemmata. Also, we currently assume that function BESTCONTEXT adds the context with optimal time savings. We do not know a-priori what number of context candidates will lead to an optimal solution. However, we simply keep the best plan for each context set cardinality and determine finally the optimum among them. □

5.2 Generating a Good Context

The greedy algorithm relies on a function for generating good context candidates (function BESTCONTEXT in Algorithm 2). We describe how to implement function BESTCONTEXT in the following.

We denote by C_{prev} the set of previously generated context candidates for the current output relation R. The set $R_{unm} = \{r \in R | \nexists c \in C_{prev} : \text{MATCHES}(c, r)\}$ is the subset of rows in R that is not matched by any of the previous context candidates. Intuitively, we should prioritize matching those rows when generating the new context.

We model a context as a set of domain assignments. Each domain assignment assigns one attribute to a value domain. We denote by $\mathcal{A}$ the set of all relevant domain assignments. For each numerical attribute a, we add the assignment pair $\langle a, [l, u]\rangle$ to $\mathcal{A}$ where l and u are lower and upper bounds with $l < u$ and $u \leq l \cdot m_W$. Upper and lower bounds correspond to values for attribute a that we find among the unmatched rows R_{unm}. For each categorical attribute a, we add the assignment pair $\langle a, D\rangle$ to $\mathcal{A}$, where D is a subset of the categorical values that we find among the unmatched rows for attribute a. Additionally, we only consider subsets D of sufficiently small cardinality (i.e., $|D| \leq m_C$).

We consider all subsets $c \subseteq \mathcal{A}$ of assignments that satisfy the following two constraints. First, we consider only subsets that satisfy our constraint on the context size, i.e. subsets c with $|c| \leq m_S$. Second, we only consider subsets that contain for each attribute at most one assignment, i.e. there is no attribute a and distinct domains X and Y such that $\langle a, X\rangle \in \mathcal{A}$ and $\langle a, Y\rangle \in \mathcal{A}$. We write $S(c)$ in the following if c satisfies both constraints.

In line with the assumptions from the previous subsection, we aim at generating a context that maximizes time savings for the unmatched rows. In summary, we want to generate a context c^* with the following property:

$$c^* = \arg \max_{c \subseteq \mathcal{A}: S(c)} Savings(\{c\}, R_{unm})$$

Next, we show that the above problem is an instance of submodular maximization for which efficient approximation algorithms exist. We have shown that time savings are submodular in the context candidates (Lemma 3). In the following, we show that time savings for a single context is also submodular with regards to that context's assignment set.

LEMMA 6. *Time savings of a single context is submodular in the assignment set.*

PROOF. We need to show that adding an assignment to a context (i.e., set of assignments) c does not increase time savings more than adding the same assignment to a superset of c. The additional savings when adding one new assignment $\langle a, D\rangle$ to c is calculated as follows: we sum the time for outputting the value for attribute a over all rows that match the context $c \cup \{\langle a, D\rangle\}$. For the rows that do not match the new context there are two possibilities. Either a row does not have a value within domain D for attribute a (i.e., it is discarded by adding the new assignment) or the row is discarded based on some other assignment in c. The set of rows discarded due to the latter case is monotone in c (i.e., the more assignments we have, the less rows will qualify). Hence, we have diminishing returns when adding more assignments. □

Our problem of generating an optimal context reduces to the problem of optimizing a submodular function. Note that we optimize a non-monotone function: adding a new assignment specializes the context and may reduce time savings if it reduces the number of matching rows. We need to consider this fact when selecting an algorithm for submodular optimization (e.g., we cannot use the classical algorithm by Nemhauser and Wolsey [17] as it does not offer any guarantees in this case). Both our constraints (at most m_S assignments and at most one assignment per attribute) are instances of matroid constraints (i.e., the uniform matroid and the partition matroid). We can use the greedy algorithm by Mirzasoleiman et al. [16] to maximize a submodular function under matroid constraints. This algorithm has polynomial run time and produces solutions with quality bounds, leading to the following guarantee for our algorithm.

THEOREM 4. *Function* BESTCONTEXT *generates a context with time savings within factor* 1/7.5 *of the optimum among unmatched rows.*

PROOF. This is a consequence of the result by Mirzasoleiman et al. [16] establishing a worst-case guarantee of factor $k/((k+1)\cdot(2\cdot k+1))$ for maximizing a non-monotone submodular function under k matroid constraints. Lemma 6 shows that time savings is submodular and we have two matroid constraints (uniform and partition matroid). □

We derived lower bounds on output quality based on simplifying assumptions. While those guarantees are not very strong, we show in Section 7 that average performance is significantly better than the guarantees.

EXAMPLE 5. *We illustrate the greedy algorithm on Example 1. We perform two iterations as an optimal solution uses at most two contexts. In the first iteration, we consider all domain assignments that are possible using the values in R. From those, we select a (near-)optimal subset with cardinality at most m_S to form a new context (e.g., the context "Entries for category traditional American cuisine"). We generate a plan by assigning each tuple to the context with maximal time savings. For the second iteration, we remove all rows from R that are covered by the context that was generated before (e.g., we remove restaurant John's and restaurant Upstate). We generate possible domain assignments for the remaining rows and select an optimal subset (i.e., a new context) again. A new plan is generated by assigning each row to an optimal context (considering the two previously generated contexts and the empty context). Finally, we return the optimum among all generated plans.*

6. COMPLEXITY ANALYSIS

We analyze time complexity of the algorithms that we presented in the previous sections. For the integer linear programs resulting from our problem transformations, we analyze the asymptotic number of variables in the generated programs. It is not possible to calculate the asymptotic time required to solve an integer program in general as it depends on the used solver. However, the search space to explore grows in the number of variables and optimization time tends to follow.

First, we analyze the size of the integer programs generated by the pure integer programming approach described in Section 3. We denote by n_R the number of rows in the relation to output (which is at the same time proportional to the number of context slots that we create), we use n_A for the number of attributes, and n_V for the maximal number of distinct values in any column.

THEOREM 5. *The MILP representation of voice output optimization uses $O(n_R \cdot n_A \cdot (n_R + n_V))$ variables.*

PROOF. The number of context slots is linear in the number n_R of rows. Hence, the number of variables $w(c, r)$ mapping rows to slots is quadratic in n_R. The number of variables $f(c, a)$ is in $O(n_R \cdot n_A)$ and therefore dominated by the number of variables $d(c, a, v)$, $l(c, a, v)$, and $u(c, a, v)$ which is in $O(n_R \cdot n_A \cdot n_V)$. To estimate speaking time, we introduce $O(n_R \cdot n_R \cdot n_A)$ variables $s(c, r, a)$, $O(n_R \cdot n_A \cdot n_V)$ variables $e(c, a, v)$, and $O(n_R)$ variables $g(c)$. □

Next, we analyze the two-phase algorithm from Section 4. First, we analyze the time complexity of the pre-processing stage in which context candidates are generated.

THEOREM 6. *Generating context candidates takes time in $O(k^2 \cdot n_A \cdot n_V^{\max(2,m_C)} \cdot m_S^2 \cdot n_R)$.*

PROOF. We keep at most k context candidates after each iteration. We extend those candidate by adding assignments from attributes to value domains. We consider $O(n_V^2)$ value domains for numerical attributes and $O(n_V^{m_C})$ domains for categorical attributes. By adding one assignment to each of the k context candidates, we obtain therefore $O(k \cdot n_A \cdot n_V^{\max(2,m_C)})$ candidates for pruning in each iteration. Deciding whether one specific context is potentially useful takes $O(m_S \cdot n_R)$ time (we compare at most m_S assignments to verify whether a row matches a context). Hence, the complexity of pruning is $O(k \cdot n_A \cdot n_V^{\max(2,m_C)} \cdot m_S \cdot n_R)$. The greedy algorithm for selecting the k best context candidates performs k iterations and compares $O(k \cdot n_A \cdot n_V^{\max(2,m_C)})$ candidates in each step. For each candidate, it calculates the number of covered rows (in time $O(m_S \cdot n_R)$). Hence, the complexity of selecting k candidates is $O(k^2 \cdot n_A \cdot n_V^{\max(2,m_C)} \cdot m_S \cdot n_R)$. Finally, we multiply the complexity of both steps by the number of iterations which is m_S. □

Now, we analyze the size of the integer program created to select between the generated context candidates. In addition to the previous notations, we designate by $n_C \in O(k \cdot m_S)$ the number of useful context candidates.

THEOREM 7. *The integer program that selects context candidates uses $O(n_R \cdot n_C)$ variables.*

PROOF. We introduce $O(n_R \cdot n_C)$ variables of the form $w(c, r)$ and $O(n_C)$ variables of the form $g(c)$. □

Finally, we analyze time complexity of the greedy algorithm from Section 5. First, we analyze the time complexity of the sub-function BESTCONTEXT. Its complexity depends on the method that is used to solve the submodular maximization problem. We assume that the algorithm by Mirzasoleiman et al. [16] is used.

LEMMA 7. *Generating a near-optimal context takes time in $O(n_R \cdot m_S \cdot (m_S \cdot n_A \cdot n_V^{\max(2,m_C)} + n_R))$.*

PROOF. We initially test for a match between each pair of a row and a context candidate. We have n_R rows and at most $n_R/2$ context candidates. Testing whether a row matches a context takes $O(m_S)$ time as justified before. Hence, we retrieve unmatched rows in $O(n_R^2 \cdot m_S)$ time. The number of assignments is in $O(n_A \cdot n_V^{\max(2,m_C)})$. The complexity of the algorithm by Mirzasoleiman et al. for submodular maximization is in general $n \cdot r \cdot p$ times the complexity of evaluating the submodular function where n designates the number of elements to choose from, r the maximal number of elements in an optimal solution, and p the number of matroid constraints. We select at most m_S out of $O(n_A \cdot n_V^{\max(2,m_C)})$ assignments and have two matroid constraints. Calculating time gain takes $O(n_R \cdot m_S)$ time. □

The time complexity of Algorithm 2 follows immediately.

THEOREM 8. *The greedy algorithm has time complexity $O(n_R^2 \cdot m_S \cdot (m_S \cdot n_A \cdot n_V^{\max(2,m_C)} + n_R))$.*

PROOF. We perform $O(n_R)$ iterations of the for loop. In each iteration, we generate a new context and generate an optimal plan with the new set of context candidates. We can generate a plan in $O(m_S \cdot n_R^2)$ time. Hence, the complexity of generating a new context dominates. □

7. EXPERIMENTAL EVALUATION

We compare the algorithms presented in the last subsections against a naive baseline (i.e., reading out one row after the other). We compare algorithms in terms of optimization time and in terms of the quality of the generated output. To judge output quality, we measure speaking time and ask crowd workers to compare alternative versions.

Our test cases are derived from four data sets. First, we use restaurant recommendations returned by Google Maps

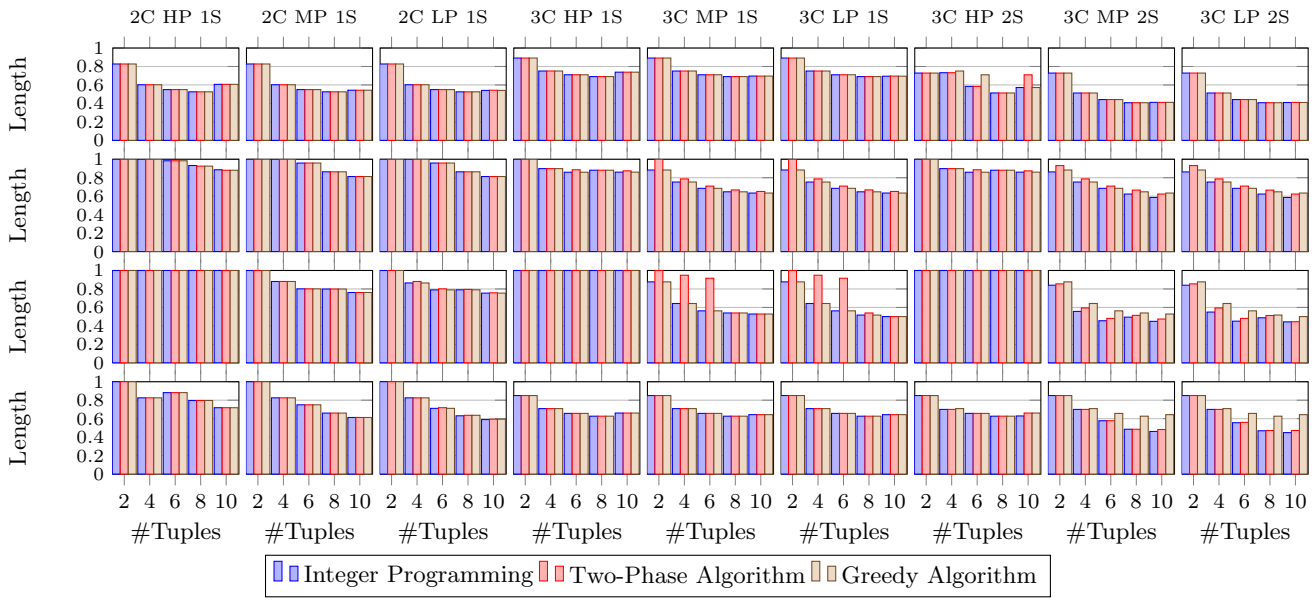


Figure 2: Scaled length of voice output generated by different methods for four data sets (from upper row down: laptops, restaurants, football statistics, mobile phones), when reading out two columns (2C) or three columns (3C), under varying constraints on precision (low, medium, or high precision: LP, MP, or HP) and context size (fix up to one or two attributes: 1S or 2S).

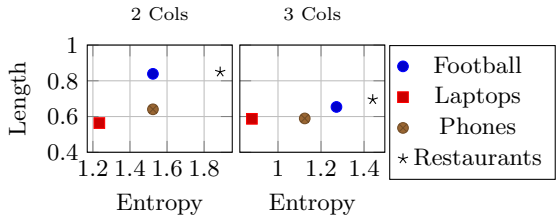


Figure 3: Entropy versus time savings.

when searching for restaurants around Time Square in New York City (using rating and food type as attributes besides the restaurant name). Second, we use descriptions of laptop models with attributes such as model name, price, and main memory size. Third, we use summaries of football games with attributes such as team name, the number of wins, and affiliations. Finally, we use descriptions of mobile phone models with attributes such as model name, operating system, and storage capacity. All data sets refer to situations where voice output seems appropriate (e.g., inform traveling users of nearby dining options via voice output from a mobile device, or inform users of shopping options via voice output from Google Home or a similar device).

Our algorithms are implemented in Java 1.8. All of the following experiments are executed on a MacBook Pro with Intel Core i7 2.3 GHz CPU and 8 GB of main memory, running MacOS 10.12. We use CPLEX in version 12.7 as integer programming solver and the IBM Watson text to speech service[2] to synthesize voice output. To minimize the number of speech fragments that we generate, we use as optimization metric the number of characters in the generated output (instead of the speaking time that we ultimately want to minimize). We found that the number of characters is sufficiently correlated with the speaking time. Speech output is only generated for the output plan that is finally selected.

We compare the integer programming algorithm from Section 3 (with a timeout of 300 seconds) against the two-phase algorithm from Section 4 (setting k to 20) and the greedy algorithm from Section 5. Figure 2 compares length of voice output generated by different methods for the same data (we report arithmetic means of 10 test cases). We scale output length to the length of naive voice output (reading out one row after the other). We compare methods in different scenarios, varying data set size and configuration parameters. We experiment with high output precision ($m_C = m_W = 1$), medium precision ($m_C = m_W = 2$), and low precision ($m_C = 2$, $m_W = 4$). We focus on data sets where voice output is a realistic option (i.e., speaking time of several tens of seconds up to a minute).

The potential for speaking time reduction generally increases in the number of tuples. This seems logical since having more tuples means more redundant values that we can avoid reading out via our approaches. Equally, having more columns leads often to increased time savings. This effect is amplified once we allow larger context sizes that can summarize values in multiple columns concurrently. Allowing less precise output equally enables further time savings as more tuples can be summarized in the same context. There are however diminishing returns and decreasing precision has little effect on speaking time after a certain point. The plans generated by integer programming are generally optimal. Both, the greedy algorithm and the two-phase approach, produce however in most cases plans that are very close to the optimum. The two-phase approach has a slight edge when approaching the maximal number of 10 tuples. All three methods achieve time savings of up to factor 2.5.

[2] `https://www.ibm.com/watson`

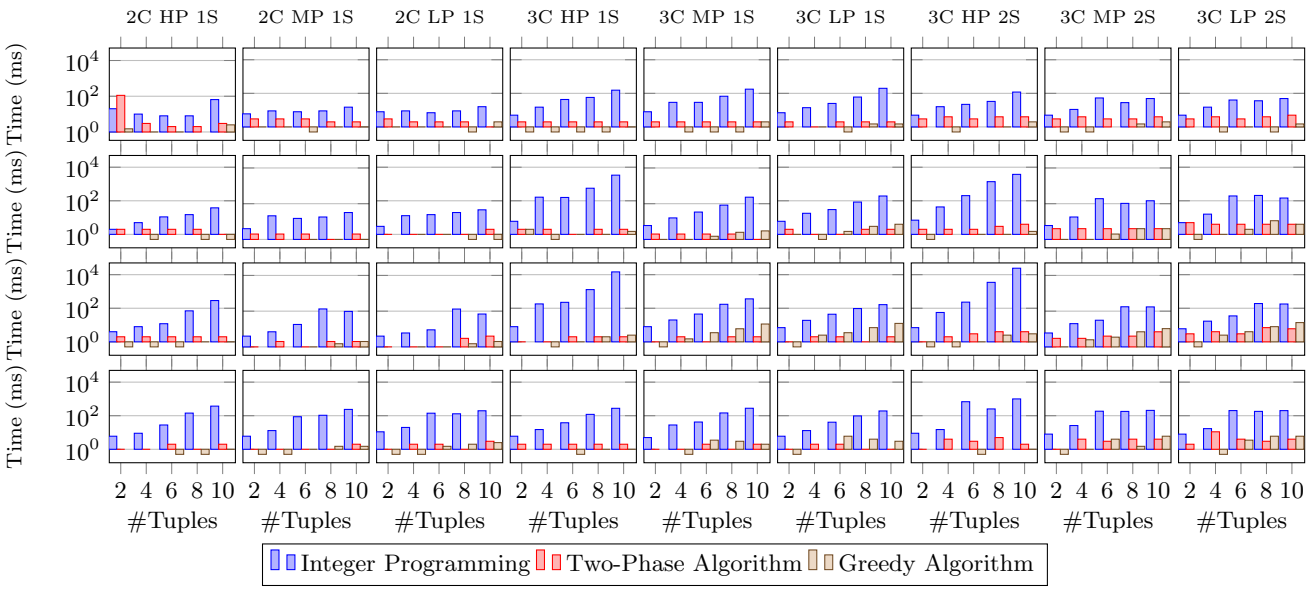


Figure 4: Optimization times of different vocalization methods for four data sets (from upper row down: laptops, restaurants, football statistics, mobile phones), when reading out two columns (2C) or three columns (3C), under varying constraints on precision (low, medium, or high precision: LP, MP, or HP) and context size (fix up to one or two attributes: 1S or 2S).

Time savings vary across different scenarios. Given that our approach reduces speaking time by reducing redundancy, we suspect that data sets with a higher amount of redundancy benefit more. Figure 3 verifies that intuition by correlating the average raw entropy over all columns (measuring the amount of non-redundant information) with average time savings for ten tuples across different setting for the context (i.e., average of exact and approximate settings). We bucketize numerical columns into intervals of relative length m_W to calculate their entropy. Indeed, we observe a slight correlation between entropy and time savings.

Figure 4 shows optimization time for the test cases in Figure 2. Clearly, the integer programming approach is the most expensive one. Even though optimization time is typically below a second, we exceed 10 seconds of optimization time in a few cases. Greedy algorithm and two-phase approach have optimization times in the order of milliseconds.

Next, we look at some larger problem instances where speaking time is rather large (this makes them less interesting for the average user while those instances might be relevant for visually impaired users). Figure 5 shows optimization time and relative speech length for up to seven columns and 50 tuples in the mobile phones scenario (we set $m_C = m_W = m_S = 2$). The integer programming approach often reaches the timeout of five minutes starting from four columns and 40 tuples (we return the naive plan in case of a timeout which is why the scaled output length converges to one). The greedy and two-phase approach can be applied under run time constraints even in those extreme cases. The greedy algorithm achieves slightly better quality for high numbers of tuples and columns while the two-phase approach is slightly faster (few hundreds of milliseconds).

Reducing speaking time generally saves time for users, compared to reading out row after row. On the other side, the speech structure becomes slightly more complicated which might increase cognitive load. We performed a user study to find out what version users prefer under specific circumstances. We based our study on AMT[3] and asked crowd workers to compare alternative voice output versions. We presented the naive version and the optimal version according to our model to crowd workers and asked them to vote on their preferred version. Our test cases describe laptops, we vary the number of tuples and the number of columns. We asked ten crowd workers with at least 50 approved HITS per test case and payed 10 cents per comparison task.

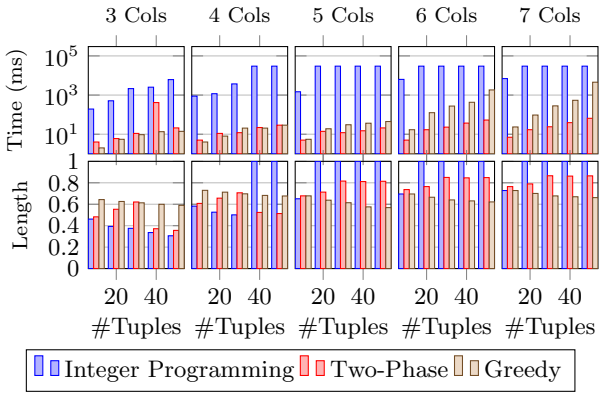
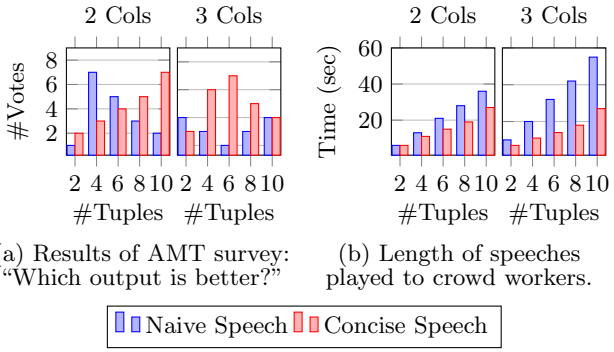


Figure 5: Optimization time and relative speech length for XXL instances.

Figure 6 reports the results of our user study and correlates them with absolute speaking times for the different versions. Votes do not always sum up to ten due to workers who selected the option that both versions are equivalent (also, we had a single unsolved test case for two columns

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



(a) Results of AMT survey: "Which output is better?"

(b) Length of speeches played to crowd workers.

Figure 6: Speech length and user satisfaction.

and two tuples). We experiment with two columns (setting $m_S = 1$, $m_C = 1$, and $m_W = 1.5$) and three columns (setting $m_S = 2$, $m_C = 1$, and $m_W = 2$). We vary the number of tuples between two and ten, resulting in speaking times up to roughly one minute. For a low number of tuples, users seem to prefer the naive version or are indifferent. This correlates with minor savings in speaking time. As the number of tuples and the time gap between the two versions grows, user preferences shift towards the concise speech generated by our approach. However, once speaking time becomes relatively large for both versions, most users find both versions equivalent. We believe that we are entering into a range of speaking times where both versions are perceived as too long and the amount of data should be reduced.

We asked workers to justify their choices and the reasons match our expectations. Reasons to pick the concise version included for instance "brief yet still gave informative answers" or "short and sweet description". Reasons to pick the row-by-row version (without approximation) included "more details" and "more comprehensive information". Hence, users value conciseness under certain conditions. Finding good policies to select between simple and concise version is an interesting direction for future work.

8. RELATED WORK

Prior work [7, 11, 14, 19] on generating natural language descriptions of data sets focuses on producing written text. We focus on *Voice Output* which is subject to specific constraints [18]: it has to be extremely concise (as opposed to written text, users cannot skim text to identify relevant parts quickly) and has to respect memory limitations of the listener (as opposed to written text, users cannot easily reread prior passages). Those constraints motivate our approach to vocalization as *Global Optimization Problem*, considering even the possibility to trade precision for conciseness. At the same time, focusing on concise output creates the opportunity to use optimization methods that would not scale to the generation of multi-page documents. Our approach operates on *Relational Data* which distinguishes it from prior work on document summarization [8] and text compression [3] (which uses text as input). Data sonification [5], as opposed to vocalization, focuses typically on transforming data into non-speech audio. Approaches to information presentation [4] in spoken dialogue systems are typically specific to scenarios where users select one out of several options (e.g., flights).

Our work is complementary to prior work on translating natural language input into SQL queries [2, 12] or queries into natural language output [9, 10]. It differs in focus and methods from general data summarization techniques [6] which do not result in natural language text.

9. CONCLUSION

Current trends towards voice-based interfaces motivate the problem of data vocalization, a complementary problem to data visualization. We introduce a variant of data vocalization where the goal is to reduce speaking time under constraints on the precision of the transmitted information. We propose multiple exhaustive and non-exhaustive algorithms and compare them theoretically and empirically.

10. ACKNOWLEDGMENTS

We thank Hongrae Lee for helpful discussions. Our research on data vocalization is supported by a Google Faculty Research Award.

11. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, volume 1215, pages 487–499, 1994.
[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering*, 1(1):29–81, 1995.
[3] J. Clarke and M. Lapata. Global inference for sentence compression - an integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429, 2008.
[4] V. Demberg and J. D. Moore. Information presentation in spoken dialogue systems. In *EACL*, pages 65–72, 2006.
[5] T. Hermann, A. Hunt, and J. G. Neuhoff. *The sonification handbook*. 2011.
[6] Z. R. Hesabi, Z. Tari, A. Goscinski, A. Fahad, I. Khalil, and C. Queiroz. Data summarization techniques for big data-a survey. In *Handbook on Data Centers*, pages 1109–1152. 2015.
[7] 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.
[8] H. Jing and K. R. McKeown. Cut and paste based text summarization. In *ACL*, pages 178–185, 2000.
[9] A. Kokkalis, P. Vagenas, A. Zervakis, A. Simitsis, G. Koutrika, and Y. Ioannidis. $\lambda\gamma$: A system for translating queries into narratives. In *SIGMOD*, pages 673–676, 2012.
[10] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *ICDE*, pages 333–344, 2010.
[11] K. Kukich. Design of a knowledge-based report generator. In *Annual Meeting on Association for Computational Linguistics*, pages 145–150, 1983.
[12] F. Li and H. Jagadish. Understanding natural language queries over relational databases. *SIGMOD Record*, 45(1):6–13, 2016.
[13] 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.
[14] K. McKeown, J. Robin, and K. Kukich. Generating concise natural language summaries. *Information Processing and Management*, 31(5):703–733, 1995.
[15] 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.
[16] B. Mirzasoleiman, A. Badanidiyuru, and A. Karbasi. Fast constrained submodular maximization: personalized data summarization. In *ICML*, pages 1358–1367, 2016.
[17] 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.
[18] T. V. Raman. *Audio system for technical readings*. PhD thesis, 1998.
[19] A. Simitsis, Y. Alexandrakis, G. Koutrika, and Y. Ioannidis. Synthesizing structured text from logical database subsets. In *EDBT*, pages 428–439, 2008.