Skyline Operators for Document Spanners

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

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

Sébastien Labbé
École normale supérieure, Paris, France

Stefan Mengel
Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

Abstract

When extracting a relation of spans (intervals) from a text document, a common practice is to filter out tuples of the relation that are deemed dominated by others. The domination rule is defined as a partial order that varies along different systems and tasks. For example, we may state that a tuple is dominated by tuples that extend it by assigning additional attributes, or assigning larger intervals. The result of filtering the relation would then be the *skyline* according to this partial order. As this filtering may remove most of the extracted tuples, we study whether we can improve the performance of the extraction by compiling the domination rule into the extractor.

To this aim, we introduce the *skyline operator* for declarative information extraction tasks expressed as document spanners. We show that this operator can be expressed via regular operations when the domination partial order can itself be expressed as a regular spanner, which covers several natural domination rules. Yet, we show that the skyline operator incurs a computational cost (under combined complexity). First, there are cases where the operator requires an exponential blowup on the number of states needed to represent the spanner as a sequential variable-set automaton. Second, the evaluation may become computationally hard. Our analysis more precisely identifies classes of domination rules for which the combined complexity is tractable or intractable.

2012 ACM Subject Classification Information systems → Information extraction; Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Information Extraction, Document Spanners, Query Evaluation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.7

Related Version *Full Version (with all Proofs)*: `https://arxiv.org/abs/2304.06155` [4]

Funding The work of Amarilli, Labbé, and Mengel were partially supported by the ANR project EQUUS ANR-19-CE48-0019. The work of Kimelfeld was supported by the Israel Science Foundation (ISF) Grant 768/19.

1 Introduction

The framework of *document spanners* [10] is an established formalism to express declarative information extraction tasks. A *spanner* specifies the possible ways to assign variables over a textual document, producing so-called *mappings* which are the result of the extraction: each mapping assigns the variables to a factor of the document, called a *span*. The spanner formalism has been defined in terms of several operators, in particular regular operations extended with capture variables (corresponding to so-called *regular spanners*), operators from relational algebra (which can sometimes be translated into regular expressions), string equality (the so-called *core spanners*), etc.

Tuples extracted from text often aim to capture mentions of real-life entities and relationships. In that respect, one of the studied challenges is that different extracted tuples might be considered as conflicting with each other [11]. A common example is that of overlapping spans; for instance, a situation where one entity mention is contained within another entity mention is considered inconsistent. For this reason, traditional declarative systems for information extraction provide explicit mechanisms for restricting the extracted spans to the *maximal* ones according to different comparisons. IBM's SystemT [17] has the *consolidation* rules such as "contained-within" (where a span dominates its subspans) and "left-to-right" (where a span dominates all shorter spans that begin at the same position). Similarly, the GATE system [9] features *controls* such as "Appelt" (which is similar to SystemT's "left-to-right"). Alternatively, in the *schemaless* context of document spanners where we can assign spans to only a subset of variables [18], we may want to only capture spans that assign a maximal subset of the variables and cannot be extended by assigning more variables; in the spirit, for instance, of the relational *full disjunction* [14] or the `OPTIONAL` operator of SPARQL [2].

To explore the expressive power of operators such as controls and consolidators, Fagin et al. [11] proposed a framework that enriches document spanners with a previous concept of *prioritized repairs* [26]. There, they defined the notion of a "denial preference-generating dependency" (denial pgd) that expresses the binary domination relationship using the underlying spanner language. When this relationship is transitive, the result of applying the denial pgd is precisely the set of maximal tuples. However, they did not address the computational complexity of this operator and, consequently, it has been left open. (Moreover, their study does not apply to the schemaless context.)

The notion of maximal matches has been abundantly studied in other areas of database research, where it is called the *skyline operator* [5]. Intuitively, the skyline of a set of results under a partial order relation is the set of the results that are maximal, i.e., are not dominated by another result. The complexity of skyline computation has been investigated under many dimensions, e.g., I/O access [25], parallel computation [1], or noisy comparisons [16]. However, we are not aware of a study of the complexity of this operator to extract the maximal matches of document spanners. This is the focus of the present paper.

Contributions. We present our contributions together with the structure of the paper. After some necessary preliminaries (Section 2), we first introduce in Section 3 the skyline operator. The operator is defined as extracting the maximal mappings of a spanner on a document with respect to a partial order on the mappings, which we call a *domination relation*. In particular, we define the *span inclusion*, *span length*, *variable inclusion*, and *left-to-right domination relations*, which cover the examples presented above.

To allow for a unified study of these operators, and similarly to [11], we propose a general model where the domination relations are themselves expressed as document spanners. More precisely, a *domination rule* is a spanner that defines a domination relation on every document: it indicates which mappings dominate which other mappings, by intuitively capturing pairs (m, m') that indicate that m' dominates m. We also focus on so-called *variable-wise rules*, where the domination relation on mappings can be defined as a product of relations on spans. In other words, a variable-wise rule is a spanner expressing which spans dominate which spans, and the domination relation on mappings is obtained in a pointwise fashion across the variables, like the *ceteris paribus* semantics for preference handling in artificial intelligence [6] or Pareto-optimal points for skyline queries on multidimensional data [16]. All examples introduced earlier can be expressed in this variable-wise way.

We then begin our study of how to evaluate the skyline operator on document spanners, and start in Section 4 with the question of *expressiveness*: does the operator strictly increase the expressive power of spanner formalisms, or can it be rewritten using existing operators? We show that *regular spanners* are closed under the skyline operator, generalizing a result of [11] to the schemaless context. By contrast, we show that *core spanners* are not closed under skylines, even for the fixed variable inclusion or span inclusion domination relations, again generalizing a result of [11].

Next, we explore the question of whether it is possible to tractably rewrite the skyline operator into regular spanners, to allow for efficient evaluation like, e.g., the polynomial-time compilation of the join operator in the schema-based context (see [19], Lemma 4.4.7). We present in Section 5 a lower bound establishing that this is not the case: even for variable inclusion domination, applying the skyline operator to a spanner expressed as a sequential variable-set automaton (VA) incurs a necessary exponential blowup. This result is shown by identifying a connection between VAs and *nondeterministic read-once branching programs* (NROBPs). This general-purpose method can be used outside of the context of skylines, and in fact we also use it to show a result of independent interest: there are regex-formulas on which the *natural join* operator incurs an unavoidable exponential blowup (Theorem 5.5).

We then move in Section 6 from state complexity to the *computational complexity* of skyline evaluation for regular spanners. This task is clearly tractable in *data complexity*, i.e., for a fixed spanner and domination rule: we simply compute all captured mappings, and filter out the non-maximal ones. More interestingly, assuming $\mathsf{P} \neq \mathsf{NP}$, we show that the task is intractable in *combined complexity*, i.e., as a function of the input spanner (Theorem 6.3), already in the case of the variable inclusion relation. Hence, we cannot tractably evaluate the skyline operator in combined complexity, even without compiling it to an explicit VA.

Lastly, we study in more detail how the complexity of skyline computation depends on the fixed domination relation: are there non-trivial domination rules for which skyline computation is tractable in combined complexity? We show in Section 7 a sufficient condition on domination rules which is satisfied by all example rules that we mentioned and which implies (conditional) intractability (Theorem 7.5). We then show that, for a class of domination rules called *variable-inclusion-like* rules, a variant of this condition can be used for a dichotomy to classify which of these rules enjoy tractable skyline computation (Theorem 7.7). We finish with examples of tractable and intractable rules in the general case.

We conclude in Section 8. For reasons of space, most proofs are deferred to the full version [4].

2 Preliminaries

Languages, spans, mappings, and spanners. We fix an *alphabet* Σ which is a finite set of letters. A *word* w is a finite sequence of letters of Σ: we write Σ^* for the set of all words. We write $|w|$ for the length of w and denote the empty word by ε, with $|\varepsilon| = 0$. A *language* $L \subseteq \Sigma^*$ is a set of words. The *concatenation* of two languages L_1 and L_2 is the language $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. The *Kleene star* of a language L is the language $L^* = \bigcup_{i \in \mathbb{N}} L^i$, where we define inductively $L^0 = \{\varepsilon\}$ and $L^{i+1} = L \cdot L^i$ for all $i > 0$. As usual in the context of document spanners, a *document* is simply a word of Σ^*.

A *span* $[i, j\rangle$ is an interval $s = [i, j\rangle$ with $0 \leqslant i \leqslant j$. Its *length* is $j - i$. We denote by Spans the set of all spans. The *spans* of a document d are the spans $[i, j\rangle$ of Spans with $j \leqslant |d|$. We write $d_{[i,j\rangle}$ to mean the contiguous subword of d at a span $[i, j\rangle$, for example "qwertyqwerty"$_{[2,5\rangle}$ = "qwertyqwerty"$_{[8,11\rangle}$ = "ert". Note that we have $d_{[i,i\rangle} = \varepsilon$ for all $0 \leqslant i \leqslant |d|$. A span $[i, j\rangle$ is *included* in a span $[i', j'\rangle$ if $i' \leqslant i$ and $j' \geqslant j$. Two spans *overlap* if there is a non-empty span included in both of them; otherwise we call them *disjoint*.

We fix an infinite set $\mathsf{Variables}$ of variable names. A *mapping* m of a document $d \in \Sigma^*$ is a function from a finite set of variables $X \subseteq \mathsf{Variables}$, called the *domain* $\mathsf{dom}(m)$ of m, to the set of spans of d; the variables of $\mathsf{dom}(m)$ are said to be *assigned* by m. We denote the set of all mappings on variables of $\mathsf{Variables}$ by Maps, and denote by $\mathsf{Maps}(d)$ the set of all mappings on a document d, i.e., the mappings of Maps such that all spans in the image are spans of d. A mapping m is called *compatible* with a mapping m', in symbols $m \sim m'$, if for all $x \in \mathsf{dom}(m) \cap \mathsf{dom}(m')$ we have $m(x) = m'(x)$.

A *spanner* is a function mapping every document d to a finite set of mappings whose spans are over d, i.e., are included in $[0, |d|\rangle$. For a spanner P, we denote by $\mathsf{SVars}(P)$ the variables appearing in the domain of at least one of its mappings, formally $\mathsf{SVars}(P) := \{x \in \mathsf{Variables} \mid \exists d \in \Sigma^*, \exists m \in P(d), x \in \mathsf{dom}(m)\}$. A spanner P is *schema-based* if all its output mappings assign exactly the variables of $\mathsf{SVars}(P)$, i.e., for every $d \in \Sigma^*$ and $m \in P(d)$, we have $\mathsf{dom}(m) = \mathsf{SVars}(P)$. Otherwise, P is called *schemaless* [21], or *incomplete* [18]. We say a spanner P *accepts* or *captures* a mapping $m \in \mathsf{Maps}$ on a document $d \in \Sigma^*$ if $m \in P(d)$.

Variable-set automata. We focus mostly on the *regular spanners*, that can be expressed using *variable-set automata* (or VAs). These are intuitively nondeterministic automata where each transition is labeled either by a letter or by a *marker* indicating which variable is opened or closed. Formally, for a set X of variables, we denote by $\mathsf{markers}(X)$ the set of *markers* over X: it contains for each variable $x \in X$ the *opening marker* $x\vdash$ and the *closing marker* $\dashv x$. Then, a VA on alphabet Σ is an automaton $\mathcal{A} = (Q, q_0, F, \delta)$ where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ are the *final states*, and $\delta \subseteq Q \times (\Sigma \cup \mathsf{markers}(X)) \times Q$ is the *transition relation*: we write the transitions $q \rightarrow^{\sigma} q'$ to mean that $(q, \sigma, q') \in \delta$. Note that the transitions contain both *letter transitions*, labeled by letters of Σ, and *marker transitions*, labeled by markers of $\mathsf{markers}(X)$.

A *run* of $\mathcal{A}$ on a document $d \in \Sigma^*$ is a sequence $\rho : q_0 \rightarrow^{\sigma_1} q_1 \cdots q_{n-1} \rightarrow^{\sigma_n} q_n$ such that the restriction of $\sigma_1 \ldots \sigma_n$ to the letters of Σ is exactly d; it is *accepting* if we have $q_n \in F$. We say that ρ is *valid* if, for each variable $x \in X$, either the markers $x\vdash$ and $\dashv x$ do not occur in $\sigma_1 \cdots \sigma_n$, or they occur exactly once and $x\vdash$ occurs before $\dashv x$. We say that $\mathcal{A}$ is *sequential* if all its accepting runs are valid. In this paper, we always assume that VAs are sequential, and only speak of VAs to mean sequential VAs. The run ρ then defines a mapping m on d by intuitively assigning the variables for which markers are read to the span delimited by these markers. Formally, we associate to each index $0 \leqslant k \leqslant n$ of the run a position $\pi(k)$ in d by initializing $\pi(0) := 0$ and setting $\pi(k+1) := \pi(k)$ if the transition $q_k \rightarrow^{\sigma_{k+1}} q_{k+1}$ reads a marker, and $\pi(k+1) := \pi(k) + 1$ if it reads a letter; note that $\pi(n) = |d|$. Then, for each variable x whose markers are read in ρ, letting $\sigma_i = x\vdash$ and $\sigma_j = \dashv x$ with $i < j$ because the run is valid, we set $m(x) := [\pi(i), \pi(j)\rangle$.

A sequential VA $\mathcal{A}$ thus defines a spanner $P_{\mathcal{A}}$ that maps each document d to the set $P_{\mathcal{A}}(d)$ of mappings obtained from its accepting runs as we explained. Note that different accepting runs may yield the same mapping. We sometimes abuse notation and identify VAs with the spanners that they define. The *regular spanners* are those that can be defined by VAs, or, equivalently [18], by sequential VAs. A sequential VA is *functional* if it defines a schema-based spanner, i.e., every mapping assigns every variable that occurs in the transitions of the VA.

Regex formulas. Our examples of spanners in this paper will be given not as VAs but in the more human-readable formalism of *regex formulas*. The *regex formulas* over an alphabet Σ are the expressions defined inductively from the empty set $\emptyset$, empty word ε, and single letters $a \in \Sigma$, using the three regular operators of disjunction $(e_1 \vee e_2)$, concatenation (e_1e_2), and Kleene star (e^*), along with *variable captures* of the form $x\{e_1\}$ where x is a variable. A regex-formula r on a document $d \in \Sigma^*$ defines a spanner on the variables occurring in r. Intuitively, every match of r on d yields a mapping where the variables are assigned to well-nested spans following the captures; see [10] for details. We require of regex-formulas that, on every document $d \in \Sigma^*$, they assign each variable at most once; but we allow them to define schemaless spanners, i.e., they may only assign a subset of the variables.

It is known that regex formulas capture a strict subset of the regular spanners; see [10] in the case of schema-based spanners and [18] for the case of schemaless spanners.

Cartesian Products. Given two spanners P_1 and P_2 where $X_1 = \mathsf{SVars}(P_1)$ and $X_2 = \mathsf{SVars}(P_2)$ are disjoint, the *Cartesian product* $P_1 \times P_2$ of P_1 and P_2 is the spanner on variables $X_1 \cup X_2$ which on every document d captures the mappings $(P_1 \times P_2)(d) := P_1(d) \times P_2(d)$. Here, we interpret a pair $(m_1, m_2) \in P_1(d) \times P_2(d)$ as the merge of the two mappings, i.e., the mapping defined according to m_1 on X_1 and according to m_2 on X_2. If P_1 and P_2 are given as sequential VAs, then one can compute in polynomial time a sequential VA for $P_1 \times P_2$.

3 The Skyline Operator

In this paper, we define and study a new operator called the *skyline operator*. Its goal is to only extract mappings that contain the maximum amount of information in a certain sense.

Domination relations. We begin by defining *domination relations* which describe how to compare the information given by two mappings on a given document d.

▶ **Definition 3.1.** *A* pre-domination relation $\preccurlyeq$ *is a binary relation on the set of mappings* $\mathsf{Maps}(d)$ *of d. We say that it is a* domination relation *if it is a (non-strict) partial order, i.e., it is reflexive, transitive, and antisymmetric. For $m_1, m_2 \in \mathsf{Maps}$, we say that m_2* dominates *m_1 if $m_1 \preccurlyeq m_2$, and write $m_1 \not\preccurlyeq m_2$ otherwise.*

The goal of the domination relation is to define which mappings are preferred to others, intuitively because they contain more information; it may depend on the document, though we will present many examples where it does not.

We introduce several domination relations that, as discussed in the introduction, are part of practical systems and which we consider throughout this paper:

▶ **Definition 3.2.** *The simplest relation is the trivial* self domination *relation $\preccurlyeq_{self}$ where every mapping only dominates itself, i.e., the pairs in the relation are (m, m) for $m \in \mathsf{Maps}$.*

▶ **Definition 3.3.** *The* variable inclusion relation *$\preccurlyeq_{varInc}$ contains the pairs (m_1, m_2) such that for all $x \in \mathsf{Variables}$, if $m_1(x)$ is defined, then $m_2(x)$ is defined as well and $m_1(x) = m_2(x)$. Thus, we have $m_1 \preccurlyeq_{varInc} m_2$ whenever $\mathsf{dom}(m_1) \subseteq \mathsf{dom}(m_2)$ and $m_1 \sim m_2$, i.e., when m_2 is an extension of m_1 that potentially assigns more variables than m_1.*

▶ **Definition 3.4.** *The* span inclusion *relation $\preccurlyeq_{spanInc}$ contains the pairs (m_1, m_2) of mappings with the same domain ($\mathsf{dom}(m_1) = \mathsf{dom}(m_2)$) such that for every $x \in \mathsf{dom}(m_1)$ the span $m_1(x)$ is included in $m_2(x)$. Intuitively, m_1 and m_2 match the same variables in the same parts of a document, but the matches of variables in m_1 are subwords of their matches in m_2.*

▶ **Definition 3.5.** *The* left-to-right relation $\preccurlyeq_{ltr}$ *contains the pairs* (m_1, m_2) *of mappings with the same domain such that, for every variable* x *on which* m_1 *and* m_2 *are defined, the spans* $m_1(x)$ *and* $m_2(x)$ *start at the same position but* $m_2(x)$ *is no shorter than* $m_1(x)$.

▶ **Definition 3.6.** *The* span length relation $\preccurlyeq_{spanLen}$ *contains the pairs* (m_1, m_2) *of mappings with the same domain where for every* $x \in \mathsf{dom}(m_1)$ *the span* $m_2(x)$ *is no shorter than* $m_1(x)$. *Intuitively,* $\preccurlyeq_{spanLen}$ *prefers longer spans over shorter ones, anywhere in the document.*

Domination rules. We now introduce *domination rules* which associate to each document d a domination relation over d. In this paper, we express domination rules as spanners on specific domains. To this end, given a set of variables X, we write $X^\dagger$ to mean a set of annotated copies of the variables of X, formally $X^\dagger := \{x^\dagger \mid x \in X\}$. We extend the notation to mappings by defining $m^\dagger$ for a mapping m to be the mapping with domain $\mathsf{dom}(m^\dagger) = \mathsf{dom}(m)^\dagger$ such that for all $x \in \mathsf{dom}(m)$ we have $m^\dagger(x^\dagger) := m(x)$. We then define:

▶ **Definition 3.7.** *A* pre-domination rule D *on a set of variables* $X \subseteq \mathsf{Variables}$ *is a (schemaless) spanner with* $\mathsf{SVars}(D) \subseteq X \cup X^\dagger$. *For every document* $d \in \Sigma^*$, *we see* $D(d)$ *as a pre-domination relation* $\preccurlyeq$ *on* d *defined by the mappings captured by* D *on* d, *the left-hand-side and right-hand-side of the comparability pairs being the restrictions of the mappings to* X *and to* $X^\dagger$ *respectively. Formally, the relation* $\preccurlyeq$ *is:* $R := \{(m|_X, m') \mid m \in D(d), (m')^\dagger = m|_{X^\dagger}\}\}$.

We say that D *is a* domination rule *if, on every document* $d \in \Sigma^*$, *the pre-domination relation* R *defined above is a domination relation, i.e., it correctly defines a partial order.*

Intuitively, for every document d, the domination rule D defines the domination relation $\preccurlyeq$ where each mapping $m \in D(d)$ denotes a pair, i.e., the restriction of m to X is dominated by the restriction of m to $X^\dagger$ (renaming the variables from $X^\dagger$ to X). Note that pre-domination rules and pre-domination relations are just an intermediary notion; in the sequel, we only consider domination rules and domination relations.

▶ **Example 3.8.** For any set X of variables, each of the domination relations introduced in Definitions 3.2–3.5 can be defined by a domination rule expressed by a regular spanner on X (for the span length domination relation of Definition 3.6, see Lemma 3.13). At the end of the section, we explain how to express them in a more concise *variable-wise* way that does not depend on X: see Example 3.12.

The skyline operator. We have introduced domination rules as a way to define domination relations over arbitrary documents. We can now introduce the *skyline operator* to extract maximal mappings, i.e., mappings that are not dominated in the domination relation:

▶ **Definition 3.9.** *Given a domination rule* D, *the* skyline operator η_D *of* D *applies to a spanner* P *and defines a spanner* $\eta_D P$ *in the following way: given a document* d, *writing* $\preccurlyeq$ *to denote the domination relation* $D(d)$ *given by* D *on* d, *the result of* $\eta_D P$ *on* d *is the set of maximal mappings of* $P(d)$ *under the domination relation* $\preccurlyeq$. *Formally, we have:* $(\eta_D P)(d) := \{m \in P(d) \mid \forall m' \in P(d) \backslash \{m\} \colon m \not\preccurlyeq m'\}$.

Intuitively, the operator filters out the mappings that are dominated by another mapping according to the domination relation defined by the domination rule over the input document.

▶ **Example 3.10.** In Figure 1 we show the effect of the skyline operator with respect to some of our example domination relations. Assume that we are given a spanner P in variables $\{x, y\}$ that on a given document d extracts the mappings given in Figure 1a (here a dash "$-$"

$m(x)$	$m(y)$
$[1,2\rangle$	$[2,3\rangle$
$-$	$[2,3\rangle$
$[0,2\rangle$	$[2,3\rangle$
$[4,6\rangle$	$[4,10\rangle$

(a) The extracted mappings $P(d)$.

$m(x)$	$m(y)$
$[1,2\rangle$	$[2,3\rangle$
$[0,2\rangle$	$[2,3\rangle$
$[4,6\rangle$	$[4,10\rangle$

(b) Skyline under the variable inclusion relation.

$m(x)$	$m(y)$
$-$	$[2,3\rangle$
$[0,2\rangle$	$[2,3\rangle$
$[4,6\rangle$	$[4,10\rangle$

(c) Skyline under the span inclusion relation.

$m(x)$	$m(y)$
$-$	$[2,3\rangle$
$[4,6\rangle$	$[4,10\rangle$

(d) Skyline under the span length relation.

Figure 1 Extracted mappings before and applying different skyline operators; see Example 3.10.

means that the variable is not assigned by a mapping). We show the result of applying the skyline operators with (possibly non-regular) domination rules defining the variable inclusion domination relation $\preccurlyeq_{varInc}$ (Figure 1b), the span inclusion domination relation $\preccurlyeq_{spanInc}$ (Figure 1c), and the span length domination relation $\preccurlyeq_{spanLen}$ (Figure 1d). Note that, for the variable inclusion domination rule, the skyline only makes sense for schemaless spanners, as two distinct mappings that assign the same variables are always incomparable.

Variable-wise rules. We have defined our skyline operator relative to domination rules expressed as spanners on explicit sets of variables. However, it will often be convenient to define the rules as products of rules on a single variable by applying the product operator. This ensures that the rule is "symmetric" in the sense that all variables behave the same:

▶ **Definition 3.11.** *Let D be a domination rule in a single variable x, i.e., a spanner using variables of $\{x, x^\dagger\}$, which we call a* single-variable domination rule. *For $y \in$ Variables, we let D^y be the domination rule where we replace x and $x^\dagger$ by y and $y^\dagger$, i.e., on every document d, the set of mappings $D^y(d)$ consists of one mapping m^y for each mapping $m \in D^y(d)$ with $m^y(y)$ and $m^y(y^\dagger)$ defined like $m(x)$ and $m(x^\dagger)$.*

The variable-wise domination rule *defined by D on a variable set X is then simply $\bigtimes_{y\in X} D^y$. A domination rule is said to be* variable-wise *if it can be expressed in this way.*

We will often leave the set of variables X implicit, and may abuse notation to identify single-variable domination rules with the variable-wise domination rule that they can define on an arbitrary variable set.

▶ **Example 3.12.** The self domination rule (Definition 3.2) is variable-wise, because it can be obtained from the following trivial single-variable domination rule:

$$D_{self} = \Sigma^* x^\dagger\{x\{\Sigma^*\}\}\Sigma^* \vee \Sigma^*.$$

The Σ^* term above is used to ensure reflexivity and express the vacuous domination relation between the mapping where x is not assigned and the mapping where $x^\dagger$ is not assigned.

The span inclusion domination rule, left-to-right domination rule, and variable inclusion domination rule (Definitions 3.3–3.5) are also variable-wise with the single-variable rules:

$$\begin{aligned} D_{spanInc} &= \Sigma^* x^\dagger\{\Sigma^* x\{\Sigma^*\}\Sigma^*\}\Sigma^* \vee \Sigma^*. \\ D_{ltr} &= \Sigma^* x^\dagger\{x\{\Sigma^*\}\Sigma^*\}\Sigma^* \vee \Sigma^*. \\ D_{varInc} &= \Sigma^* x^\dagger\{\Sigma^*\}\Sigma^* \vee D_{self}. \end{aligned}$$

Here, $\Sigma^* x^\dagger\{\Sigma^*\}\Sigma^*$ expresses that assigning a variable is better than not assigning it.

As for the span length domination rule (Definition 3.6), it is also variable-wise, but a standard pumping argument shows that it cannot be defined by a regular spanner:

▶ **Lemma 3.13.** *The single-variable span length domination rule $D_{spanLen}$ is not expressible as a regular spanner.*

4 Closure under the Skyline Operator

We have defined the skyline operator relative to domination rules expressed by regular spanners. One natural question is then to understand whether the skyline operator under such rules extends the expressive power of spanner formalisms, or whether it can be defined in existing models. This is what we investigate in this section.

Regular spanners. We first focus on regular spanners, and show that they are closed under the skyline operator for domination rules expressed as regular spanners. We do so by showing how the skyline operator can be expressed with operations under which regular spanners are closed, namely join, intersection and difference (see Appendix A for definitions).

▶ **Theorem 4.1.** *There is an algorithm that, given a sequential VA defining a regular spanner P and a sequential VA defining a domination rule D, computes a sequential VA for $\eta_D P$.*

Theorem 4.1 generalizes a result of Fagin et al. [11, Theorem 5.3] on the expressiveness of transitive "denial pgds." In our terminology, their theorem states that the class of *complete* regular spanners is closed under the restriction to maximal answers defined by a regular domination rule. Theorem 4.1 thus extends their result to schemaless regular spanners.

Theorem 4.1 implies that taking the skyline relative to regular domination rules does not increase the expressivity of regular spanners. However, like the result of [11], our construction may compute VAs that are exponentially bigger than the input VA. In Section 5, we will see that this is unavoidable for any sequential VA expressing the skyline.

As an application of Theorem 4.1 we get in particular that regular spanners are closed under the skyline operator for most of the examples presented earlier, i.e., Definitions 3.2–3.5.

▶ **Corollary 4.2.** *There are algorithms that, given a sequential VA P, compute sequential VAs for $\eta_{self} P$, $\eta_{varInc} P$, $\eta_{ltr} P$, and $\eta_{spanInc} P$, respectively.*

By contrast, Theorem 4.1 does not apply to the span-length domination rule, as it is not expressible as a regular spanner (Lemma 3.13). In fact, we can show that taking the skyline under this domination rule is generally *not* expressible as a regular spanner:

▶ **Proposition 4.3.** *There is a sequential VA P such that $\eta_{spanLen} P$ is not regular.*

Other spanner formalisms. It is natural to ask whether closure results such as Theorem 4.1 also hold for other spanner formalisms. In particular, we can ask this for the language of *core spanners*, which extend regular spanners with string equalities; see [10] for the precise definitions and [24] for the schemaless case. We can show that core spanners, contrary to regular spanners, are *not* closed under the skyline operator:

▶ **Theorem 4.4.** *The core spanners are not closed under the skyline operator with respect to the span inclusion domination rule $D_{spanInc}$, even on schema-based spanners: there is a schema-based core spanner P such that $\eta_{spanInc} P$ cannot be expressed as a core spanner. The same is true of the skyline η_{varInc} with the variable inclusion domination rule.*

This result was already shown in [11] for the span inclusion domination rule, but that result only showed inexpressibility as a schema-based core spanner. Our result extends to the schemaless setting, and also establishes the result for the variable inclusion domination rule. See the full version [4] for the formal definitions and the proof.

We leave open the question of extending other formalisms with the skyline operator, e.g., the *generalized core spanners* which extend core spanners with the difference operator [22], or the *context-free spanners* [20] that define spanners via context-free grammars. Note that, by contrast, closure is easily seen to hold in the formalism of *RGXlog* programs, where spanners are defined using Datalog rules [22]. Indeed, this class consists of precisely the polynomial-time spanners (under data complexity). Thus, for any domination rule D for which the maximal answers can be computed in polynomial time data complexity (in particular, for domination rules expressed as regular spanners), the result of the skyline operator for D on an RGXlog program can be expressed as an RGXlog program.

In the rest of this paper, we focus on applying the skyline operators to regular spanners, with domination relations also defined via regular domination rules.

5 State Complexity of the Skyline Operator

We have seen how the skyline operator does not increase the expressive power of regular spanners, in the sense that it could be expressed using regular operations. However, this does not account for the price of this transformation. In this section, we show that the size of sequential VAs for some domination rules increases exponentially when applying the skyline operator. Hence, we essentially study the state complexity of VAs under the skyline operator, similarly to traditional studies of the state complexity for regular languages (e.g., [15]). Specifically, we show the following lower bound, for the variable inclusion domination rule:

▶ **Theorem 5.1.** *For every $n \in \mathbb{N}$, there is a sequential VA $\mathcal{A}$ with $O(n)$ states such that, letting $P_\mathcal{A}$ be the regular spanner that it defines, any sequential VA representing the regular spanner $\eta_{varInc} P_\mathcal{A}$ must have $2^{\Omega(n)}$ states.*

We will show in later sections how this lower bound on the state complexity of the skyline operation can be complemented with computational complexity lower bounds.

Proof technique: Representing Boolean functions as VAs. We show Theorem 5.1 using representations of Boolean functions as sequential VAs, as we now explain. Let $\mathsf{SVars} \subseteq \mathsf{Variables}$ be a finite set of variables (which will be used to define spanners), and let $\mathsf{Vars}_b := \{x_b \mid x \in \mathsf{SVars}\}$ be a set of Boolean variables. For every mapping m assigning spans to some of the variables in SVars (i.e., $\mathsf{dom}(m) \subseteq \mathsf{SVars}$), we define a Boolean assignment $m_b \colon \mathsf{Vars}_b \to \{0,1\}$ by setting $m_b(x_b) := 1$ if and only if $x \in \mathsf{dom}(m)$, i.e., x gets assigned a span by m. Let P be a document spanner with variables SVars and let d be an input document. Then we denote by $\mathrm{Bool}(P,d)$ the Boolean function whose models are $\{m_b \mid m \in P(d)\}$.

Our intuitive idea is that, if the function $\mathrm{Bool}(P,d)$ is hard to represent, then the same should be true of the spanner P. To make this precise, let us introduce the representations of Boolean functions that we work with:

▶ **Definition 5.2.** *A* nondeterministic read-once branching program[1] *(NROBP) over the variable set Vars_b is a tuple $\Pi = (G,s,t,\mu)$ where $G = (V,E)$ is a directed acyclic graph, $s \in V$ and $t \in V$ are respectively the* source *and* sink *nodes, and the function μ labels some of the edges with literals of variables in Vars_b, i.e., variables and their negations; formally μ is a partial function from E to the literals over Vars_b. We require that, for every source-sink path $s = v_0, \ldots, v_n = t$, every variable appears at most once in the literals labeling the edges of the path, i.e., there are no two indices $0 \leqslant i < j \leqslant n-1$ such that $\mu((v_i,v_{i+1}))$ and $\mu((v_j,v_{j+1}))$ are both defined and map to literals of the same variable.*

[1] We remark that what we introduce here are sometimes called *acyclic read-once switching and rectifier networks*, but theses are known to be equivalent to the more common definition of NROBPs up to constant factors [23], so we do not make the difference here.

An NROBP Π *computes a Boolean function over* Vars_b *whose models are defined in the following way. An assignment* $m_b\colon \mathsf{Vars}_b \to \{0,1\}$ *is a model of* Π *if there is a source-sink path in* G *such that all literal labels on the path are satisfied by* m_b*, i.e., there is a sequence* $s = v_0, \ldots, v_n = t$ *such that, for each* $0 \leqslant i < n$ *for which* $\ell := \mu((v_i, v_{i+1}))$ *is defined, then the literal* ℓ *evaluates to true according to* m_b*.*

NROBPs are intuitively similar to automata. To formalize this connection, we show how, given a sequential VA and document, we can efficiently compute an NROBP describing which subsets of the variables can be assigned in captured mappings:

▶ **Lemma 5.3.** *Let* P *be a regular spanner on variable set* SVars *represented by a sequential VA* $\mathcal{A}$ *with* n *states. Then, for every document* d*, there is an NROBP* G *representing* $Bool(P, d)$ *with* $O(|d| \times n \times |\mathsf{SVars}|)$ *nodes.*

Proof sketch. We compute the product of the VA with the input document, to obtain a directed acyclic graph representing the runs of the VA on the document. We obtain the NROBP by relabeling the marker transitions and performing some other modifications. ◀

We will now use the fact that NROBPs are exponentially less concise than other Boolean function representations. Namely, we define a *read-3 monotone 2-CNF formula* on a set of variables X as a conjunction of clauses which are disjunctions of 2 variables from X, where each variable appears at most 3 times overall. We use the fact that converting such formulas to NROBPs can incur an exponential blowup. This result is known (see, e.g., [7]) but we give a proof in the full version [4] for convenience:

▶ **Proposition 5.4** ([7]). *For any* $n \in \mathbb{N}$*, there is a read-3 monotone 2-CNF formula* Φ *on* n *variables having size* $O(n)$ *such that every representation of* Φ *as an NROBP has size* $2^{\Omega(n)}$*.*

We now conclude the proof of Theorem 5.1, sketched below (see the full version [4] for details):

Proof sketch. Given a read-3 monotone 2-CNF formula Φ, we show how to construct a regular spanner on which the skyline operator captures mappings corresponding precisely to the satisfying assignments of Φ. As a sequential VA expressing this spanner can be efficiently converted to an NROBP by Lemma 5.3, we can conclude that, when applied to the family of formulas from Proposition 5.4, all sequential VA representations have exponential size. ◀

An independent result: Lower bound on the state complexity of schema-less joins. We believe that the connection to Boolean functions used to show Theorem 5.1 can be of independent interest as a general technique to show lower bounds on the state complexity of document spanners. Indeed, independently from the skyline operator, we can also use this connection to show a lower bound on the size of sequential VAs representing the *natural join* of two regex-formulas. The *natural join operator* is a standard operator on spanners that merges together compatible mappings: see Appendix A for the formal definition. We have:

▶ **Theorem 5.5.** *For every* $n \in \mathbb{N}$*, there are regex-formulas* e_n *and* e'_n *of size* $O(n)$ *such that every sequential VA equivalent to* $e_n \bowtie e'_n$ *has* $2^{\Omega(n)}$ *states.*

This result is the counterpart for state complexity of the NP-hardness of evaluating the join of two regex-formulas [21]. It only holds in the schemaless case; indeed in the schema-based case it is known that the join of two functional VAs can be computed as a functional VA in polynomial time [13].

6 Complexity of the Skyline Operator

We have shown that the skyline operator applied to regular spanners cannot be expressed as a regular spanner without an exponential blowup in the size, even for domination rules expressed as regular spanners (namely, for the variable inclusion domination rule). We now study whether we can efficiently evaluate the skyline operator without compiling it into the automaton. Formally, we study its computational complexity of skyline extraction:

▶ **Definition 6.1.** *The* skyline extraction problem *is the following: given a document d, a sequential VA $\mathcal{A}$ capturing a regular spanner $P_{\mathcal{A}}$, and a domination rule D expressed as a sequential VA, compute the set of mappings in the results of the skyline operator $(\eta_D P_{\mathcal{A}})(d)$.*

Data complexity. We start by observing that skyline extraction is clearly tractable in the data complexity perspective in which d is the only input:

▶ **Proposition 6.2.** *For any fixed sequential VA $\mathcal{A}$ and domination rule expressed as a sequential VA D, the skyline extraction problem for $P_{\mathcal{A}}$ and D can be solved in polynomial time data complexity, i.e., in polynomial time in the input d.*

Proof. We use Theorem 4.1 to compute a sequential VA for $(\eta_D P_{\mathcal{A}})(d)$, which is independent from the input document d. We can then produce the mappings captured by this fixed sequential VA on d in polynomial data complexity.

Alternatively, without using the theorem, note that we can simply materialize the set of all captured mappings $(P_{\mathcal{A}})(d)$, in polynomial time because $\mathcal{A}$ is fixed. Then, for any pair of mappings, we can check if the domination relation holds using the domination rule D; this is again in polynomial time. We then return the set of maximal mappings in polynomial time. ◀

Note that this result would easily extend to fixed expressions using multiple skyline operators together with regular spanner operators, as all these operators are polynomial-time.

Combined complexity. We now turn to combined complexity settings in which the domination rule D and the spanner P are considered as part of the input. In fact, we will mostly consider the problem variant in which we fix a single-variable domination rule (e.g., variable inclusion), we take the skyline relative to the corresponding variable-wise domination rule, and only the spanner P is part of the input. Remember that we focus on regular spanners represented as sequential VAs, since for those it is known that the combined complexity of spanner evaluation is output polynomial [18].

As we have seen in Section 4, in terms of expressiveness, the regular spanners are closed under all domination rules expressible as regular spanners, in particular those of Definitions 3.2–3.5. However, we have seen in Section 5 that compiling the skyline into the VA may generally incur an exponential blowup, already for fixed domination rules. This bars any hope of showing tractability of the skyline extraction problem by applying known evaluation algorithms on the result of this transformation (e.g., those from [13, 12, 3]),

This leads to the question if there are other approaches to solve the skyline extraction problem with efficient combined complexity, without materializing an equivalent VA. In this section, we show that this is not the case, assuming $\mathsf{P} \neq \mathsf{NP}$. Our lower bound already holds for a fixed domination rule, namely, the variable inclusion domination rule; and in fact it even holds in *query complexity*, i.e., when the document is fixed.

▶ **Theorem 6.3.** *There is a fixed document d such that the following problem is* NP*-hard: given a sequential VA $\mathcal{A}$ encoding a regular spanner $P_{\mathcal{A}}$ and a number $n \in \mathbb{N}$ which is at most the size of $\mathcal{A}$, decide whether $(\eta_{varInc} P_{\mathcal{A}})(d)$ contains more than n mappings.*

This will imply that, conditionally, the skyline extraction problem is intractable in combined complexity. Intuitively, if it is intractable to decide whether the skyline extracts a large number of mappings, then producing the mappings is also intractable. To make this formal, we use the framework of *output-polynomial algorithms*, where an algorithm for a problem $f \colon \Sigma^* \to \Sigma^*$ runs in *output-polynomial time* if, given an input x, it runs in time polynomial in $|x| + |f(x)|$. Namely, we use the following folklore connection between output-polynomial time and decision problems, see e.g. [8] for a similar construction:

▶ **Lemma 6.4.** *Let $f : \Sigma^* \to \Sigma^*$ and let p be a polynomial. Assume that it is* NP*-hard, given an input x and integer $k \leqslant p(|x|)$, to decide if $|f(x)| < k$. Then there is no output polynomial time algorithm for f, unless* P = NP*.*

From Theorem 6.3 and Lemma 6.4, we directly get our intractability result:

▶ **Corollary 6.5.** *Unless* P = NP*, there is no algorithm for the skyline extraction problem with respect to the variable inclusion domination rule that is output-polynomial in combined complexity (i.e., in the input sequential VA), even when the input document is fixed.*

Note that this result is incomparable to Theorem 5.1: lower bounds on the size of equivalent VAs generally do not preclude the existence of other algorithms that are tractable in combined complexity, and conversely it could in principle be the case that evaluation is intractable in combined complexity but that there are small equivalent VAs that are intractable to compute. Besides, the proofs are also different. Namely, the proof of Theorem 5.1 used monotone 2-CNF formulas, for which we could compute spanners giving an exact representation of the satisfying assignments, but for which the satisfiability problem is tractable. As we will see, the proof of Theorem 6.3 uses the intractability of SAT on CNF formulas, but does not use an exact representation of the satisfying assignments.

Proving Theorem 6.3. We give the proof of Theorem 6.3 in the rest of this section, together with an additional observation at the end. In the next section, we will study how hardness can be generalized to other domination rules (in particular all domination rules introduced in Section 3 except the trivial self-domination rule), and will investigate the existence of tractable cases.

Proof of Theorem 6.3. We reduce from the satisfiability problem SAT. Let F be a CNF formula with n_x Boolean variables x_i with $i \in [n_x]$ and n_c clauses C_j with $j \in [n_c]$. For convenience, define the set $T_i = \{j \mid x_i \text{ appears positively in } C_j\}$, and define the set $F_i = \{j \mid x_i \text{ appears negatively in } C_j\}$. We will build a regular spanner on variables $v_{i,j}$ for $i \in [n_x]$ and $j \in [n_c]$, together with a special variable a.

We will define two spanners r_{valid} and r_{mask}, both as regex formulas, and will evaluate them on the empty document $d = \varepsilon$. Let us first sketch the idea: the spanner r_{valid} will extract one mapping for each possible assignment to the variables of F. Each such mapping will encode which clauses get satisfied by which variable in the assignment, by assigning spans to the corresponding spanner variables $v_{i,j}$. The second spanner r_{mask} will capture n_c additional mappings which will be maximal (thanks to the additional variable a) and will each dominate the mappings captured by r_{valid} for which the corresponding assignment does *not* satisfy a specific clause of F. This will ensure that F is satisfiable if and only if there are strictly more than n_c mappings in the skyline of $r_{\text{valid}} \vee r_{\text{mask}}$ on d.

Formally, we define the spanners as regex-formulas, where the dots denote concatenation:

$$r_{\text{valid}} = \cdot_{i \in [n_x]} ((\cdot_{j \in T_i} v_{i,j}\{\varepsilon\}) \vee (\cdot_{j \in F_i} v_{i,j}\{\varepsilon\})) \qquad r_{\text{mask}} = a\{\varepsilon\} \cdot \bigvee_{k \in [n_c]} \cdot_{i \in [n_x], j \in [n_c] \setminus \{k\}} v_{i,j}\{\varepsilon\}.$$

This definition is in polynomial time in the input CNF F.

Note that the mappings captured by r_{mask} are never dominated. First, they do not dominate each other: each of them assigns no $v_{i,k}$ for some k. Further, all mappings of r_{mask} assign a and all mappings of r_{valid} do not, so the latter cannot dominate the former.

To construct a CNF variable assignment from a mapping m captured by r_{valid}, we use the following encoding: if the mapping m assigns the span $[0, 0\rangle$ to the spanner variable $v_{i,j}$ then this encodes that the variable x_i appears in the clause C_j and x_i is assigned in a way that satisfies C_j. The definition of r_{valid} ensures that all variables appearing at least once will be assigned exactly one truth value among true or false.

We claim that on $d = \varepsilon$, the skyline $(\eta_{varInc}(r_{\text{valid}} \vee r_{\text{mask}}))(d)$ contains at least $n_c + 1$ mappings if and only if F is satisfiable. Assume first that F is satisfiable, and let v be a satisfying assignment. Then there is a corresponding mapping m captured by r_{valid} encoding v. Indeed, as v satisfies all clauses, for every clause index $j \in [n_c]$ there is a variable x_i assigned by v in a way that makes C_j true, i.e., $v_{i,j}$ is assigned. Hence m will not be dominated by any mapping captured by r_{mask}. Thus, the skyline of $r_{\text{valid}} \vee r_{\text{mask}}$ must contain some mapping captured by r_{valid}, namely, either m or some other mapping captured by r_{valid} which dominates m. In all cases, the skyline must have at least $n_c + 1$ mappings.

Now assume the skyline of $r_{\text{valid}} \vee r_{\text{mask}}$ has at least $n_c + 1$ mappings. By construction, r_{mask} captures exactly n_c maximal mappings, so there is at least one mapping m in the skyline which is captured by r_{valid}. This mapping m encodes an assignment v of the variables of F. As m is not dominated by any mapping captured by r_{mask}, for each clause index $j \in [n_c]$ there must exist a variable index $i \in [n_x]$ such that $v_{i,j}$ is assigned. Therefore v is a satisfying assignment of F. Overall, we have shown that F is satisfiable if and only if $\eta_{varInc}(r_{\text{valid}} \vee r_{\text{mask}})$ has at least $n_c + 1$ satisfying mappings, which concludes the proof. ◀

We last notice that we can modify Corollary 6.5 slightly: instead of applying to a fixed variable-wise domination rule (defined by fixing a single-variable domination rule), the result also applies when the domination rule is specified explicitly on the entire domain as a regular spanner:

▶ **Corollary 6.6.** *Assuming* P ≠ NP*, there is no algorithm for the skyline extraction problem which is output polynomial in combined complexity even if the domination rule is given as one sequential VA (not by implicitly taking the product of single-variable sequential VAs).*

7 Intractable and Tractable Domination Rules

We have shown that the skyline extraction problem is intractable in combined complexity for regular spanners, and this intractability already holds in the case of a fixed variable-wise domination rule, namely, the variable inclusion rule. However, this leaves open the same question for other domination rules, e.g., for the span inclusion rule – in particular if we restrict our attention to schema-based spanners, which are typically better-behaved (e.g., for the complexity of the join and difference operators [10]).

In this section, we show that, unfortunately, hardness still holds in that context. Specifically, we introduce a condition on domination rules, called having *unboundedly many disjoint strict domination pairs* (UMDSDP). This condition is clearly satisfied by our example domination rules (except self-domination). We then show that UMDSDP is a sufficient condition for intractability: this result re-captures the hardness of variable inclusion (Theorem 6.3) and also shows hardness for the span inclusion, left-to-right, and span length domination rules.

We then introduce a restricted class of domination rules, called *variable inclusion-like*, and show that on this class a variant of the UMDSDP condition in fact *characterizes* the intractable cases. In particular, all such domination rules without the condition enjoy tractable skyline extraction. Last, we study additional examples for general domination rules, and show that among rules not covered by UMDSDP, some are easy and some are hard.

The UMDSDP condition. To introduce our sufficient condition for intractability of skyline extraction, we first define *disjoint strict domination pairs.*

▶ **Definition 7.1.** *Fixing a variable x, a* single-variable mapping *is a mapping m using only the variable x: the mapping m may map x to a span $m(x)$, or it may not map x to anything, which is represented by the special symbol "$-$".*

For a domination relation $\preccurlyeq$ on a document d, a domination pair *of $\preccurlyeq$ on d is a pair (m_1, m_2) of single-variable mappings with $m_1 \preccurlyeq m_2$. The domination pair is* strict *if $m_1 \neq m_2$.*

Two strict domination pairs (m_1, m_2) and (m'_1, m'_2) are disjoint *if, letting s be the smallest span containing the spans $m_1(x)$ and $m_2(x)$ if defined, and letting s' be the smallest span containing the spans $m'_1(x)$ and $m'_2(x)$ if defined, then s and s' are disjoint spans. Otherwise, the two strict domination pairs* overlap.

Note that, in a strict domination pair (m_1, m_2), at least one of m_1 and m_2 has to assign x to a span; and if one of them does not, then the resulting unassigned span ("$-$") is not taken into account. In what follows, we abuse notation and identify single-variable mappings with the span to which they map x, or identify them to "$-$" if they do not map x to anything.

▶ **Example 7.2.** The pairs $([1, 3\rangle, [2, 4\rangle)$ and $([9, 10\rangle, [6, 8\rangle)$ are disjoint. The pairs $([1, 3\rangle, [7, 9\rangle)$ and $([4, 6\rangle, [10, 12\rangle)$ overlap (even though all of the constituent spans are disjoint). Finally, $(-, [1, 3\rangle)$ and $([4, 6\rangle, [10, 12\rangle)$ are also disjoint.

We can now define the UMDSDP condition, which will be sufficient to show hardness:

▶ **Definition 7.3.** *A single-variable domination rule D has* unboundedly many disjoint strict domination pairs *(UMDSDP) if, given $n \in \mathbb{N}$, we can compute in time polynomial in n a document $d \in \Sigma^*$ and n strict domination pairs $S_1, \ldots, S_n$ of $D(d)$ that are pairwise disjoint.*

▶ **Example 7.4.** D_{self} does not satisfy UMDSDP as it has no strict domination pairs.

The span length domination rule satisfies UMDSDP. Indeed, for $n \in \mathbb{N}$, we can take the word a^n and the disjoint strict domination pairs $\{([i, i\rangle, [i, i + 1\rangle) \mid i \in [0, n - 1]\}$. The same pairs show that UMDSDP holds for the span inclusion rule and for the left-to-right rule.

Finally, the variable inclusion domination rule satisfies UMDSDP with the set of pairs $\{(-, [i, i + 1\rangle) \mid i \in [0, n - 1]\}$.

Consider the domination rule D_{start} defining the domination relation $\preccurlyeq_{start}$ that contains the pairs $\{([1, i\rangle, [1, j\rangle) \mid i, j \in \mathbb{N}, i \leqslant j\}$ plus the trivial pair $(-, -)$ for reflexivity. Then $\preccurlyeq_{start}$ has unboundedly many strict domination pairs, but no two of them are disjoint, so the UMDSDP condition is not respected. (However, we will still be able to show intractability for this rule; see Proposition 7.9.)

We remark that, for single-variable domination rules that are regular, the UMDSDP condition holds whenever there *exist* arbitrarily many pairwise disjoint strict domination pairs (i.e., in this case we can always efficiently compute them); see the full version [4] for details.

UMDSDP implies hardness. We now show that the UMDSDP condition implies that skyline extraction is hard. The proof is a variant of the one for variable inclusion:

▶ **Theorem 7.5.** *Let D be a single-variable domination rule satisfying UMDSDP. The skyline extraction problem for D, given a sequential VA $\mathcal{A}$ and a document $d \in \Sigma^*$, is not output-polynomial unless* $\mathsf{P} = \mathsf{NP}$.

This implies the hardness of the other variable-wise domination rules presented earlier, completing Corollary 6.5. Note that these rules are schema-based spanners, and we can also notice that hardness already holds if the input spanner is functional, i.e., schema-based:

▶ **Corollary 7.6.** *There is no algorithm for the skyline extraction problem with respect to the span inclusion domination rule or the left-to-right domination rule or the span length domination rule which is output-polynomial in combined complexity, unless* $\mathsf{P} = \mathsf{NP}$. *This holds even if the input VA is required to be functional.*

Variable inclusion-like rules. We have seen that the UMDSDP condition is a sufficient condition for skyline extraction to be hard, but this leaves open the question of whether it is necessary. We will now focus on a fragment of domination rules which we call *variable inclusion-like domination rules*, where this is the case. Formally, we say that a domination relation $\preccurlyeq$ is *variable inclusion-like* if for all strict domination pairs (m_1, m_2) we have for all $x \in \mathsf{Variables}$ that if $m_1(x)$ is defined, then $m_2(x)$ is defined as well and $m_1(x) = m_2(x)$.

In contrast with the variable inclusion rule that contains all such pairs (m_1, m_2), we only require that a subset of them hold in $\preccurlyeq$. We will define variable inclusion-like domination rules in a variable-wise fashion: for single-variable variable inclusion-like rules, the strict domination pairs are necessarily of the form $(-, s)$ for a span s. In other words, a variable-wise inclusion-like domination rule is defined by indicating, on each document, which spans s can appear as the right-hand-side of such a pair. Further, for variable inclusion-like rules, two strict domination pairs are disjoint if and only if their right-hand-sides are.

We can show that, on variable inclusion-like domination rules, we have a dichotomy on a variant of the UMDSDP condition:

▶ **Theorem 7.7.** *Let D be a single-variable domination rule which is variable inclusion-like. If D satisfies the UMDSDP condition or accepts a pair of the form $(-, [i, i\rangle)$ on some document, then the skyline extraction problem for D, given a sequential VA and document, is not output-polynomial in combined complexity unless* $\mathsf{P} = \mathsf{NP}$. *Otherwise, the skyline extraction problem for D is output-polynomial in combined complexity.*

The lower bound of the dichotomy follows from Theorem 7.5, plus the observation that a single pair of the form $(-, [i, i\rangle)$ is sufficient to show hardness:

▶ **Lemma 7.8.** *Let D be a single-variable domination rule that accepts on some document a pair $(-, [i, i\rangle)$. Then the skyline extraction problem for D, given a sequential VA and document, is not output-polynomial in combined complexity unless* $\mathsf{P} = \mathsf{NP}$.

Hence, the interesting result in Theorem 7.7 is the upper bound. We show it in the full version [4] by observing that the set of right-hand-sides of strict domination pairs for variable inclusion-like rules that do not satisfy UMDSDP have bounded hitting set number, and showing that this implies tractability.

Other cases. Theorems 7.5 and 7.7 do not settle the complexity of non-UMDSDP domination rules which are not variable inclusion-like. We conclude with some examples of rules that can be shown to be intractable. We first show it for the rule $\preccurlyeq_{start}$ introduced earlier:

▶ **Proposition 7.9.** *Refer back to the variable-wise domination rule D_{start} from Example 7.4. There is no output-polynomial combined complexity algorithm for the skyline extraction problem for that rule, assuming* $\mathsf{P} \neq \mathsf{NP}$*.*

We show hardness for another rule that fails the UMDSDP, where all strict domination pairs share the same right-hand-side:

▶ **Proposition 7.10.** *Consider the variable-wise domination rule expressed by the regular expression $x\{a^*\}a^*x^\dagger\{b\} \vee D_{self}$. There is no output-polynomial combined complexity algorithm for the skyline extraction problem for that rule, assuming* $\mathsf{P} \neq \mathsf{NP}$*.*

We note, however, that the *reverse* of that rule, where all strict domination pairs share the same left-hand-side, is in fact tractable (and also fails the UMDSDP). This illustrates that, counter-intuitively, a complexity classification on variable-wise domination rules would not be symmetric between the left-hand-side and right-hand-side:

▶ **Proposition 7.11.** *The skyline extraction problem for the variable-wise domination rule $x^\dagger\{a^*\}a^*x\{b\} \vee D_{self}$ is output-polynomial in combined complexity.*

8 Conclusions

We have introduced the general framework of domination rules to express the skyline operator for document spanners, with rules that are themselves expressed as a spanner. We have shown that this operator (with regular rules) does not increase the expressiveness of regular spanners, but that it incurs an unavoidable exponential blowup in the state complexity and is intractable to evaluate in combined complexity for many natural fixed rules.

Our work leaves several questions open for future investigation. The most immediate question is whether the skyline extraction problem admits a dichotomy on the variable-wise regular domination rule in the general case, i.e., extending Theorem 7.5 to arbitrary such rules. However, this seems challenging. Another question is whether the hardness results of Section 7 also give state complexity lower bounds of the kind shown in Section 5, in particular in the schema-based context; and whether there is a dichotomy on state complexity.

Last, an intriguing question is whether the *top-k problem* of computing a constant number k of mappings from the skyline is always tractable in combined complexity. None of our hardness results precludes it, but we are not aware of an algorithm for that problem.

References

1 Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. In *ICDT*, 2012. `doi:10.1145/2274576.2274605`.

2 Shqiponja Ahmetaj, Wolfgang Fischl, Markus Kröll, Reinhard Pichler, Mantas Šimkus, and Sebastian Skritek. The challenge of optional matching in SPARQL. In *FoIKS*, 2016. `doi:10.1007/978-3-319-30024-5_10`.

3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, 2019. `doi:10.4230/LIPIcs.ICDT.2019.22`.

4 Antoine Amarilli, Benny Kimelfeld, Sébastien Labbé, and Stefan Mengel. Skyline operators for document spanners. *CoRR*, 2024. Full version of this article with all proofs. `doi:10.48550/arXiv.2304.06155`.

5 Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*. IEEE, 2001. doi:10.1109/ICDE.2001.914855.

6 Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.*, 21, 2004. doi:10.1613/jair.1234.

7 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. A strongly exponential separation of DNNFs from CNF formulas. *CoRR*, abs/1411.1995, 2014. doi:10.48550/arXiv.1411.1995.

8 Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discret. Appl. Math.*, 268, 2019. doi:10.1016/j.dam.2018.06.038.

9 Hamish Cunningham, Kevin Humphreys, Robert J. Gaizauskas, and Yorick Wilks. GATE – A general architecture for text engineering. In *ANLP*. ACL, 1997. doi:10.3115/974281.974299.

10 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015. doi:10.1145/2699442.

11 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative cleaning of inconsistencies in information extraction. *ACM Trans. Database Syst.*, 41(1), 2016. doi:10.1145/2877202.

12 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, 2018. doi:10.1145/3196959.3196987.

13 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, 2018. doi:10.1145/3196959.3196967.

14 César A. Galindo-Legaria. Outerjoins as disjunctions. *SIGMOD Rec.*, 23(2), 1994. doi:10.1145/191843.191908.

15 Yuan Gao, Nelma Moreira, Rogério Reis, and Sheng Yu. A survey on operational state complexity. *J. Autom. Lang. Comb.*, 21(4):251–310, 2017. doi:10.25596/jalc-2016-251.

16 Benoit Groz and Tova Milo. Skyline queries with noisy comparisons. In *PODS*, 2015. doi:10.1145/2745754.2745775.

17 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, 2011. URL: https://aclanthology.org/P11-4019/.

18 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, 2018. doi:10.1145/3196959.3196968.

19 Liat Peterfreund. *The Complexity of Relational Queries over Extractions from Text*. PhD thesis, Technion - Computer Science Department, 2019. URL: https://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2019/PHD/PHD-2019-10.

20 Liat Peterfreund. Grammars for document spanners. In *ICDT*, 2021. doi:10.4230/LIPIcs.ICDT.2021.7.

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

22 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *ICDT*, 2019. doi:10.4230/LIPIcs.ICDT.2019.13.

23 Igor Razgon. On the read-once property of branching programs and CNFs of bounded treewidth. *Algorithmica*, 75(2), 2016. doi:10.1007/s00453-015-0059-x.

24 Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *ICDT*, 2021. doi:10.4230/LIPICS.ICDT.2021.4.

25 Cheng Sheng and Yufei Tao. Worst-case I/O-efficient skyline algorithms. *ACM Transactions on Database Systems (TODS)*, 37(4), 2012. doi:10.1145/2389241.2389245.

26 Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3), 2012. doi:10.1007/s10472-012-9288-8.

A Spanner Algebra

In this appendix, we introduce some operators on spanners that are used in the main text.

For every spanner P and every subset $Y \subseteq \mathsf{SVars}(P)$, we define the projection operator π_Y by saying that $\pi_Y P$ is the spanner that extracts on every document d the set $(\pi_Y P)(d) := \{m|_Y \mid m \in P(d)\}$ where $m|_Y$ is the restriction of m to Y.

The *natural join* $P_1 \bowtie P_2$ of two spanners P_1 and P_2 is a spanner which accepts all the mappings m which are the union of two compatible mappings m_1 accepted by P_1 and m_2 accepted by P_2. Said differently, $(P_1 \bowtie P_2)(d) := \{m \in \mathsf{Maps} \mid \exists m_1 \in P_1(d), \exists m_2 \in P_2(d), m_1 \sim m \wedge m_2 \sim m \wedge \mathsf{dom}(m_1) \cup \mathsf{dom}(m_2) = \mathsf{dom}(m)\}$.

We remark that if $\mathsf{SVars}(P_1) \cap \mathsf{SVars}(P_2) = \varnothing$ then the join operator is the Cartesian product defined before.

The intersection operator $\cap$ is defined to compute the spanner $P_1 \cap P_2$ which on every document computes the set $(P_1 \cap P_2)(d) := P_1(d) \cap P_2(d)$. Observe that that if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$ and both spanners are schema-based, then the join operator is the intersection: $(P_1 \bowtie P_2)(d) = P_1(d) \cap P_2(d)$.

The union $P_1 \cup P_2$ is defined to as the spanner which on every document computes the set $(P_1 \cup P_2)(d) := P_1(d) \cup P_2(d)$.

The *difference*, $P_1 - P_2$ is a binary operator which accepts all mappings accepted by P_1 which are not accepted by P_2. Said differently, $(P_1 - P_2)(d) := P_1(d) \backslash P_2(d)$. Note that this is the usual difference operator on sets, and *not* the difference operator defined in [19] which accepts mappings of P_1 for which no compatible mapping is accepted by P_2.

It is known that the projection, natural join operator and union operators do not increase the expressive power of regular spanners, see [10] for the case of schema-based spanners and [18] for schemaless spanners. It follows that the same is true for the Cartesian product operator. We show in the full version [4] that intersection does not increase the expressivity, either. As for the difference operator, the same result is proven in [10] for schema-based regular spanners, but we are not aware of the same result for schemaless spanners and for our semantics of difference, so we prove it in the full version [4].