The Complexity of Aggregates over Extractions by Regular Expressions

Johannes Doleschal
Universität Bayreuth, Germany
Hasselt University, Belgium

Noa Bratman
Technion - Israel Institute of Technology, Haifa, Israel

Benny Kimelfeld
Technion - Israel Institute of Technology, Haifa, Israel

Wim Martens
Universität Bayreuth, Germany

Abstract

Regular expressions with capture variables, also known as "regex-formulas," extract relations of spans (intervals identified by their start and end indices) from text. In turn, the class of regular document spanners is the closure of the regex formulas under the Relational Algebra. We investigate the computational complexity of querying text by aggregate functions, such as sum, average, and quantile, on top of regular document spanners. To this end, we formally define aggregate functions over regular document spanners and analyze the computational complexity of exact and approximate computation. More precisely, we show that in a restricted case, all studied aggregate functions can be computed in polynomial time. In general, however, even though exact computation is intractable, some aggregates can still be approximated with fully polynomial-time randomized approximation schemes (FPRAS).

2012 ACM Subject Classification Information systems → Information extraction; Theory of computation → Database query languages (principles); Theory of computation → Problems, reductions and completeness

Keywords and phrases Information extraction, document spanners, regular expressions, aggregation functions

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.10

Funding This work was supported by the German-Israeli Foundation for Scientific Research and Development (GIF), grant I-1502-407.6/2019. The work of Benny Kimelfeld was also supported by the Israel Science Foundation (ISF), Grant 768/19, and the German Research Foundation (DFG) Project 412400621 (DIP program).

1 Introduction

Information extraction commonly refers to the task of extracting structured information from text. A document spanner (or just spanner for short) is an abstraction of an information extraction program: it states how to transform a document into a relation over its spans. More formally, a *document* is a string d over a finite alphabet, a *span* of d represents a substring of d by its start and end positions, and a *spanner* is a function that maps every document d into a relation over the spans of d [7]. The spanner framework has originally been introduced as the theoretical basis underlying IBM's SQL-like rule system for information extraction, namely SystemT [15, 18]. The most studied spanner instantiation is the class of *regular spanners* – the closure of *regex-formulas* (regular expressions with capture variables) under the standard operations of the relational algebra (projection, natural join, union, and difference). Equivalently, the regular spanners are the ones expressible as *variable-set automata* (VSet-automata for short) – nondeterministic finite-state automata that can open and close capture variables. These spanners extract from the text relations wherein the capture variables are the attributes.

While regular spanners and natural generalizations thereof are the basis of rule-based systems for text analytics, they are also used implicitly in other types of systems, and particularly ones based on statistical models and machine learning. Rules similar to regular spanners are used for *feature generators* of graphical models (e.g., Conditional Random Fields) [17, 32], *weak constraints* of Markov Logic Networks [28] and extensions such as DeepDive [31], and the generators of *noisy training data* ("labeling functions") in the state-of-the-art Snorkel system [29]. Further connections to regular spanners can potentially arise from efforts to express artificial neural networks for natural language processing as finite-state automata [21, 22, 34]. The computational complexity of evaluating regular spanners has been well studied from various angles, including the data and combined complexity of answer enumeration [1, 8, 10, 20], the cost of combining spanners via relational algebra operators [26] and recursive programs [27], their dynamic complexity [11], evaluation in the presence of weighted transitions [5], and the ability to distribute their evaluation over fragments of the document [4].

In this paper, we study the computational complexity of evaluating *aggregate functions* over regular spanners. These are queries that map a document d and a spanner P into a number $\alpha(P(d))$, where $P(d)$ is the relation obtained by applying P to d and α is a standard aggregate function: count, sum, average, min, max, or quantile. There are various scenarios where queries that involve aggregate functions over spanners can be useful. For example, such queries arise in the extraction of statistics from textual resources like medical publications [25] and news reports [30]. As another example, when applying advanced text search or protein/DNA motif matching using regular expressions [3, 24], the search engine typically provides the (exact or approximate) number of answers, and we would like to be able to compute this number without actually computing the answers, especially when the number of answers is prohibitively large. Finally, when programming feature generators or labeling functions in extractor development, the programmer is likely to be interested in aggregate statistics and summaries for the extractions (e.g., to get a holistic view of what is being extracted from the dataset, such as quantiles over extracted ages and so on), and again, we would like to be able to estimate these statistics faster than it takes to materialize the entire set of answers.

Our main objective in this work is to understand when it is tractable to compute $\alpha(P(d))$. This question raises closely related questions that we also discuss in the paper, such as when the materialization of intermediate results (which can be exponentially large) can be avoided. Furthermore, when the exact computation of $\alpha(P(d))$ is intractable, we study whether it can be approximated.

At the technical level, each aggregate function (with the exception of count) requires a specification of how an extracted tuple of spans represents a number. For example, the number 21 can be represented by the span of the string "21", "21.0", "twenty one", "twenty first", "three packs of seven" and so on. To abstract away from specific textual representations of numbers, we consider several means of assigning weights to tuples. To this end, we assume that a (representation of a) *weight function* w, which maps every tuple of $P(d)$ into a number, is part of the input of the aggregate functions. Hence, the general form of the aggregate query we study is $\alpha(P, d, w)$. The direct approach to evaluating $\alpha(P, d, w)$ is to compute $P(d)$, apply w to each tuple, and apply α to the resulting sequence of numbers. This approach works well if the number of tuples in $P(d)$ is manageable (e.g., bounded by some polynomial). However, the number of tuples in $P(d)$ can be exponential in the number of variables of P, and so, the direct approach takes exponential time in the worst case. We will identify several cases in which $P(d)$ is exponential, yet $\alpha(P(d))$ can be computed in polynomial time.

It is not very surprising that, at the level of generality we adopt, each of the aggregate functions is intractable (#P-hard) in general. Hence, we focus on several assumptions that can potentially reduce the inherent hardness of evaluation:

- Restricting to positive numbers;
- Restricting to weight functions w that are determined by a single span or defined by (unambiguous) weighted VSet-automata;
- Restricting to spanners that are represented by an unambiguous variant of VSet-automata;
- Allowing for a randomized approximation (FPRAS, i.e., fully polynomial randomized approximation schemes).

Our analysis shows which of these assumptions bring the complexity down to polynomial time, and which is insufficient for tractability. Importantly, we derive an interesting and general tractable case for each of the aggregate functions we study.

The problem of counting the number of extractions of a VSet-automaton has been studied by Florenzano et al. [8]. The approximate version has been studied by Arenas et al. [2] who give a polynomial-time algorithm for uniformly sampling from the space of accepted words of a given length for an NFA, and to estimate the number of such accepted words. Using that sampling, they establish an FPRAS for counting the number of tuples extracted by a VSet-automaton (i.e., the Count aggregate function). Our FPRAS results are also based on their results. The counting problem is also implicitly discussed by Doleschal et al. [5] who study annotation by semiring elements over weighted VSet-automata. Throughout the paper, we explain the connection between all of these and our work in more detail. Yet, to the best of our knowledge, this paper is the first to consider aggregate functions over numerical values extracted by document spanners.

The remainder of the paper is organized as follows. In Section 2, we give preliminary definitions and notation. In Section 3, we give a general summary of the main results of the paper; we expand on these results in the later sections. In Sections 4, 5 and 6 we describe our investigation for single-variable weight functions, polynomial-time weight functions and regular weight functions, respectively. Finally, we consider approximate evaluations in Section 7 and conclude in Section 8. Due to space constraints, we sometimes omit proofs or only provide a proof sketch.

2 Preliminaries

The cardinality of a set A is denoted by $|A|$. A *multiset* over A is a function $M : A \to \mathbb{N}$. We call $M(a)$ the *multiplicity* of a in M and say that $a \in M$ if $M(a) > 0$. The *size* of M denoted $|M|$, is the sum $\sum_{a \in A} M(a)$, which may be infinite. We denote multisets in brackets $\{\!\!\{$ and $\}\!\!\}$ in the usual way. E.g., in $M = \{\!\!\{1, 1, 3\}\!\!\}$ we have that $M(1) = 2$ and $M(3) = 1$.

We revisit some definitions from the *document spanners framework* [7]. Let Σ be a finite set of symbols called the *alphabet*. By Σ^* we denote the set of all finite words over Σ, also called *documents*. The *length* $|d|$ of document $d = \sigma_1 \cdots \sigma_n \in \Sigma^*$ (with every $\sigma_i \in \Sigma$) is n. A *span* of d is an expression of the form $[i, j\rangle$ with $1 \leq i \leq j \leq n+1$. For a span $[i, j\rangle$ of d, we denote by $d_{[i,j\rangle}$ the word $\sigma_i \cdots \sigma_{j-1}$. For a document d, we denote by *Spans(d)* the set of all possible spans of d. Two spans $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ are *equal* if $i_1 = i_2$ and $j_1 = j_2$.

The framework focuses on functions that extract spans from documents and assigns them to variables. To this end, we fix a countably infinite set Var of *span variables*, which range over spans, such that Var and Σ are disjoint. A *d-tuple* t is a total function from a finite set of span variables into *Spans(d)*. We denote the domain of t by $\mathrm{Vars}(\mathrm{t})$. If the document d is clear from the context, we sometimes say *tuple* instead of d-tuple. A set of d-tuples over the same variables is called a *d-relation.* For a d-tuple t and a set $Y \subseteq \mathrm{Vars}(\mathrm{t})$ we define the d-tuple $\mathrm{t}\restriction Y$ as the restriction of t to the variables in Y. A *document spanner* is a function P that maps every document d into a finite d-relation, which we denote by $P(d)$. By $\mathrm{Vars}(P)$ we denote the domain of the tuples in $P(d)$, which we call the variables of the spanner. We refer to Appendix A for the definition of algebraic operations on spanners.

Variable Set-Automata. This paper will focus on *regular spanners*, which can be defined as follows. A *variable-set automaton (VSet-automaton)* is an NFA that accepts words with *variable operations*, which are symbols of the form "$x\vdash$" (open x) and "$\dashv x$" (close x), where x is a variable. More precisely, for a set of variables $V \subseteq \mathsf{Var}$, the set of *variable operations over V* is $\Gamma_V := \{x\vdash, \dashv x \mid x \in V\}$, which we assume to be disjoint from Σ and Var.

A VSet-automaton is a tuple $A := (\Sigma, V, Q, q_0, Q_F, \delta)$, where Σ is a finite set of alphabet symbols, $V \subseteq \mathrm{Vars}$ is a finite set of variables, Q is a finite set of states, $q_0 \in Q$ is a start state, $Q_F \subseteq Q$ is a set of final states, and $\delta : Q \times (\Sigma \cup \Gamma_V \cup \{\varepsilon\}) \to 2^Q$ is the transition function. We refer to words over the alphabet $\Sigma \cup \Gamma_V$ as *ref-words* [9] and, therefore, the *ref-word language* $\mathcal{R}(A)$ of A is the set of words accepted by the NFA $(\Sigma \cup \Gamma_V, Q, q_0, Q_F, \delta)$, which is an ordinary NFA over alphabet $(\Sigma \cup \Gamma_V)$.

We now discuss how A defines a spanner. The set $\mathrm{Vars}(\mathbf{r})$ is the set of variables x such that $x\vdash$ or $\dashv x$ occur in ref-word $\mathbf{r}$. A ref-word $\mathbf{r}$ is *valid* if, for each $x \in \mathrm{Vars}(\mathbf{r})$, it has precisely one occurrence of $x\vdash$ and precisely one occurrence of $\dashv x$, which is after the occurrence of $x\vdash$. It is *valid for V* if it is valid and if $V = \mathrm{Vars}(\mathbf{r})$. By $\mathcal{VR}(A)$ we denote the words in $\mathcal{R}(A)$ that are valid for V.

Consider the mapping $\mathrm{clr} : (\Sigma \cup \Gamma_{\mathrm{Vars}})^* \to \Sigma^*$ (pronounced "clear"), as $\mathrm{clr}(\sigma) := \sigma$ for every $\sigma \in \Sigma$ and $\mathrm{clr}(\sigma) := \varepsilon$ for every $\sigma \in \Gamma_{\mathrm{Vars}}$, where ε denotes the empty word. If $\mathbf{r}$ is a valid ref-word with $\mathrm{clr}(\mathbf{r}) = d$, with $\mathbf{r} = \mathbf{r}_x^{\mathrm{pre}} \cdot x\vdash \cdot \mathbf{r}_x \cdot \dashv x \cdot \mathbf{r}_x^{\mathrm{post}}$, then the *$d$-tuple* $\mathrm{tup}_{\mathbf{r}}$ *induced by* $\mathbf{r}$ is defined as $\mathrm{tup}_{\mathbf{r}}(x) := [i, j\rangle$, where $i := |\,\mathrm{clr}(\mathbf{r}_x^{\mathrm{pre}})| + 1$ and $j := i + |\,\mathrm{clr}(\mathbf{r}_x)|$ for every variable $x \in \mathrm{Vars}(\mathbf{r})$. The *spanner* $[\![A]\!]$ induced by A maps each document d to $\{\mathrm{tup}_{\mathbf{r}} \mid \mathbf{r} \in \mathcal{VR}(A), \mathrm{clr}(\mathbf{r}) = d\}$.

Notice that only the valid ref-words of A produce output tuples. A VSet-automaton is *functional* if it only accepts valid ref-words, i.e., $\mathcal{VR}(A) = \mathcal{R}(A)$. Since VSet-automata can always be translated into equivalent functional VSet-automata [9, Proposition 3.9], we assume in this paper that VSet-automata are functional. This is a common assumption for document spanners involving regular languages [7, 9, 26]. In the following, we denote by VSA the class of functional VSet-automata.

Regular spanners can also be represented by *regex-formulas*, which are regular expressions that may include variables. We refer to Appendix B for a formal definition.

Aggregate Queries. Aggregation functions, such as min, max, and sum operate on numerical values from database tuples, whereas all the values of d-tuples are spans. Yet, these spans may represent numerical values, from the document d, encoded by the captured words (e.g., "3," "three," "March" and so on). To connect spans to numerical values, we will use *weight functions w* that map document/tuple pairs to numbers in $\mathbb{Q}$, that is, if d is a document and t is a d-tuple then $w(d, \mathrm{t}) \in \mathbb{Q}$. We discuss weight functions in more detail in Section 3.3.

```
There␣are␣7␣events␣in␣Belgium,␣10-15␣in␣France,␣4␣in␣Luxembourg,␣three␣in␣Berlin.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
```

$d_{x_{\mathsf{loc}}}$	$d_{x_{\mathsf{events}}}$	$w(d, \mathrm{t})$
Belgium	7	7
France	10-15	10
Luxembourg	4	4
Berlin	three	3

x_{loc}	x_{events}
$[23, 30\rangle$	$[11, 12\rangle$
$[41, 47\rangle$	$[32, 37\rangle$
$[54, 64\rangle$	$[49, 50\rangle$
$[75, 81\rangle$	$[66, 71\rangle$

Figure 1 A document d (top), a relation with corresponding weights (bottom left), and the corresponding d-relation R (bottom right).

▶ **Example 2.1.** Consider the document in Figure 1 and assume that we want to calculate the total number of mentioned events. The table at the bottom left depicts a possible extraction of locations with their number of events, where each tuple is annotated with a weight $w(d, \mathrm{t})$. The table on the bottom right depicts the corresponding span relation. To get an understanding of the total number of events, we may want to take the sum over the weights of the extracted tuples, namely $7 + 10 + 4 + 3 = 24$.

For a spanner P, a document d, and weight function w, we denote by $\mathit{Img}(P, d, w)$ the set of weights of output tuples of P on d, that is, $\mathit{Img}(P, d, w) = \{w(d, \mathrm{t}) \mid \mathrm{t} \in P(d)\}$. Furthermore, let $\mathit{Img}(w) \subseteq \mathbb{Q}$ be the set of weights assigned by w, that is, $k \in \mathit{Img}(w)$ if and only if there is a document d and a d-tuple t with $w(d, \mathrm{t}) = k$.

▶ **Definition 2.2.** *Let d be a document and A be a VSet-automaton such that $[\![A]\!](d) \neq \emptyset$. Let $P = [\![A]\!]$, let w be a weight function, and $q \in \mathbb{Q}$ with $0 \leq q \leq 1$. We define the following spanner aggregation functions:*

$$\mathrm{Count}(P, d) := |P(d)|$$

$$\mathrm{Min}(P, d, w) := \min_{\mathrm{t} \in P(d)} w(d, \mathrm{t}) \qquad \mathrm{Max}(P, d, w) := \max_{\mathrm{t} \in P(d)} w(d, \mathrm{t})$$

$$\mathrm{Sum}(P, d, w) := \sum_{\mathrm{t} \in P(d)} w(d, \mathrm{t}) \qquad \mathrm{Avg}(P, d, w) := \frac{\mathrm{Sum}(P, d, w)}{\mathrm{Count}(P, d)}$$

$$q\text{-Quantile}(P, d, w) := \min \left\{ r \in \mathit{Img}(P, d, w) \;\middle|\; \frac{|\{\mathrm{t} \in P(d) \mid w(d, \mathrm{t}) \leq r\}|}{|P(d)|} \geq q \right\}$$

Observe that $0\text{-Quantile}(P, d, w) = \mathrm{Min}(P, d, w)$ *and* $1\text{-Quantile}(P, d, w) = \mathrm{Max}(P, d, w)$.

Main Problems. Let $\mathcal{P}$ be a class of regular document spanners and $\mathcal{W}$ be a class of weight functions. We define the following problems.

COUNT[$\mathcal{P}$]	SUM[$\mathcal{P}, \mathcal{W}$]
Input: Spanner $P \in \mathcal{P}$ and document $d \in \Sigma^*$.	Input: Spanner $P \in \mathcal{P}$, document $d \in \Sigma^*$, a weight function $w \in \mathcal{W}$.
Task: Compute $\mathrm{Count}(P, d)$.	Task: Compute $\mathrm{Sum}(P, d, w)$.

The problems AVERAGE[$\mathcal{P}, \mathcal{W}$], q-QUANTILE[$\mathcal{P}, \mathcal{W}$], MIN[$\mathcal{P}, \mathcal{W}$], and MAX[$\mathcal{P}, \mathcal{W}$] are defined analogously to SUM[$\mathcal{P}, \mathcal{W}$]. Notice that all these problems study *combined complexity*. Since the number of tuples in $P(d)$ is always in $O(|d|^{2k})$, where k is the number of variables of the

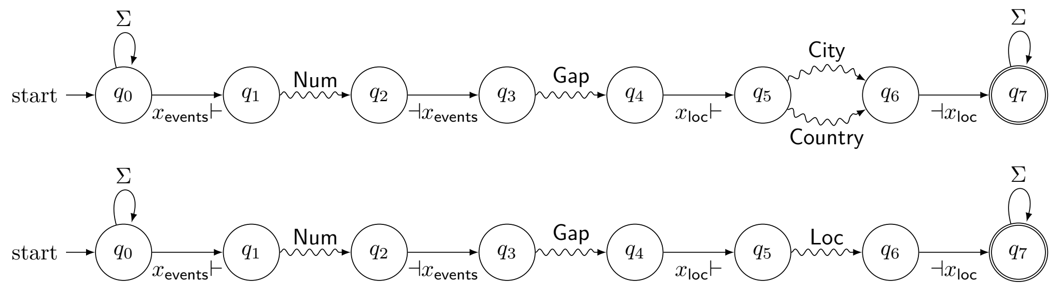

Figure 2 Two example VSet-automata that extract the d-relation R on input d as defined in Figure 1. For the sake of presentation, the automata are simplified as follows: Num is a sub-automaton matching anything representing a number (of events) or range, Gap is a sub-automaton matching sequences of at most three words, City and Country are sub-automata matching city and country names respectively. Loc is a sub-automaton for the union of City and Country. All these sub-automata are assumed to be deterministic.

spanner P, the *data complexity* of all the problems is in FP: one can just materialize $P(d)$ and apply the necessary aggregate. Under combined complexity, we will therefore need to find ways to avoid materializing $P(d)$ to achieve tractability.

3 Main Results

In this section we present our main results. We present the results for spanners represented as VSet-automata, but they also hold for their regular expression counterpart. Notice that this is not trivial because translating finite automata to regular expressions can incur an unavoidable exponential blow-up [6].

Unambiguous VSet-Automata. For a number of our tractability results, it is important that the VSet-automata in the input are *unambiguous*. In order to define unambiguity, we fix a total, linear order $\prec$ on the set Γ_{Var} of variable operations, such that $x\vdash \prec \dashv x$ for every variable x. A VSet-automaton $A = (\Sigma, \text{Vars}, Q, q_0, Q_f, \delta)$ satisfies the *variable order condition* if $v \prec v'$ for every $v, v' \in \Gamma_{\text{Vars}}$ for which there are $q_1, q_2, q_3 \in Q$ such that $q_2 \in \delta(q_1, v)$ and $q_3 \in \delta(q_2, v')$. The variable order condition ensures that, for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in [\![A]\!](d)$, there is exactly one ref-word $\mathbf{r} \in \mathcal{VR}(A)$ with $\mathrm{tup}_{\mathbf{r}} = \mathrm{t}$.

A *run* of A on $\mathbf{r} = \sigma_1 \cdots \sigma_n$ is a sequence $q_0 \cdots q_n$ of states of A such that $q_i \in \delta(q_{i-1}, \sigma_i)$ for every $i \in \{1, \ldots, n\}$. It is *accepting* if $q_n \in Q_f$. A VSet-automaton A is *unambiguous*, if

1. A satisfies the variable order condition and
2. there is exactly one accepting run of A on every $\mathbf{r} \in \mathcal{R}(A)$.

In the following, we denote by uVSA the class of unambiguous VSet-automata.

▶ **Example 3.1.** The d-relation on the bottom right of Figure 1 can be extracted from d by a spanner that matches textual representations of numbers (or ranges) in the variable x_{events}, followed by a city or country name, matched in x_{loc}. Figure 2 shows how two such VSet-automata may look like. Note that some strings, like Luxembourg are the name of a city as well as a country. Thus, the upper automaton is ambiguous, because the tuple with Luxembourg is captured twice. The lower automaton is unambiguous, because the sub-automaton for Loc only matches such names once.

Table 1 Known results on the complexity of COUNT.

Aggregate	Spanner	Complexity	Reference
Count	uVSA	in FP	[2, Corollary 4.2]
Count	VSA	#P-complete	[8, Theorem 5.2] (implicit)
Count	VSA	FPRAS	[2, Corollary 4.1]

Complexity Classes. We assume familiarity with the complexity classes FP (polynomial-time computable functions), #P, $\mathrm{FP}^{\#\mathrm{P}}$, and OptP. (We provide some background in Appendix C.) Unless mentioned otherwise, we use Cook reductions, also known as Turing reductions. Notice that, under Turing reductions, #P-complete problems are also $\mathrm{FP}^{\#\mathrm{P}}$-complete.

3.1 Known Results

A number of results on COUNT are already known or easily follow from known results, see Table 1 and Theorem 3.2. Two observations can be made from this table. First, COUNT requires the input spanner to be *unambiguous* for tractability. This tractability implies that COUNT can be computed without materializing the possibly exponentially large set $P(d)$ if the spanner is unambiguous. Second, if the spanner is not unambiguous then, due to #P-completeness of COUNT, we do not know an efficient algorithm for its exact computation (and therefore may have to materialize $P(d)$), but COUNT can be *approximated* by an FPRAS. We will explore to which extent this picture generalizes to other aggregates.

▶ **Theorem 3.2** (Arenas et al. [2], Florenzano et al. [8]). *COUNT[uVSA] is in* FP *and COUNT[VSA] is* #P*-complete. Furthermore, COUNT[VSA] can be approximated by an* FPRAS*.*

3.2 Overview of New Results

Our new complexity results are summarized in Table 2. By now the reader is familiar with the aggregate problems and the types of spanners we study. In the next subsection, we will define the different representations of weight functions that we will use. Here, SINGLE are single-variable weight functions, POLY are polynomial-time computable weight functions, and REG (resp., UREG) are weight functions represented by weighted (resp., unambiguous weighted) VSet-automata. We use the following notation in the table.

▶ **Notation 3.3.** *If $\mathcal{W}$ is a class of representations of weight functions and S is a set, we denote by $\mathcal{W}_S$ the subset of $\mathcal{W}$ that represent a weight function w with $Img(w) \subseteq S$. Typical sets that we use for S are $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}^+ := \{q \in \mathbb{Q} \mid q \geq 0\}$, and $\mathbb{B} := \{0, 1\}$.*

Entries in the table should be read from left to right. For instance, the FP result in the first row states that the problems MIN, for both spanner classes uVSA and VSA, and for all three classes SINGLE, UREG and REG of weight functions is in FP. Likewise, the second row states that the same problems with $\text{POLY}_{\mathbb{N}}$ weight functions become OptP-complete and that the existence of an FPRAS would imply a collapse of the polynomial hierarchy.

In general, the table gives a detailed overview of the impact of (1) unambiguity of spanners and (2) different weight function representations on the complexity of computing aggregates.

Table 2 Detailed overview of complexities of aggregate problems for document spanners. All these results are new. By X-c we denote that the problem is complete for class X. The "no FPRAS" claims assume that the polynomial hierarchy does not collapse to the second level.

Aggregate	Spanner	Weights	Complexity
MIN	uVSA, VSA	SINGLE, UREG, REG	in FP
		$\text{POLY}_{\mathbb{N}}$	OptP-c, no FPRAS
MAX	uVSA, VSA	SINGLE, UREG	in FP
		$\text{REG}_{\mathbb{N}}$, $\text{POLY}_{\mathbb{N}}$	OptP-c, no FPRAS
SUM, AVERAGE	uVSA	SINGLE, UREG	in FP
		REG, $\text{POLY}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
	VSA	$\text{SINGLE}_{\mathbb{Q}^+}$	$\text{FP}^{\#\text{P}}$-c, FPRAS
		SINGLE, UREG, REG, $\text{POLY}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
q-QUANTILE	uVSA	SINGLE	in FP
		UREG, REG, $\text{POLY}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
	VSA	SINGLE, UREG, REG, $\text{POLY}_{\mathbb{Z}}$	$\text{FP}^{\#\text{P}}$-c, no FPRAS
q-QUANTILE (position)	VSA	POLY	Polynomial time positional approximation

3.3 Results for Different Weight Functions

We formalize how we represent the weight functions for our new results. Recall that weight functions w map pairs consisting of a document d and d-tuple t to values in $\mathbb{Q}$.

3.3.1 Single-Variable Weight Functions

The simplest type of weight functions we consider are the *single-variable* weight functions. To facilitate presentation, we assume that a designated variable x is always present in t. A *single-variable (SV)* weight function w assigns values only based on the substring selected by variable x. It is given in the input as a partial mapping μ of words to $\mathbb{Q}$ where only finitely many values are defined. The weight $w(d, \mathrm{t})$ is defined as

$$w(d,\mathrm{t}) = \begin{cases} \mu(d_{\mathrm{t}(x)}) & \text{if } d_{\mathrm{t}(x)} \text{ is in the domain of } \mu; \\ 0 & \text{otherwise.} \end{cases}$$

As we will see in Section 4, MAX[VSA, SINGLE] and MIN[VSA, SINGLE] are in FP (Corollary 4.3). Furthermore, we show that the problems SUM[$\mathcal{P}$, SINGLE], AVERAGE[$\mathcal{P}$, SINGLE], and q-QUANTILE[$\mathcal{P}$, SINGLE] behave similarly to COUNT[$\mathcal{P}$], that is, they are in FP if $\mathcal{P} = $ uVSA (Corollary 4.6) and $\text{FP}^{\#\text{P}}$-complete if $\mathcal{P} = $ VSA (Theorem 4.7).

3.3.2 Polynomial-Time Weight Functions

How far can we push our tractability results? Next, we consider more general ways of mapping d-tuples into numbers. The most general class of weight functions we consider is the set of polynomial-time weight functions (POLY). A function w from POLY is given in the input as a polynomial-time Turing Machine M that maps (d, t) pairs to values in $\mathbb{Q}$ and defines $w(d, \mathrm{t}) = M(d, \mathrm{t})$. Not surprisingly there are multiple drawbacks of having arbitrary polynomial time weight functions. The first is that all considered aggregates become intractable, even if we only consider unambiguous VSet-automata (Theorem 5.1). The second drawback is that we don't even know whether Sum and Average can be computed in $\mathrm{FP}^{\#\mathrm{P}}$ if $w \in$ Poly is a polynomial-time weight function. Therefore, we also consider two additional classes weight functions.

3.3.3 Regular Weight Functions

As the class of polynomial-time weight functions quickly leads to intractability, we focus on a restricted class that allows to examine multiple attributes, but such that we can understand the structure of the representation towards efficient algorithms. Our final classes of weight functions are based on *weighted VSet-automata* [5], which are VSet-automata that assign weights to tuples, based on a semiring. We focus on one particular semiring (the tropical semiring with min/plus) to make the presentation less abstract. One can also consider other semirings. For instance over the tropical semiring with max/plus, the complexity results are analogous to the ones we have here, with Min and Max interchanged.

To this end, let $\mathbb{Q}_\infty = \mathbb{Q} \cup \{\infty\}$. A *weighted (VSet)-automaton* is a tuple $W = (\Sigma, V, Q, I, F, \delta)$, where Σ, V, Q are as in the definition of VSet-automata, $I : Q \to \mathbb{Q}_\infty$ is the *initial weight function*; $F : Q \to \mathbb{Q}_\infty$ is the *final weight function*; and $\delta : Q \times (\Sigma \cup \Gamma_V) \times Q \to \mathbb{Q}_\infty$ is its *transition function*. Its *transitions* are the triples (p, o, q) with $\delta(p, o, q) \neq \infty$. Likewise, the *initial* (resp., *accepting*) states are those states q with $I(q) \neq \infty$ (resp., $F(q) \neq \infty$). A *run* ρ of W over a word $\mathbf{r} = \sigma_1 \cdots \sigma_n \in (\Sigma \cup \Gamma_V)^*$ is a sequence $\rho = (q_0, \sigma_1, q_1) \cdots (q_{n-1}, \sigma_n, q_n)$ of transitions. We denote the runs of W on $\mathbf{r}$ by $\mathrm{Runs}(\mathbf{r})$. The *weight* $W(\rho)$ of ρ is $I(q_0) + \delta(q_0, \sigma_1, q_1) + \cdots + \delta(q_{n-1}, \sigma_n, q_n) + F(q_n)$ and the weight $W(\mathbf{r})$ of a word $\mathbf{r}$ is $\min_{\rho \in \mathrm{Runs}(\mathbf{r})} W(\rho)$. Intuitively, a word is accepted if and only if has a finite weight. The automaton W is *unambiguous* if it satisfies conditions (1) and (2) of unambiguous VSet-automata.[1] It is *functional* if all runs are over a valid ref-word. As for ordinary VSet-automata, we also assume in this paper that weighted VSet-automata are functional.

We will consider two types of weight functions which are based on weighted VSet-automata. A *regular* (Reg) weight function w is represented by a weighted VSet-automaton W over $\Sigma \cup \Gamma_V$ and defines $w(d, \mathrm{t}) = \min_{d = \mathrm{clr}(\mathbf{r}), \mathrm{t} = \mathrm{tup}_{\mathbf{r}}} W(\mathbf{r})$. Given a document d and a d-tuple t, the weight $w(d, \mathrm{t})$ can be computed in polynomial time [5, Theorem 6.1].[2] The set of *unambiguous regular* (UReg) weight functions is the subset of Reg that is represented by unambiguous weighted VSet-automata.[3]

▶ **Example 3.4.** Figure 3 gives an unambiguous weighted VSet-automaton that extracts the values of three digit natural numbers from text. It can easily be extended to extract natural numbers of up to a constant number of digits by adding nondeterminism. Likewise, it is possible to extend it to extract weights as in Example 2.1. If a single variable captures a list of numbers, similar to $d_{[32,37\rangle} = 10{-}15$, one may use ambiguity to extract the minimal number represented in this range.

[1] Observe that the weight $W(\mathbf{r})$ of a word $\mathbf{r}$ is the weight of the run ρ which accepts $\mathbf{r}$.
[2] This tractability result requires functionality of W in the sense that computing $w(d, \mathrm{t})$ is NP-complete without it (Proposition 4.1). Making a weighted VSet-automaton functional is always possible and takes time polynomial in its number of states and exponential in its number of variables [5, Proposition 5.2].
[3] Testing whether a weighted VSet-automaton is in UReg can be done in PTIME: functionality can be

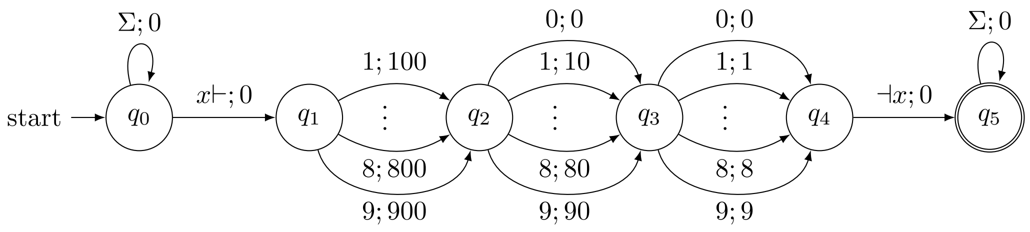

Figure 3 An unambiguous weighted VSet-automaton with initial state q_0 (with weight 0) and accepting state q_5 (with weight 0), extracting three digit natural numbers captured in variable x. Recall that the weight of a run is the sum of all its edge weights.

Our results for regular and unambiguous regular weight functions are that the situation is similar to Single when it comes to Min, Max, Sum, and Average. The main difference is that we require more unambiguity. For Max one needs unambiguity of the regular weight function and for Sum, and Average one needs unambiguity for *both* the spanner and the regular weight function to achieve tractability. For q-Quantile, the situation is different from Single in the sense that regular weight functions render the problem intractable. We refer to Table 2 for an overview.

3.4 Approximation

In the cases where exact computation of the aggregate problem is intractable, we consider the question of approximation. It turns out that there exist FPRAS's in two settings that we believe to be interesting. First, in the case of Sum and Average and single-variable weight functions, the restriction of unambiguity in the spanner can be dropped if the weight function uses only non-negative weights. Second, although q-Quantile is $\mathrm{FP}^{\#\mathrm{P}}$-complete for general VSA, it is possible to approximate the *position* of the q-quantile element in an FPRAS fashion, even with the very general polynomial-time weight functions. We discuss this problem in more detail in Section 7.

4 Single-Variable Weight Functions

We start this section by recalling that counting the number of output tuples is tractable if the spanner is functional and unambiguous (Theorem 3.2). It is well known that unambiguity is necessary in the sense that the problem becomes #P-complete without it. We next observe that functionality of the spanner is also crucial for the problem's tractability. The following proposition is heavily based on Freydenberger [9, Lemma 3.1] who showed that given a VSet-automaton A it is NP-hard to decide whether $[\![A]\!](\varepsilon) \neq \emptyset$. Based on the reduction by Freydenberger, one can also show that it the problem remains NP-hard if the VSet-automaton is unambiguous.

▶ **Proposition 4.1.** *Given a document d and non-functional VSet-automaton A, testing if $[\![A]\!](d) \neq \emptyset$ is* NP*-complete, even if A is unambiguous.*

Next, we show that Min and Max are tractable for single-variable weight functions. The reason for their tractability is that, for any fixed variable $x \in \mathrm{Vars}(A)$, the spans associated to x in output tuples can be computed in polynomial time.

tested in PTIME [5, Proposition 5.3], and testing if a functional automaton is unambiguous is also in PTIME.

▶ **Proposition 4.2.** *Let $A \in \text{VSA}$, $x \in \mathit{Vars}(A)$, and $d \in \Sigma^*$. The set $\{\mathrm{t}(x) \mid \mathrm{t} \in [\![A]\!](d)\}$ can be computed in time polynomial in the sizes of A and d.*

We immediately have:

▶ **Corollary 4.3.** *$\textsc{Min}[\text{VSA}, \textsc{Single}]$ and $\textsc{Max}[\text{VSA}, \textsc{Single}]$ are in* FP*.*

In order to calculate aggregates like Sum, Avg, or q-Quantile, it is not sufficient to know which weights are assigned, but also the multiplicity of each weight is necessary. For general VSet-automata, this immediately results in intractability, because computing these multiplicities is hard, as we show next.

▶ **Lemma 4.4.** *Let $0 < q < 1$. Then, $\textsc{Sum}[\text{VSA}, \textsc{Single}]$, $\textsc{Average}[\text{VSA}, \textsc{Single}]$, and q-$\textsc{Quantile}[\text{VSA}, \textsc{Single}]$ are $\text{FP}^{\#\text{P}}$-hard, even if w is the fixed weight function*

$$w(d, \mathrm{t}) = \begin{cases} 1 & \text{if } d_{\mathrm{t}(x)} = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Proof sketch. The lower bounds are proven by reductions from the #P-complete problem $\textsc{Count}[\text{VSA}]$. We provide a proof sketch for $\textsc{Average}$.

Let $A \in \text{VSA}$ and $d \in \Sigma^*$. We assume w.l.o.g. that $1 \notin \Sigma$. Let $d' = d \cdot 1$. We define VSet-automaton A' as A, but change two things. First, A' only produces outputs on documents of the form $s \cdot 1$, where s is an arbitrary document. It first simulates A on s and then selects, in a fresh variable x, the symbol 1. Second, on the document d', it selects a single additional tuple t with $\mathrm{t}(y) = [1, 1\rangle$ for all its variables y. More precisely, using a regular-expression-like notation, we therefore define

$$A' = (A \cdot x{\vdash} \cdot 1 \cdot {\dashv}x) \vee (x{\vdash} \cdot x_1{\vdash} \cdots x_n{\vdash} \cdot \varepsilon \cdot {\dashv}x_n \cdots {\dashv}x_1 \cdot {\dashv}x \cdot d \cdot 1).$$

Observe that, for all $\mathrm{t} \in A'(d')$, it holds that $d_{\mathrm{t}(x)} = 1$ if and only if $\mathrm{t}{\restriction}\text{Vars}(A) \in [\![A]\!](d)$. Thus, by definition of A' and w, $\text{Sum}([\![A']\!], d', w) = \text{Count}([\![A]\!], d)$ and $\text{Count}([\![A']\!], d') = \text{Count}([\![A]\!], d) + 1$. Therefore, $\text{Avg}([\![A']\!], d', w) = \frac{\text{Count}([\![A]\!], d)}{\text{Count}([\![A]\!], d)+1}$. By solving the equation for $\text{Count}([\![A]\!], d)$, it follows that $\text{Count}([\![A]\!], d) = \frac{\text{Avg}([\![A']\!], d', w)}{1-\text{Avg}([\![A']\!], d', w)}$. This concludes the proof for $\textsc{Average}$. ◀

This result shows that for general VSet-automata, the Sum, Average, and Quantile aggregates are intractable already for very simple weight functions. However, if the spanner is *unambiguous*, we can achieve tractability. The reason is that we can compute in polynomial time the multiset $\mathbb{T}_{A,d} = \{\!\{\mathrm{t}(x) \mid \mathrm{t} \in [\![A]\!](d)\}\!\}$, where we represent the multiplicity of each span $[i, j\rangle$ (the number of tuples $\mathrm{t} \in P(d)$ such that $\mathrm{t}(x) = [i, j\rangle$) in binary.

▶ **Lemma 4.5.** *Given a VSet-automaton A and a document d, the multiset $\mathbb{T}_{A,d}$ can be computed in* FP *if $A \in \text{uVSA}$ and in $\text{FP}^{\#\text{P}}$ if $A \in \text{VSA}$.*

It follows that all remaining aggregate functions can be efficiently computed if the spanner is given as an unambiguity functional VSet-automaton and in $\text{FP}^{\#\text{P}}$ otherwise.

▶ **Corollary 4.6.** *$\textsc{Sum}[\text{uVSA}, \textsc{Single}]$, $\textsc{Average}[\text{uVSA}, \textsc{Single}]$, and q-$\textsc{Quantile}[\text{uVSA}, \textsc{Single}]$ are in* FP*, for every $0 \leq q \leq 1$.*

The following theorem follows directly from Lemma 4.4 and Lemma 4.5.

▶ **Theorem 4.7.** *Let $0 < q < 1$. Then,* $\textsc{Sum}[\mathrm{VSA}, \textsc{Single}]$*,* $\textsc{Average}[\mathrm{VSA}, \textsc{Single}]$*, and* $q\text{-}\textsc{Quantile}[\mathrm{VSA}, \textsc{Single}_{\mathbb{B}}]$ *are* $\mathrm{FP}^{\#\mathrm{P}}$*-complete, even if w is the fixed weight function*

$$w(d, \mathrm{t}) = \begin{cases} 1 & \text{if } d_{\mathrm{t}(x)} = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we note that all tractability results in this section continue to hold for weight functions with a constant number of variables, i.e., when the weight functions are given as mappings μ of k-tuples of words to $\mathbb{Q}$, where k is a constant. We will provide proofs in the full version of the paper.

5 Polynomial-Time Weight Functions

Before we study regular weight functions, we make a few observations on the very general polynomial-time computable weight functions. For weight functions $w \in \textsc{Poly}$, we assume that the number $w(d, \mathrm{t})$ is a rational number represented by its numerator and dominator, and that the function w is represented as a Turing Machine A that returns a value $A(d, \mathrm{t})$ in polynomially many steps for some fixed polynomial of choice (e.g., n^2).[4] Furthermore, to avoid any complexity due to the need to verify whether A is indeed a valid input (i.e., timely termination), we will assume that $w(d, \mathrm{t}) = 0$, if A does not produce a value within the allocated time.

We first observe that polynomial-time weight functions make all our aggregation problems intractable, which is not surprising.

▶ **Theorem 5.1.** $\textsc{Min}[\mathrm{uVSA}, \textsc{Poly}]$ *and* $\textsc{Max}[\mathrm{uVSA}, \textsc{Poly}]$ *are* OptP*-hard. Furthermore,* $\textsc{Sum}[\mathrm{uVSA}, \textsc{Poly}]$*,* $\textsc{Average}[\mathrm{uVSA}, \textsc{Poly}]$*, and* $q\text{-}\textsc{Quantile}[\mathrm{uVSA}, \textsc{Poly}]$ *are* $\mathrm{FP}^{\#\mathrm{P}}$*-hard.*

In fact, all but the lower bound for Min already hold for regular weight functions (Theorems 6.3, 6.5 and 6.6). Min becomes tractable for regular weight functions, but it can be shown that Min is OptP-hard for weight functions represented by weighted VSet-automata over the *numeric semiring*. Therefore, we do not require powerful weight functions for the hardness proof of Min. Furthermore, we are able to provide OptP and $\mathrm{FP}^{\#\mathrm{P}}$ upper bounds if the weight functions return natural numbers (or integers in the case of the $\mathrm{FP}^{\#\mathrm{P}}$ upper bounds).

▶ **Theorem 5.2.** $\textsc{Min}[\mathrm{VSA}, \textsc{Poly}_{\mathbb{N}}]$ *and* $\textsc{Max}[\mathrm{VSA}, \textsc{Poly}_{\mathbb{N}}]$ *are in* OptP*.*

▶ **Theorem 5.3.** $\textsc{Sum}[\mathrm{VSA}, \textsc{Poly}_{\mathbb{Z}}]$*,* $\textsc{Average}[\mathrm{VSA}, \textsc{Poly}_{\mathbb{Z}}]$*, and* $q\text{-}\textsc{Quantile}[\mathrm{VSA}, \textsc{Poly}_{\mathbb{Z}}]$ *are in* $\mathrm{FP}^{\#\mathrm{P}}$*, for every $0 < q < 1$.*

6 Regular Weight Functions

We now turn to Reg and UReg weight functions. We first observe that Single weight functions can be translated in polynomial time to equivalent UReg weight functions by a simple automata construction. So, all lower bounds for Single also hold for UReg.

We show that aggregation problems for regular weight functions can often be reduced to problems about paths on weighted DAGs, where the weights come from the semiring of the weight function. To this end, a *weighted DAG* is a DAG $D = (V, E, \ell, src, snk)$, where

[4] Our complexity results are independent of the choice of this polynomial.

$\ell : E \to \mathbb{Q}_\infty$ associates a *weight* or *length* to each edge and *src* (resp., *snk*) is a unique node in V without incoming (resp., outgoing) edges. We define paths p in the obvious manner as sequences of edges and the length $\ell(p)$ of p as the sum of the lengths of its edges.

▶ **Lemma 6.1.** *Let d be a document, $A \in \text{VSA}$ and W be a weighted VSet-automaton representing $w \in \textsc{Reg}$. Then we can compute, in polynomial time, a weighted DAG D such that there is a surjective mapping m from paths p from src to snk in D to tuples $\mathrm{t} \in [\![A]\!](d)$. Furthermore,*

1. *if A and W are unambiguous, then m is a bijection and*
2. *for every $\mathrm{t} \in [\![A]\!](d)$ we have $w(d,\mathrm{t}) = \min_{\{p \mid m(p)=\mathrm{t}\}} \ell(p)$.*

Proof sketch. The DAG D is obtained by a product construction between A, W, and d, such that every path from *src* to *snk* corresponds to an accepting run of W on some ref-word that represents a tuple in $[\![A]\!](d)$. This correspondence ensures that m is a bijection if A and W are unambiguous. ◀

The weighted DAG from Lemma 6.1 plays the role of a compact representation of the materialized intermediate result. It allows us to reduce Min to the shortest path problem in DAGs. If the weight function is unambiguous, Max can be reduced to the longest path problem in DAGs. Notice that, although the longest path problem is intractable in general, it is tractable for DAGs.

▶ **Theorem 6.2.** *Min[VSA, Reg] and Max[VSA, UReg] are in* FP*.*

The result for Max is close to the tractability frontier: if we relax the unambiguity condition in the weight function, the problem doesn't correspond to finding the longest paths in DAGs anymore and becomes intractable. In the following theorem, we restrict weight functions to natural numbers, because then we can show completeness for OptP, which is a class of functions that return natural numbers. Allowing positive and negative numbers does not fundamentally change the complexity of the problems though.

▶ **Theorem 6.3.** *Max[uVSA, $\textsc{Reg}_{\mathbb{N}}$] is* OptP*-complete.*

Since Sum and Average are already $\text{FP}^{\#\text{P}}$-hard for VSA spanners and Single weight functions (Theorem 4.7), they are $\text{FP}^{\#\text{P}}$-hard for VSA spanners and Reg/UReg weight functions as well. However, in a similar vein as in Section 4, the problems become tractable if we have unambiguity. Here, however, we require unambiguity of *both* the spanner and the representation of the weight function.

▶ **Theorem 6.4.** *Sum[uVSA, UReg] and Average[uVSA, UReg] are in* FP*.*

Proof sketch. Due to Lemma 6.1, these problems boil down to computing the sum of the lengths of source-to-target paths in a DAG and the average length of source-to-target paths in a DAG, respectively. Concerning Sum, we can count, for each individual edge in the DAG, the number of paths that use this edge. The sum of all output tuples is obtained by multiplying these values with the length of the edge and taking the sum over all edges. The tractability of Average then follows from the tractability of Sum and of counting the number of source-to-target paths in a DAG. ◀

Indeed, if we relax the restriction that weight functions are given as unambiguous automata, Sum and Average become $\text{FP}^{\#\text{P}}$-hard again.

▶ **Theorem 6.5.** *Sum[uVSA, Reg] and Average[uVSA, Reg] are* $\text{FP}^{\#\text{P}}$*-complete.*

The situation for q-Quantile is different from Max, Sum, and Average, since it remains hard even when both the spanner and weight function are unambiguous. The reason is that the problem reduces to counting the number of paths in a weighted DAG that are shorter than a given target weight, which is #P-complete due to Mihalak et al. [23].

▶ **Theorem 6.6.** *q-Quantile[uVSA, UReg] is $\mathrm{FP}^{\#\mathrm{P}}$-complete, for every $0 < q < 1$.*

Proof sketch. The upper bound is immediate from Theorem 6.7. For the lower bound, at the core of the quantile problem is the problem of counting up to a threshold k:

$$\textsc{Count}_{\leq k}(P, d, w) := |\{\mathrm{t} \in P(d) \mid w(d, \mathrm{t}) \leq k\}|.$$

The corresponding problem $\textsc{Count}_{\leq k}[\mathcal{P}, \mathcal{W}]$ is defined analogously to $\textsc{Sum}[\mathcal{P}, \mathcal{W}]$. It can be shown that $\textsc{Count}_{\leq k}[\mathcal{P}, \mathcal{W}]$ is #P-hard – using a reduction from #Partition, similar to Mihalak et al. [23, Theorem 1]. Using a binary search argument, $\textsc{Count}_{\leq k}[\mathrm{uVSA}, \mathrm{UReg}]$ can be reduced to q-Quantile[uVSA, UReg], concluding the proof. ◀

Finally, we show that Sum, Average, and q-Quantile for Reg weight functions are in $\mathrm{FP}^{\#\mathrm{P}}$.

▶ **Theorem 6.7.** *Sum[VSA, Reg], Average[VSA, Reg], and q-Quantile[VSA, Reg] are in $\mathrm{FP}^{\#\mathrm{P}}$, for every $0 < q < 1$.*

7 Aggregate Approximation

Now that we have a detailed understanding on the complexity of computing exact aggregates, we want to see in which cases the result can be approximated. We only consider the situation where the exact problems are intractable and want to understand when the considered aggregation problems can be approximated by fully polynomial randomized approximation schemes (FPRAS), and when the existence of such an FPRAS would imply a collapse of the polynomial hierarchy.

▶ **Definition 7.1.** *Let f be a function that maps inputs x to rational numbers and let $\mathcal{A}$ be a probabilistic algorithm, which takes an input instance x and a parameter $\delta > 0$. Then $\mathcal{A}$ is called a* fully polynomial randomized approximation scheme (FPRAS)*, if*

- $Pr\Big(\big|\mathcal{A}(x, \delta) - f(x)\big| \ \leq \ \delta \cdot \big|f(x)\big|\Big) \ \geq \ \frac{3}{4}$;
- *the runtime of $\mathcal{A}$ is polynomial in $|x|$ and $\frac{1}{\delta}$.*

7.1 Approximation is Hard at First Sight

For the problems Min, Max with Poly weight functions, the existence of an FPRAS would imply a collapse of the polynomial hierarchy, even when spanners are unambiguous. In the Appendix we show that the lower bound for Min already holds for weight functions represented by weighted VSet-automata over the numeric semiring.

▶ **Theorem 7.2.** *Min[uVSA, $\mathrm{Poly}_{\mathbb{N}}$] and Max[uVSA, $\mathrm{Reg}_{\mathbb{N}}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Approximation of Sum and Average is already hard for single variable weight functions. It is crucial for the hardness, however, that the weight functions can output positive and negative numbers.

▶ **Theorem 7.3.** *$\textsc{Sum}$[VSA, $\textsc{Single}_{\{-1,1\}}$] and $\textsc{Average}$[VSA, $\textsc{Single}_{\{-1,1\}}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

If the spanners are unambiguous, the simplest intractable setting for $\textsc{Sum}$ and $\textsc{Average}$ is the one with $\textsc{Reg}$ weight functions (see Table 2). Also here, the existence of an FPRAS implies a collapse of the polynomial hierarchy.

▶ **Theorem 7.4.** *$\textsc{Sum}$[uVSA, $\textsc{Reg}$] and $\textsc{Average}$[uVSA, $\textsc{Reg}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

We now turn to the quantile problem. It turns out that this problem is difficult to approximate even if the weight functions only return 0 or 1.

▶ **Theorem 7.5.** *Let $0 < q < 1$. Then, q-$\textsc{Quantile}$[VSA, $\textsc{Single}_{\mathbb{B}}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses on the second level.*

When the spanners are unambiguous, the simplest intractable case for q-$\textsc{Quantile}$ is the one with $\textsc{UReg}$ weight functions (see Table 2). Again, we can show that approximation is hard.

▶ **Theorem 7.6.** *Let $0 < q < 1$. Then, q-$\textsc{Quantile}$[uVSA, $\textsc{UReg}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses on the second level.*

7.2 When an FPRAS is Possible

We show that Theorem 7.3 is very much on the tractability frontier: it shows that approximation is intractable if weight functions can assign 1 and -1. On the other hand, if the weight functions are restricted to *non-negative* numbers, then approximating $\textsc{Sum}$ and $\textsc{Average}$ is possible with an FPRAS.

▶ **Theorem 7.7.** *$\textsc{Sum}$[VSA, $\textsc{Single}_{\mathbb{Q}_+}$] and $\textsc{Average}$[VSA, $\textsc{Single}_{\mathbb{Q}_+}$] can be approximated by an* FPRAS*.*

Our second positive result is about approximating quantiles *in a positional manner*. To this end, let d be a document, P be a document spanner, w be a weight function and $0 \leq q \leq 1$ with $q \in \mathbb{Q}$. Then, for any $\delta > 0$, we say that $k \in \mathbb{Q}$ is a positional δ-approximation of q-Quantile(P, d, w) if there is a $q' \in \mathbb{Q}$, with $q - \delta \leq q' \leq q + \delta$ and $k = q'$-Quantile(P, d, w).[5]

▶ **Theorem 7.8.** *Let $0 \leq q \leq 1$. There is a probabilistic algorithm that calculates a positional δ-approximation of q-$\textsc{Quantile}$[VSA, $\textsc{Poly}$] with success probability at least $\frac{3}{4}$. Furthermore, the run time of the algorithm is polynomial in the input and $\frac{1}{\delta}$.*

Proof sketch. Arenas et al. [2, Corollary 4.1] showed that given a functional VSet-automaton A, one can sample tuples $\mathrm{t} \in [\![A]\!](d)$ uniformly at random with success probability at least $\frac{1}{2}$. This algorithm can be used to create a sample of $[\![A]\!](d)$ and return the q-Quantile of the sample. We show that a sample of $s \geq \frac{\ln(16)}{2\delta^2}$ tuples is sufficient to ensure that the returned quantile is indeed a positional δ-approximation of the quantile with success probability at least $\frac{7}{8}$. ◀

[5] The idea of positional quantile approximations was originally introduced by Manku et al. [19] in the context of quantile computations with limited memory.

8 Concluding Remarks

We investigated the computational complexity of common aggregate functions over regular document spanners given as regex formulas and VSet-automata. While each of the studied aggregate functions is intractable in the general case, there are polynomial-time algorithms under certain general assumptions. These include the assumption that the numerical value of the tuples is determined by a fixed variable, or that the spanner is represented as an unambiguous VSet-automaton, or the conjunction of the two assumptions. Moreover, we established quite general tractability results when randomized approximations (FPRAS) are possible. The upper bounds that we obtained for general (functional) VSet-automata immediately generalize to aggregate functions over queries that involve relational-algebra operators and string-equality conditions on top of spanners, whenever these inner queries can be *efficiently* compiled into a single VSet-automaton [10, 26]. Moreover, these upper bounds immediately generalize to allow for *grouping* (i.e., the GROUP BY operator) by computing the tuples of the grouping variables and applying the algorithms to each group separately.

We identified several interesting cases where the computation of $\alpha(P(d))$ can avoid the materialization of the exponentially large set $P(d)$, where, d is the document, P is the spanner, and α is the aggregate function. Notably, this is the case (1) for Min with general VSet-spanners and weight functions in Reg, UReg, and Single, (2) for Max with general VSet-spanners and weight functions in UReg and Single, (3) for Sum and Average with uVSA-spanners and weight functions in UReg and Single, and (4) for q-Quantile with uVSA-spanners and Single weight functions.

Yet, several basic questions are left for future investigation. A natural next step would be to seek additional useful assumptions that cast the aggregate queries tractable: Can monotonicity properties of the numerical functions lead to efficient algorithms in cases that are otherwise intractable? What are the regex formulas that can be efficiently translated into unambiguous VSet-automata (and, hence, allow to leverage the algorithms for such VSet-automata)? Another important direction is to generalize our results in a more abstract framework, such as the *Functional Aggregate Queries* (FAQ) [13], in order to provide a uniform explanation of our findings and encompass general families of aggregate functions rather than specific ones. Finally, the practical side of our work remains to be studied: How do we make our algorithms efficient in practice? How effective is the sampling approach in terms of the balancing between accuracy and execution cost? Can we accurately compute estimators of aggregate functions over (joins of) spanners within the setting of *online aggregation* [12, 16]?

References

1 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In Pablo Barceló and Marco Calautti, editors, *ICDT*, volume 127 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.ICDT.2019.22`.

2 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *PODS*, pages 59–73. ACM, 2019. `doi:10.1145/3294052.3319704`.

3 K. Y. Cockwell and I. G. Giles. Software tools for motif and pattern scanning: program descriptions including a universal sequence reading algorithm. *Computer Applications in the Biosciences*, 5(3):227–232, 1989.

4 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *PODS*, pages 149–163, 2019. doi:10.1145/3294052.3319684.

5 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *ICDT*, pages 8:1–8:18, 2020.

6 A. Ehrenfeucht and H. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

7 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *Journal of the ACM*, 62(2):12:1–12:51, 2015. doi:10.1145/2699442.

8 Fernando Florenzano, Cristian Riveros, Martin Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, page 165–177. Association for Computing Machinery, 2018. doi:10.1145/3196959.3196987.

9 Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019. doi:10.1007/s00224-018-9874-1.

10 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, pages 137–149, 2018.

11 Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In Carsten Lutz and Jean Christoph Jung, editors, *ICDT*, volume 155 of *LIPIcs*, pages 11:1–11:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICDT.2020.11.

12 Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298. ACM Press, 1999.

13 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In Tova Milo and Wang-Chiew Tan, editors, *PODS*, pages 13–28. ACM, 2016. doi:10.1145/2902251.2902280.

14 Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988. doi:10.1016/0022-0000(88)90039-6.

15 Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. SystemT: A system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008. doi:10.1145/1519103.1519105.

16 Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *SIGMOD Conference*, pages 615–629. ACM, 2016.

17 Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2004.

18 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, pages 109–114. ACL, 2011.

19 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD Conference*, page 426–435. Association for Computing Machinery, 1998. doi:10.1145/276304.276342.

20 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, pages 125–136, 2018.

21 Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In *CD-MAKE*, volume 11015 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2018.

22 Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. In *ICLR (Poster)*. OpenReview.net, 2019.

23 Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016. doi:10.1007/s00224-014-9571-7.

24 A. Neuwald and P. Green. Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712, 1994.

25 Galia Nordon, Gideon Koren, Varda Shalev, Benny Kimelfeld, Uri Shalit, and Kira Radinsky. Building causal graphs from medical literature and electronic medical records. In *AAAI*, pages 1102–1109. AAAI Press, 2019.

26 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *PODS*, pages 320–334. ACM, 2019. doi:10.1145/3294052.3319699.

27 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *ICDT*, volume 127 of *LIPIcs*, pages 13:1–13:18, 2019.

28 Hoifung Poon and Pedro M. Domingos. Joint inference in information extraction. In *AAAI*, pages 913–918. AAAI Press, 2007.

29 Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

30 Robert P. Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Trans. Inf. Syst.*, 27(2):12:1–12:19, 2009. doi:10.1145/1462198.1462204.

31 Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11):1310–1321, 2015. doi:10.14778/2809974.2809991.

32 Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

33 Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. MIT Press, Cambridge, MA, USA, 1991.

34 Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *ICML*, volume 80, pages 5244–5253. JMLR.org, 2018.

A Algebraic Operations on Spanners

We will now recall some definitions of algebraic operations on spanners. To this end, we start with some basic definitions. Two d-tuples t_1 and t_2 are *compatible* if they agree on every common variable, i.e., $\mathrm{t}_1(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1) \cap \mathrm{Vars}(\mathrm{t}_2)$. In this case, define $\mathrm{t}_1 \bowtie \mathrm{t}_2$ as the tuple with $\mathrm{Vars}(\mathrm{t}_1 \bowtie \mathrm{t}_2) = \mathrm{Vars}(\mathrm{t}_1) \cup \mathrm{Vars}(\mathrm{t}_2)$ such that $(\mathrm{t}_1 \bowtie \mathrm{t}_2)(x) = \mathrm{t}_1(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1)$ and $(\mathrm{t}_1 \bowtie \mathrm{t}_2)(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_2)$. Note that the following operators are the same as those defined by Fagin et al [7].

▶ **Definition A.1** (Algebraic Operations on Spanners). *Let P, P_1, P_2 be spanners and let $d \in \Sigma^*$ be a document.*

- Union. *The union $P = P_1 \cup P_2$ is defined when $Vars(P_1) = Vars(P_2)$. In that case, $P(d) = P_1(d) \cup P_2(d)$.*
- Projection. *The projection $P = \pi_Y P_1$ is defined by $P(d) = \{\mathrm{t}{\restriction}Y \mid \mathrm{t} \in P_1(d)\}$. Recall that $\mathrm{t}{\restriction}Y$ denotes the restriction of t to the variables in Y.*
- Natural Join. *The (natural) join $P = P_1 \bowtie P_2$ is defined such that $P(d)$ consists of all tuples $\mathrm{t}_1 \bowtie \mathrm{t}_2$ such that $\mathrm{t}_1 \in P_1(d)$, $\mathrm{t}_2 \in P_2(d)$, and t_1 and t_2 are compatible.*

B Regex-formulas

A *regex-formula (over* Σ*)* is a regular expression that may include variables (called capture variables). Formally, we define the syntax with the recursive rule

$$\alpha := \emptyset \mid \varepsilon \mid \sigma \mid (\alpha \vee \alpha) \mid (\alpha \cdot \alpha) \mid \alpha^* \mid x\vdash \alpha \dashv x\,,$$

where $\sigma \in \Sigma$ and $x \in \mathrm{Vars}$. We use α^+ as a shorthand for $\alpha \cdot \alpha^*$ and Σ as a shorthand for $\bigvee_{\sigma\in\Sigma} \sigma$. The set of variables that occur in α is denoted by $\mathrm{Vars}(\alpha)$ and the size $|\alpha|$ is defined as the number of symbols in α.

Every regex-formula can be interpreted as a generator of a (regular) ref-word language $\mathcal{R}(\alpha)$ over the extended alphabet $\Sigma \cup \Gamma_{\mathrm{Vars}(\alpha)}$. If α is of the form $x\vdash \beta \dashv x$, then $\mathcal{R}(\alpha) := \{x\vdash\} \cdot \mathcal{R}(\beta) \cdot \{\dashv x\}$. Otherwise, $\mathcal{R}(\alpha)$ is defined as the language $\mathcal{L}(\alpha)$, that is $\mathcal{R}(\emptyset) := \emptyset$, $\mathcal{R}(a) := \{a\}$ for every $a \in \Sigma \cup \{\varepsilon\}$, $\mathcal{R}(\alpha \vee \beta) := \mathcal{R}(\alpha) \cup \mathcal{R}(\beta)$, $\mathcal{R}(\alpha \cdot \beta) := \mathcal{R}(\alpha) \cdot \mathcal{R}(\beta)$, $\mathcal{R}(\alpha^*) := \{\mathcal{R}(\alpha)^i \mid i \geq 0\}$.

Notice that $\mathcal{R}(\alpha)$ can contain ref-words in which the same variable is used multiple times. By $\mathcal{VR}(\alpha)$ we denote the set of ref-words in $\mathcal{R}(\alpha)$ that are valid and by $\mathcal{VR}_{\mathrm{Vars}(\alpha)}(\alpha)$ we denote the set of ref-words in $\mathcal{R}(\alpha)$ that are valid for $\mathrm{Vars}(\alpha)$. For example, if $\alpha = (x\vdash a \dashv x)^*$, then $\mathcal{VR}(\alpha) = \{\varepsilon, x\vdash a \dashv x\}$ and $\mathcal{VR}_{\mathrm{Vars}(\alpha)}(\alpha) = \{x\vdash a \dashv x\}$. For every document $d \in \Sigma^*$, we define $\mathcal{VR}(\alpha, d) := \mathcal{VR}(\alpha) \cap \mathcal{VR}(d)$. In other words, $\mathcal{VR}(\alpha, d)$ contains exactly those valid ref-words from $\mathcal{VR}(\alpha)$ that clr maps to d. Finally, the spanner $[\![\alpha]\!]$ is the one that maps every document $d \in \Sigma^*$ to the following set of tuples:

$$[\![\alpha]\!](d) := \{\mathrm{tup}_{\mathbf{r}} \mid \mathbf{r} \in \mathcal{VR}(\alpha, d)\}$$

We will sometimes denote the set of tuples $[\![\alpha]\!](d)$ by $\alpha(d)$ to simplify notation. We say that a regex-formula is *functional* if $\mathcal{R}(\alpha) = \mathcal{VR}_{\mathrm{Vars}(\alpha)}(\alpha)$, that is, every ref-word in $\mathcal{R}(\alpha)$ is valid for $\mathrm{Vars}(\alpha)$. As for VSet-automata, in this paper, we assume that regex-formulas are functional. The set of all functional regex-formulas is denoted by RGX.

C Some Background on Complexity

We will recall the definitions for some of the complexity classes we will use in the following sections, closely following [33]. The class FP is the set of all functions that are computable in polynomial time. A *counting Turing Machine* is an non-deterministic Turing Machine whose output for a given input is the number of accepting computations for that input. The class #P is the set of all functions that are computable by polynomial-time counting Turing Machines. A problem X is #P*-hard* if there are polynomial time Turing reductions to it from all problems in #P. If in addition $X \in$ #P, we say that X is #P-complete. The class $\mathrm{FP}^{\#\mathrm{P}}$ is the set of all functions that are computable in polynomial time by an oracle Turing Machine with an #P oracle. It is easy to see that, under Turing reductions, a problem is hard for the class #P if and only if it is hard for $\mathrm{FP}^{\#\mathrm{P}}$. For counting problems, use Cook reductions, also known as Turing reductions. Under these reductions, counting the number of satisfying assignments of a DNF formula or the number of words accepted by some NFA is #P-complete.

The class OptP is the set of all functions computable by taking the maximum output values over all accepting computations of a polynomial-time non-deterministic Turing Machine that outputs natural numbers. Assume that Γ is the Turing Machine alphabet. Let $f, g : \Gamma^* \to \mathbb{N}$

be functions. A *metric reduction*, as introduced by Krentel [14], from f to g is a pair of polynomial-time computable functions T_1, T_2, where $T_1 : \Gamma^* \to \Gamma^*$ and $T_2 : \Gamma^* \times \mathbb{N} \to \mathbb{N}$, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Gamma^*$.

The class BPP is the set of all decision problems solvable in polynomial time by a probabilistic Turing Machine in which the answer always has probability at least $\frac{1}{2} + \delta$ of being correct for some fixed $\delta > 0$.